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

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Моделирование статической сцены расстановки шахматных фигур на шахматной доске»

Студент ИУ7-53Б (Группа) _____ (Подпись, дата) Конкина А. Н. (И. О. Фамилия)

Руководитель курсовой работы _____ (Подпись, дата) Мартынюк Н. Н. (И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Компьютерная графика – совокупность методов и средств преобразования в графическую форму и из графической формы с помощью ЭВМ [1]. Одной из задач компьютерной графики является синтез изображения [1], который состоит из следующих этапов.

1) разработка трёхмерной математической модели синтезируемой визуальной обстановки, задание положения наблюдателя, картинной плоскости, размеров окна вывода, значений управляющих сигналов, определение операторов, осуществляющих пространственное перемещение объектов визуализации;

2) преобразования координат объектов в координаты наблюдателя, отсечение объектов сцены по границам пирамиды отсечения (видимости), вычисление двумерных перспективных проекций объектов сцены на картинную плоскость, удаление невидимых линий и поверхностей, при заданном положении наблюдателя, закрашивание и затенение видимых объектов сцены;

3) вывод полученного полутонового изображения на экран растрового дисплея.

Целью данной работы является разработка программного обеспечения с пользовательским интерфейсом, реализовывающего моделирование статической сцены расстановки шахматных фигур на шахматной доске.

Для достижения поставленной цели необходимо выполнить следующие задачи.

— проанализировать предметную область, рассмотреть известные подходы и алгоритмы решения задачи синтеза изображения в контексте моделирования статической сцены;

— спроектировать программное обеспечение;

— выбрать средства реализации и разработать программное обеспечение;

— исследовать характеристики разработанного программного обеспечения.

1 Аналитическая часть

В данном разделе будут формализованы объекты синтезируемой сцены, определены их геометрические, оптические характеристики, проанализированы существующие алгоритмы, решающие задачу синтеза сложного изображения в контексте моделирования статической сцены расстановки шахматных фигур на шахматной доске.

1.1 Формализация объектов статической сцены

Сцена состоит из следующего набора объектов.

1) Шахматная доска – правильный параллелепипед, на одной из граней которой расположено шахматное поле из 8x8 ячеек в соответствии с правилами ФИДЕ. Размеры шахматной доски по умолчанию константно заданы в программе.

2) Шахматные фигуры – модели, занимающие ячейки шахматного поля. Параметры шахматных фигур являются константными и задаются на основе заранее созданных сторонними пакетами моделей. Положение и количество не обязательно соответствует правилам шахматной игры, иными словами, в программном обеспечении будет отсутствовать функционал проверки корректности расположения шахматных фигур в соответствии с правилами игры. Оптические характеристики материала фигур определяются задаваемыми пользователями параметрами.

3) Точечный источник света - имеет заданное положение в пространстве и равномерно излучает во всех направлениях, а интенсивность освещения спадает с расстоянием [1].

4) Наблюдатель (камера) характеризуется своим местоположением и направлением взгляда [1], направлением оси вверх.

В рамках технического задания рассматривается пространственное перемещение только таких объектов, как шахматная фигура (а именно, перенос для её постановки в заданную клетку шахматной доски), камера (перенос, поворот), точечный источник света (перенос). Иных пространственных преобразований не предусмотрено.

1.2 Анализ способов задания трехмерных моделей

Модели являются отображением формы и размеров объектов [1]. Основное назначение модели – правильно отображать форму и размеры определенного объекта [1].

Существуют три основных формы моделей [1].

1) Каркасная (проволочная) модель, которая задает информацию о вершинах и ребрах объекта. Модель не всегда точно передает форму объекта [1].

2) Поверхностная модель. Поверхность может быть описана аналитически или задана другим способом, например, через отдельные участки поверхности, однако в данной модели не учитывается, с какой стороны находится материал поверхности [1].

3) Объёмные (твёрдотельные) модели отличаются от поверхностных моделей тем, что к информации о поверхностях добавляется информация о расположении материала, например через хранение информации о направлении внутренней нормали поверхности [1].

В связи с тем, что каркасная модель не всегда точно передаёт форму объекта, а поверхностная модель не хранит информацию о том, с какой стороны находится материал поверхности, то при разработке программного обеспечения будет использована объёмная модель.

1.3 Способ хранения заранее созданных моделей

Ниже представлены, согласно источнику [2], наиболее распространённые форматы представления трёхмерных моделей в виде двоичных или текстовых файлов.

1) OBJ — текстовый формат файлов описания геометрии. Формат файла OBJ хранит информацию о трёхмерных моделях, может кодировать геометрию поверхности 3D-модели, а также хранить информацию о цвете и текстуре [2].

2) 3DS – двоичный формат файлов, используемых программой анимации и рендеринга. Содержит информацию об объектах, представленных в виде сеток их кинематику (анимацию), а также данные об источниках освещения.

3) STL — содержит минимум для описания трёхмерных моделей. Информация об объекте хранится как список треугольных граней, которые описывают его поверхность, и их нормалей.

Поскольку форматы представления будут использоваться только для хранения заранее созданных трёхмерных моделей шахматных фигур, а оптические характеристики материала фигур определяются задаваемыми пользователями параметрами, то нет необходимости хранения излишней информации, помимо поверхностей и их нормалей, согласно определению объёмной модели.

Таким образом, для хранения заранее созданных сторонними пакетами моделей будет использоваться формат STL.

1.4 Изменение местоположения в пространстве

Согласно техническому заданию необходимо изменять местоположение камеры и источника света. Иными словами, необходимо пользователю дать возможность, согласно источнику [1], выполнить перенос в отношении камеры и источника света, а также поворот в отношении камеры, что можно сделать, используя матрицы поворота и переноса в трёхмерном пространстве [1].

Формулы 1.1, 1.2, 1.3 – матрицы поворота вокруг соответствующих осей, а 1.4 – матрица переноса.

$$M_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{pmatrix} \tag{1.1}$$

$$M_y(\alpha) = \begin{pmatrix} cos(\alpha) & 0 & sin(\alpha) \\ 0 & 1 & 0 \\ -sin(\alpha) & 0 & cos(\alpha) \end{pmatrix} \tag{1.2}$$

$$M_z(\alpha) = \begin{pmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.3}$$

$$M(dx, dy, dz) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{pmatrix} \tag{1.4}$$

1.5 Модели освещения

1.5.1 Диффузное отражение

Матовые поверхности обладают свойством диффузного отражения, т. е. равномерного по всем направлениям рассеивания света, благодаря чему поверхности визуально имеют одинаковую яркость независимо от угла обзора [3]. Для таких поверхностей, согласно источнику [3], справедлив закон косинусов Ламберта, устанавливающий соответствие между количеством отраженного света и косинусом угла θ между направлением $\bar{L}$ на точечный источник света интенсивности I_p и нормалью $\bar{N}$ к поверхности (рисунок 1.1)

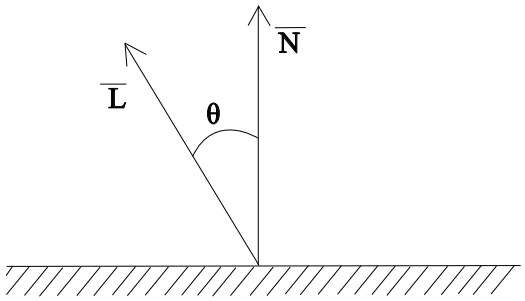

Рисунок 1.1 – Падающий свет и нормаль к поверхности

Интенсивность вычисляется по формуле 1.5, где K_d - коэффициент диффузного отражения, является константой в диапазоне (0, 1) и зависит от материала.

$$I_d = I_p \cdot K_d \cdot cos(\theta) \tag{1.5}$$

При этом количество отраженного света не зависит от положения наблюдателя [3].

Поскольку, даже если предмет защищен от прямых лучей, исходящих от точечного источника света, он все равно будет виден из-за наличия рассеянного

света, то формулу 1.5 можно модифицировать формулой 1.6, где рассеянный свет представлен членом I_a, а K_a определяет количество рассеянного света, которое отражается от поверхности предмета [3].

$$I_d = I_a \cdot K_a + I_p \cdot K_d \cdot cos(\theta) \tag{1.6}$$

$cos(\theta)$ можно выразить через нормаль к поверхности в точке $\bar{N}$ и исходный луч с направлением $\bar{L}$ следующим образом (формула 1.7) [4].

$$cos(\theta) =< \bar{N}, \bar{L} > /|\bar{N}|/|\bar{L}| \tag{1.7}$$

1.5.2 Зеркальное отражение

Зеркальное отражение можно получить от любой блестящей поверхности [3]. Блестящие поверхности отражают свет неодинаково по всем направлениям [3]. От идеального зеркала свет отражается только в том направлении, для которого углы падения и отражения совпадают. Это означает, что наблюдатель, с направлением взгляда $-\bar{V}$, сможет увидеть зеркально отраженный свет только в том случае, если угол α равен нулю (рисунок 1.2) [3].

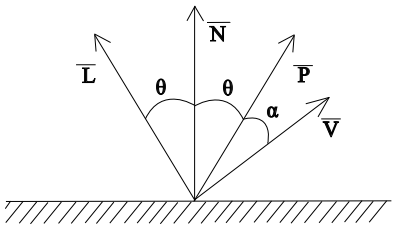

Рисунок 1.2 – Зеркальное отражение

В модели освещения предложенной Фонгом, быстрое убывание интенсивности описывается функцией $cos^n\alpha$, где n обычно лежит в диапазоне 1–200, в зависимости от вида поверхности [3]. В основе модели лежит эмпирическое наблюдение, а не фундаментальное понимание процесса зеркального отражения [3]. Таким образом, интенсивность зеркального отражения I_s определяется по формуле 1.8 [3].

$$I_s = I_p \cdot K_s \cdot cos^n\alpha \tag{1.8}$$

Направление отражённого света можно получить с использованием

формулы 1.9 [4].

$$\bar{R} = \bar{L} - 2 * \bar{N} \cdot dot(\bar{N}, \bar{L}) \tag{1.9}$$

$cos(\alpha)$ можно выразить направление отражённого луча и направления взгляда наблюдателя.

$$cos(\theta) =< \bar{R}, \bar{V} > /|\bar{R}|/|\bar{V}| \tag{1.10}$$

1.6 Алгоритм трассировки лучей

Алгоритм трассировки лучей позволяет решить все задачи второго этапа синтеза изображения [4]. Алгоритм основан на одном из положении геометрической оптики, согласно которому луч света распространяется прямолинейно до тех пор, пока не встретится отражающая поверхность или граница среды преломления [4]. Алгоритм состоит из следующих этапов.

1) от источников излучения исходит по различным направлениям бесчисленное множество первичных лучей;

2) один из лучей попадает на объект;

3) если объект зеркальный, то луч отражается, а часть световой энергии будет поглощена. Так, в результате воздействия на объекты первичных лучей, возникают вторичные лучи;

4) многократно преломляясь и отражаясь, отдельные световые лучи приходят в точку наблюдения.

Таким образом, изображение формируется некоторым множеством световых лучей.

В данном алгоритме используется [3] глобальная модель освещения Уиггеда, при использовании которой интенсивность некоторой точки объекта, с исходной интенсивность точки C, коэффициентом отражения $K_r = 1 - K_d$, коэффициентом преломления K_t, интенсивностями по отражённому I_r и преломлённому I_t лучам определяется суммарной интенсивностью, представленной в формуле 1.11.

$$I = I_d \cdot C + I_s + K_r \cdot I_r + K_t \cdot I_t \tag{1.11}$$

Интенсивность для рассеянного света обычно константа. Интенсивности излучений, проходящих по отражённому лучу и по преломлённому, умножают на коэффициент, учитывающий ослабление интенсивности в зависимости от расстояния, пройденного лучом $e^{-\beta d}$, где β – параметр ослабления, учитывающий свойство среды, в которой распространяется луч, а d – пройденное расстояние. Поскольку в техническом задании не упоминается преломление луча, то в дальнейшем $K_t = 0$.

1.6.1 Обратная трассировка лучей

Метод обратной трассировки лучей позволяет сократить перебор световых лучей, поскольку учитываются только те лучи, которые вносят вклад в формирование изображения [3]. Согласно этому методу отслеживание лучей производится не от источников света, а в обратном направлении – от точки наблюдателя (рисунок 1.3). Вводится такое понятие как экран - прямоугольник, размерностью эквивалентеный пользовательскому экрану.

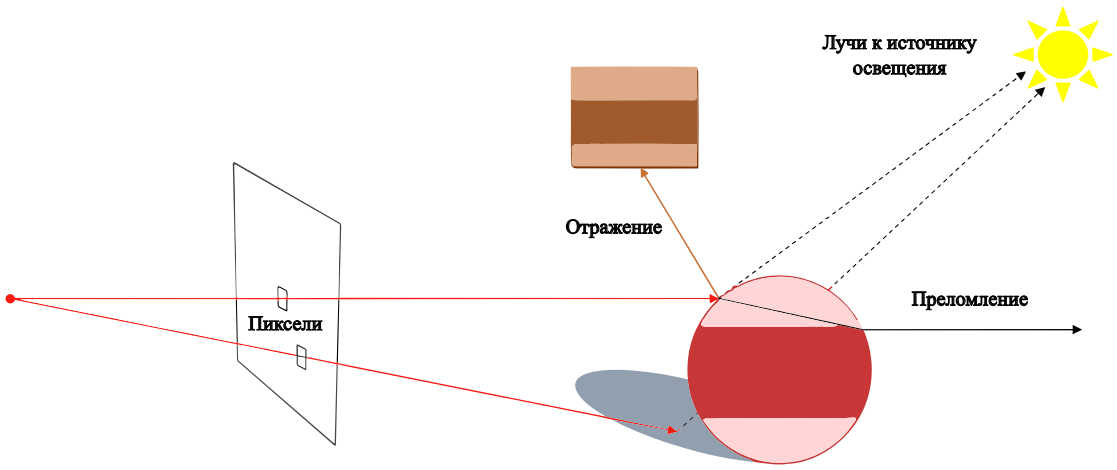

Рисунок 1.3 – Обратная трассировка лучей

Алгоритм состоит из следующих этапов.

1) от наблюдателя в каждую точку (пиксель) экрана проводится луч;

2) луч достигает объекта, преломляясь или отражаясь;

3) обработка луча прекращается, когда он перестаёт пересекать объекты сцены.

Обычно количество раз, которое луч может отразиться или переломиться ограничивают тремя значениями [3], что называется максимальной глубиной трассировки.

Согласно исследованию, указанному в источнике [5], метод показывает низкую производительность в сравнении с его модификациями по следующим причинам.

1) необходимо перебирать каждый объект сцены при поиске пересечения, таким образом временные затраты увеличиваются с ростом количества объектов сцены;

2) вычисления начинаются заново для каждого пикселя экрана.

1.6.2 Параллельная версия алгоритма обратной трассировки лучей

Параллельная версия алгоритма обратной трассировки лучей основывается на том наблюдении, что когерентные лучи можно трассировать вместе. Таким образом, имеет место быть многопоточность по данным, где в данного выступают пиксели экрана. Данный алгоритм комбинируется с другими методами [6].

1.6.3 Использование KD-деревьев для вычисления пересечений с поверхностями сцены

KD-дерево представляет собой бинарное дерево ограничивающих параллелепипедов (*англ.* BoundingBox), вложенных друг в друга [7]. Ограничивающий параллелепипед - параллелепипед, одними из вершин которого являются максимальные и минимальные компоненты координат модели. Каждую из граней можно представить как двумерный отсекатель Кируса-Бека для параметрически заданного отрезка [8] (рисунок 1.4).

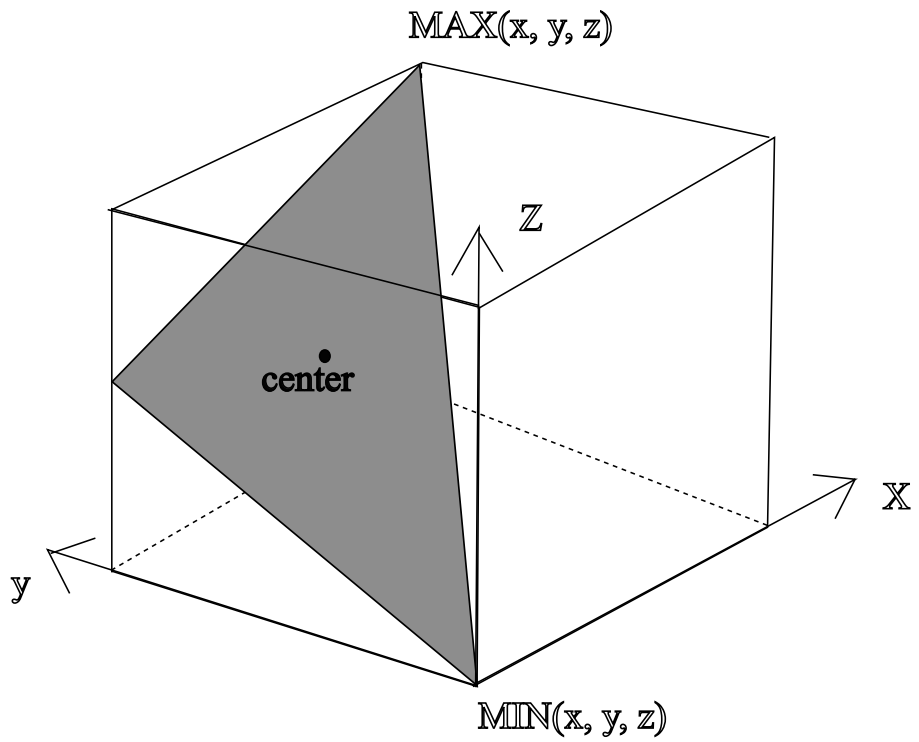

Рисунок 1.4 – Ограничивающий параллелепипед

Параметрическое уравнение отрезка с параметром t и точками P_1, P_2 имеет вид, представленный в формуле 1.12.

$$P(t) = P_1 + (P_2 - P_1) \cdot t \tag{1.12}$$

Соответственно, если $t < 0$, то точка P находится за точкой P_1, если $0 \leq t \leq 1$, то точка находится между точками P_1 и P_2, а если $t > 1$, то точка находится за P_2. Рисунок иллюстрирует зависимость расположения точек P от параметра t относительно точек P_1, P_2.

В случае обратного алгоритма трассировки лучей, рассматривается луч, точка $P_1 = O$ – местоположение камеры (O – origin), $(P_2 - P_1) = D$ – направление взгляда (D – direction).

В таком случае, уравнение 1.12 имеет следующий вид (формула 1.13).

$$P(t) = \bar{O} + \bar{D} \cdot t \tag{1.13}$$

В случае пересечения луча с параллелепипедом, поскольку известны вершины параллелепипеда, то t можно выразить, используя формулу 1.14.

$$t = (P - \bar{O})/\bar{D} \tag{1.14}$$

В случае же, если же направление вектора $\bar{D}$ отрицательно, то необходимо рассматривать обратное расположение вершин параллелепипеда [8].

Соответственно, поскольку находится пересечение с каждой гранью параллелепипеда, то будет получено до 6 различных пересечений с гранями, три из которых будет связано с минимальными координатами относительно направления луча, а три с максимальными. Необходимыми t будут максимальная среди трёх минимальных и минимальная среди трёх максимальных. Однако, существует два случая, когда луч не пересекает параллелепипед, которые можно выяснить при вычислении t.

1) оба t оказались отрицательными – луч находится за пределами параллелепипеда и направлен в противоположную сторону;

2) минимальное t оказалось больше максимального – луч находится за пределами параллелепипеда;

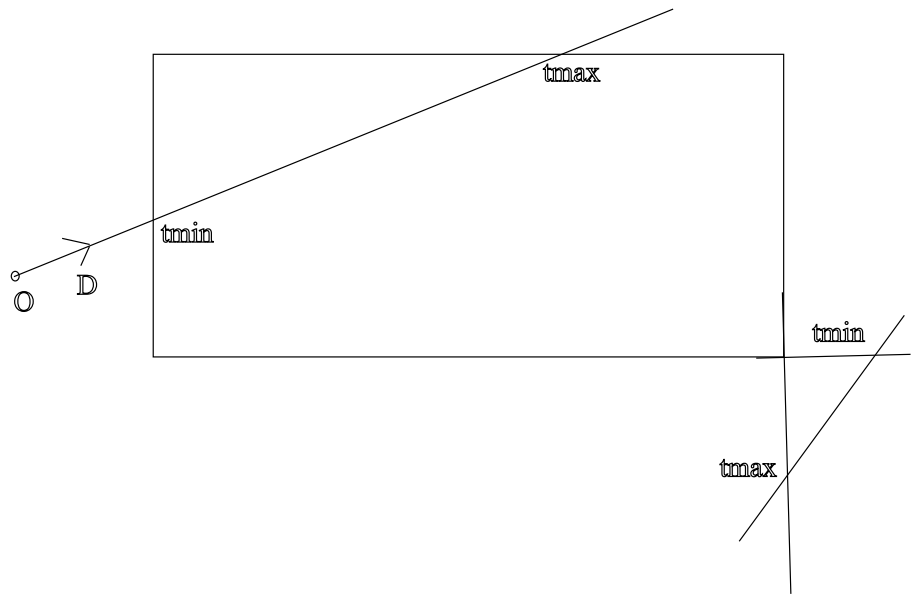

Рисунок 1.5 – Пример пересения луча с одной граней параллелепипеда

Для определения пересечения с треугольными гранями, используемых в STL представлении модели, можно использовать барицентрическую систему координат [7].

Пусть есть треугольник, заданный своими вершинами a, b и c. Тогда этот треугольник является выпуклой оболочкой этих точек. Это значит, что для любой точки p из этого треугольника всегда найдутся три числа u, v и w такие, что верная система уравнений 1.15 [7].

$$\begin{cases} p = a \cdot u + vb + w \cdot c \\ u, v, w > 0 \\ u + v + w = 1 \end{cases} \tag{1.15}$$

Числа u, v и w называются барицентрическими координатами точки p. При этом сами вершины треугольника будут иметь координаты (1, 0, 0),(0, 1, 0) и (0, 0, 1) [7]. Если точка p лежит строго внутри треугольника, то все компоненты ее барицентрических координат будут принадлежать интервалу (0, 1) [7]. У точек на ребрах одна из трех координат будет равной нулю [7].

Соответственно, если дана точка p, то её барицентрические координаты можно определить следующим образом.

$$w = 1 - u - v \tag{1.16}$$

$$p = u \cdot (a - c) + v \cdot (b - c) \tag{1.17}$$

Умножая уравнение 1.17 скалярно на $a - c$ и $b - c$, получится система алгебраических уравнений 1.18.

$$\begin{cases} u(a - c, a - c) + v(a - c, b - c) = (p - c, a - c) \\ u(a - c, b - c) + v(b - c, b - c) = (p - c, b - c) \end{cases} \tag{1.18}$$

Определитель системы 1.18 выражается формулой 1.19.

$$d = (a - c, a - c)(b - c, b - c) - (a - c, b - c)^2 \tag{1.19}$$

Решение системы определяется правилу Крамера (формула 1.20).

$$\begin{cases} u = 1/d \cdot ((p - c, a - c)(b - c, b - c) - (p - c, b - c)(a - c, b - c)) \\ v = 1/d \cdot ((p - c, a - c)(a - c, b - c) - (p - c, b - c)(a - c, a - c)) \end{cases} \tag{1.20}$$

Каждый параллелепипед в KD-дереве разбивается плоскостью, перпендикулярной одной из осей координат, на два дочерних параллелепипеда (условно, левый и правый) в зависимости от того, к какой из стороны разбие-

ния центр фигуры, заключённой внутри каждого параллелепипеда, находится ближе. Вся сцена целиком содержится внутри корневого параллелепипеда, но, продолжая рекурсивное разбиение параллелепипедов, можно прийти к тому, что в каждом листовом параллелепипеде будет содержаться небольшое число примитивов [7]. Таким образом, KD-дерево позволяет использовать бинарный поиск для нахождения примитива, пересекаемого лучом [7]. Недостатком алгоритма является трудоёмкость построения дерева [6].

1.6.4 Выбор методов и алгоритмов

Хотя при использовании KD-дерева необходимо учитывать трудоёмкость добавления элемента в дерево и время, необходимое для его построения, согласно техническому заданию требуется обеспечить добавление и удаление объектов со сцены (добавление и удаление шахматных фигур), что приводит к увеличению количества объектов сцены, и как, следствие, росту временных затрат. Таким образом, недостаток алгоритма уступает оптимизации обратного алгоритма трассировки лучей с точки зрения поиска пересечения с поверхностями, что позволяет его в дальнейшем использовать при разработке программного обеспечения.

Поскольку параллельная версия алгоритма обратной трассировки лучей комбинируется с другими оптимизациями данного алгоритма, то данную версию алгоритма можно также использовать с KD-деревьями.

Вывод

В данном разделе были формализованы объекты синтезируемой сцены, определены их геометрические, оптические характеристики, проанализированы существующие алгоритмы, решающие задачу синтеза сложного изображения в контексте моделирования статической сцены расстановки шахматных фигур на шахматной доске.

Основным алгоритмом была выбрана многопоточная версия обратной трассировки лучей с использованием KD-деревьев для вычисления пересечений с поверхностями сцены.

2 Конструкторская часть

В данном разделе будет формально описана декомпозиция разрабатываемого ПО с формальным описанием всех используемых алгоритмов.

2.1 Декомпозиция разрабатываемого ПО

На рисунке 2.1 приведена BPMN-диаграмма процесса инициализации статической сцены расстановки шахматных фигур на шахматной доске на основании введённого пользователем количества фигур.

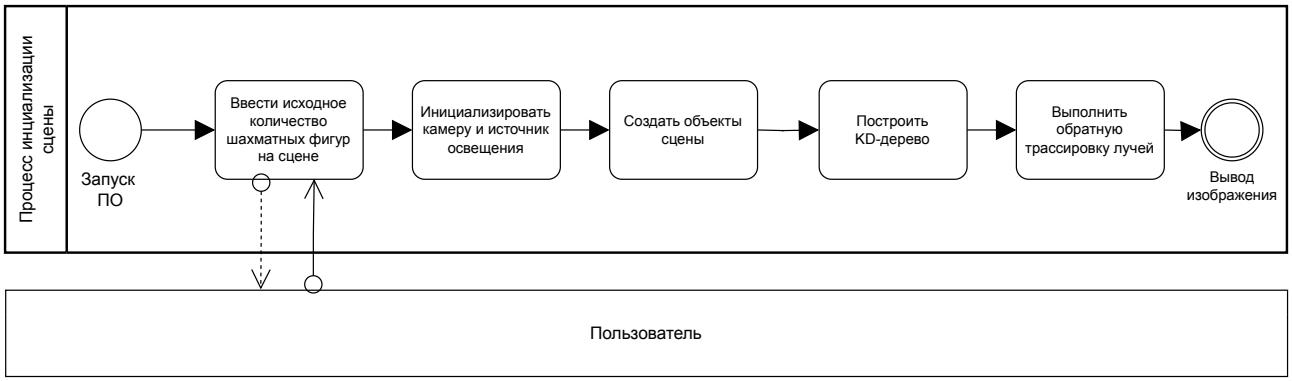

Рисунок 2.1 – BPMN-диаграмма процесса инициализации статической сцены расстановки шахматных фигур на шахматной доске на основании введённого пользователем количества фигур

На рисунке 2.2 приведена BPMN-диаграмма процесса добавления шахматной фигуры.

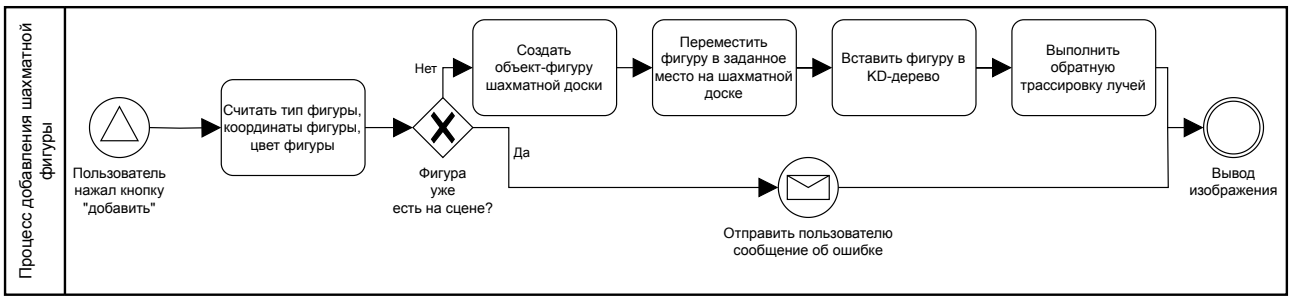

Рисунок 2.2 – BPMN-диаграмма процесса добавления шахматной фигуры

На рисунке 2.3 приведена BPMN-диаграмма процесса удаления шахматной фигуры со сцены.

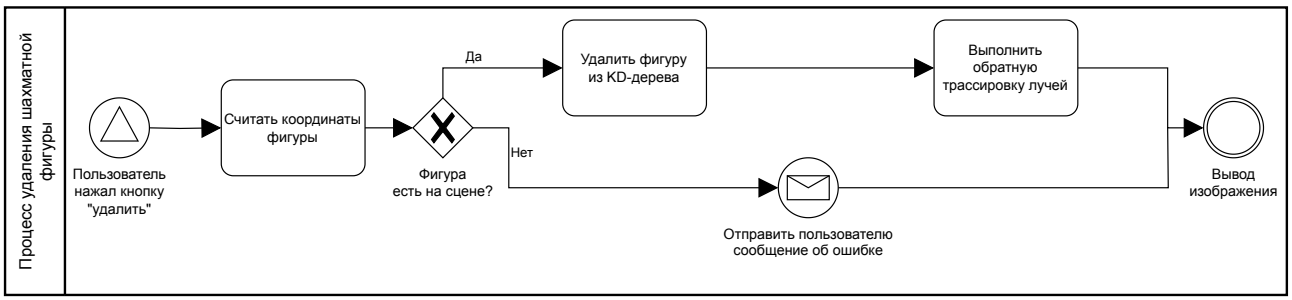

Рисунок 2.3 – BPMN-диаграмма процесса удаления шахматной фигуры со сцены

На рисунке 2.4 приведена BPMN-диаграмма процесса изменения спектральных характеристик объектов сцены.

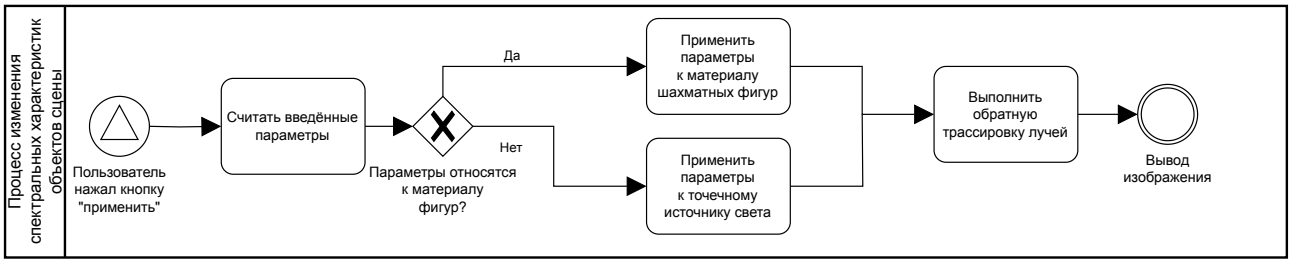

Рисунок 2.4 – BPMN-диаграмма процесса изменения спектральных характеристик объектов сцены

На рисунке 2.5 приведена BPMN-диаграмма изменения местоположения объектов сцены.

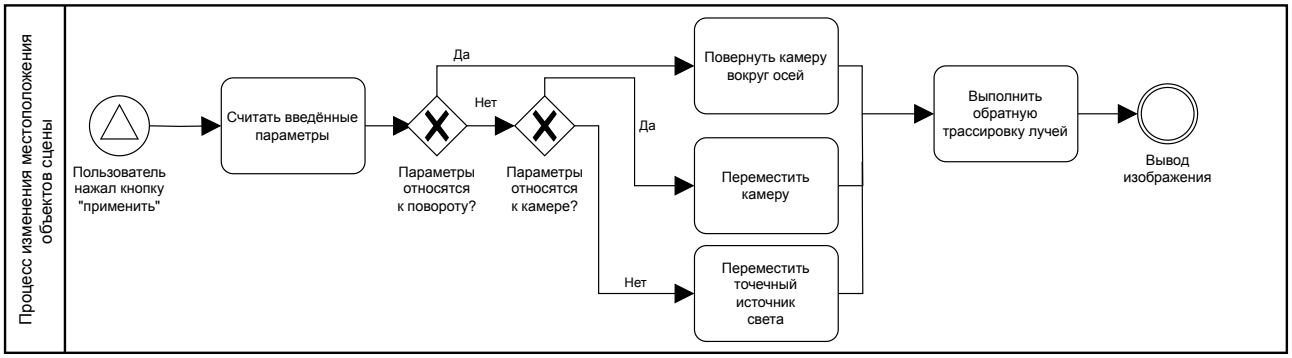

Рисунок 2.5 – BPMN-диаграмма процесса изменения местоположения объектов сцены

2.1.1 Структуры данных

В алгоритме обратной трассировки лучей необходимо, чтобы, в случае пересечения, была известна информация о точке пересечения (для построения отражённого луча), а также информация о спектральных характеристиках

материала. Поскольку при поиске пересечения в дальнейшем необходимо определять, присутствует ли в узле KD-дерева ещё одно KD-дерево, то необходимо иметь информацию об объекте соответственно.Также необходимо, что бы узлы дерева хранили информацию о левых и правых поддеревьях, ограничивающий параллелепипед, а также направление о оси, по которой произошла декомпозиция параллелепипеда и параметры самого ограничивающего параллелепипеда.

Таким образом, возникает необходимость наличия следующих структур (в понятии как совокупность данных) для дальнейшего упрощения описания схем алгоритмов.

— Ray - луч, исходящий из камеры, создаётся функцией Ray(origin, direction), хранит две данные: начало луча (origin) b его направление (direction);

— Material - структура данных, хранящая информацию о спектральных характеристиках поверхности (ambient – рассеянное излучение, diffuse – диффузная составляющая интенсивности поверхности, reflection – коэффициент отражения, p - коэффициент качества полировки);

— Hitinfo - структура хранит необходимую информацию о пересечении: t – значение параметра при пересечнии с объектом, material – структура Material, hit_point - значение точки пересечения в декартовой системе координат;

— Point - структура, хранящая информацию о координате в декартовой системе координат (данные x, y, z);

— BoundingBox - структура, хранящая информация об ограничивающем параллелепипеде, содержит две структуры Point, в одной из которых записаны минимальные значения координат объекта (min), а в другой максимальные (max);

— KDNode - узел KD-дерева, хранит необходимую информацию об узле: left, right - узлы, могут иметь значение null в случае, если узел лиственный, axis - ось, в направлении которой была выполнена декомпозиция

ограничивающего параллелепипеда, bx - структура BoundingBox ограничивающего параллелепипеда, objects - массив объектов лиственного узла, size - количество объектов в objects;

— Object - структура, которая хранит объект сцены или поверхность (data), центр объекта (center) а также его соответствующий ограничивающий параллелепипед (bx).

Доступ к данным структур в дальнейшем будет обозначен с использованием оператора «.», однако в действительности доступ к данным структуры зависит от выбора средств реализации.

2.2 Камера

Рассмотрим камеру как структуру данных.

Камера должна хранить несколько данных: местоположение камеры, направление взгляда, а также необходимо заранее определить, как ость относительное камеры смотрит вверх. К данным, описывающим камеру в случае обратной трассировки лучей, относятся также соотношение сторон используемого экрана, а также угол fov – областью видимости. Считается, что достоверное изображение можно получить, если $fov = 60^\circ$ по вертикали [4]. Пусть вверх смотрит ось z.

2.2.1 Вычисление луча из камеры в экран

Пусть необходимо получить луч из местоположения камеры в точку, соответствующую пиксель (i, j), а отношение ширины и высоты экрана равно k, середина экрана – display_center, местоположение камеры – center.

Расстояние от камеры до экрана – d. $tg(fov) = height/2/d$, из чего следует $height = 2 * tg(fov) \cdot d$. Пусть $d = 1$. Тогда $height = 2 \cdot tg(fov)$. Поскольку $k = width/height$, то $width = height \cdot k$.

Однако, местоположение экрана зависит от направления взгляда $(look\_up)$ камеры и оси, которая направлена вверх относительно неё up. Тем не менее, горизонтальная составляющая экрана должна быть направлена перпендикулярно оси, направленной верх, и направляю взгляда, что можно получить из векторного произведения $horizontal = [look\_up, up]$. Аналогично вертикальная составляющая $vertical = [horizontal, look\_up]$.

Если рассматривать $hotizontal$ и $vertical$ как единицы размерности экрана, то для получения реальных экранных высоты и ширины необходимо каждую из составляющих домножить на $width$ и $height$ соответственно.

Для вычисления пикселя согласно его номеру по ширине и высоте на пользовательском объекте-изображении необходимо ввести точку начала отсчёта. Пусть это будет нижний левый угол экрана, который можно получить следующим образом: $left\_button = loop\_up - (vertical + horizontal)/2$.

Таким образом, объект необходимого луча $Ray(center, left\_button + i * horizontal + j * vertical)$.

На рисунке 2.6 приведена используемая модель камеры.

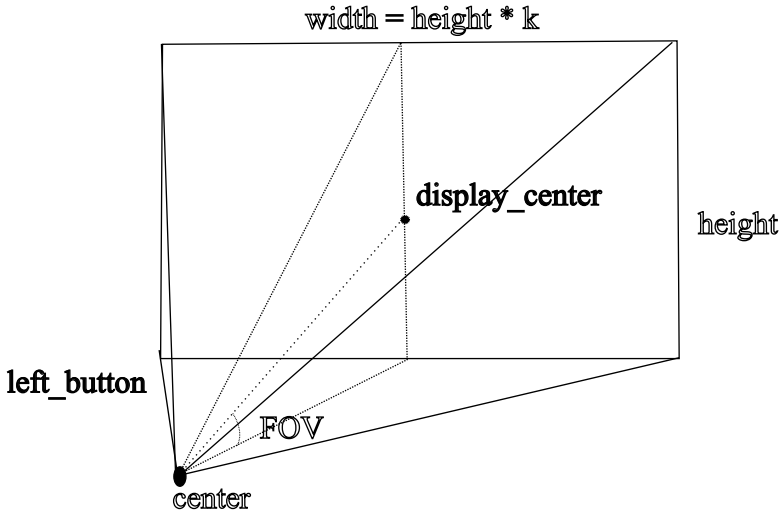

Рисунок 2.6 – Используемая модель камеры

2.3 KD-дерево

Поскольку каждый объект сцены состоит из множества треугольных поверхностей, то его можно представить как отдельное KD-дерево, а поскольку любое KD-дерево имеет глобальный ограничивающий параллелепипед, то совокупность объектов сцены можно представить как KD-дерево, состоящее из KD-поддеревьев.

2.3.1 Алгоритм построения KD-дерева

На рисунке 2.7 приведена схема алгоритма построения KD-дерева.

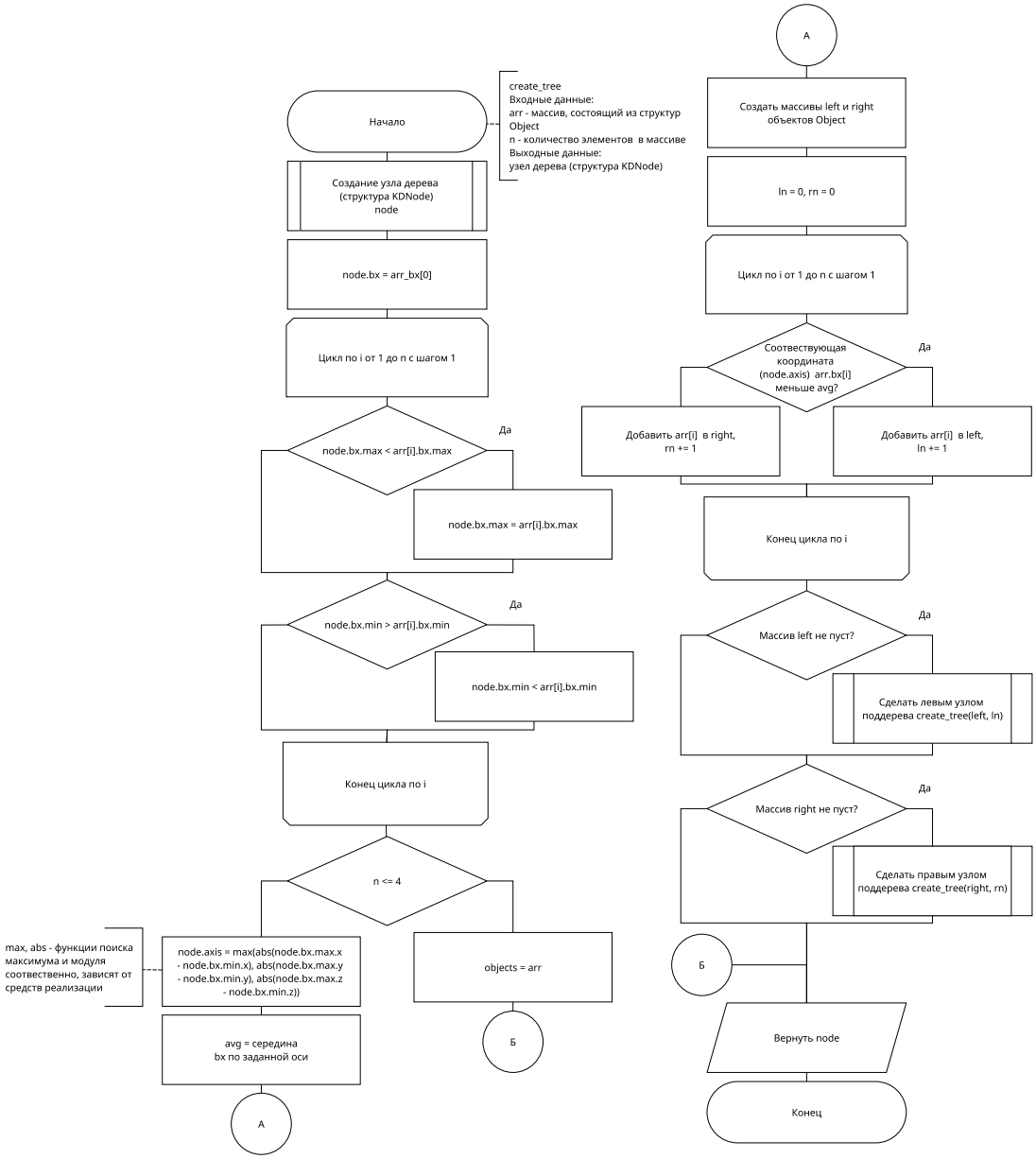

Рисунок 2.7 – Схема алгоритма построения KD-дерева

2.3.2 Алгоритм поиска пересечения в KD-дереве

На рисунке 2.8 приведена схема алгоритма поиска пересечения в KD-дерева.

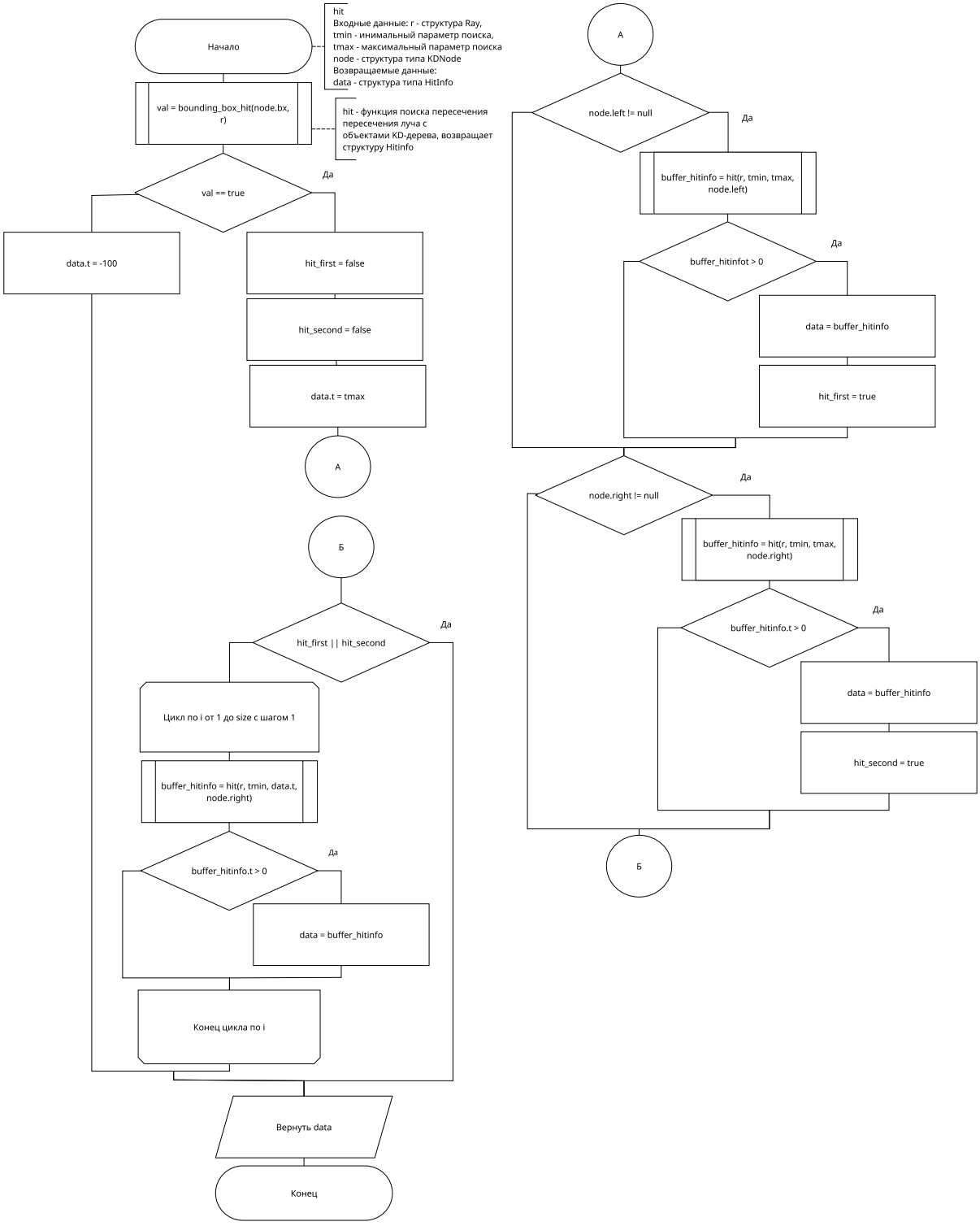

Рисунок 2.8 – Схема алгоритма поиска пересечения в KD-дерева

2.3.3 Алгоритм добавления объекта в KD-дерево

На рисунке 2.9 приведена схема алгоритма добавления объекта в KD-дерево.

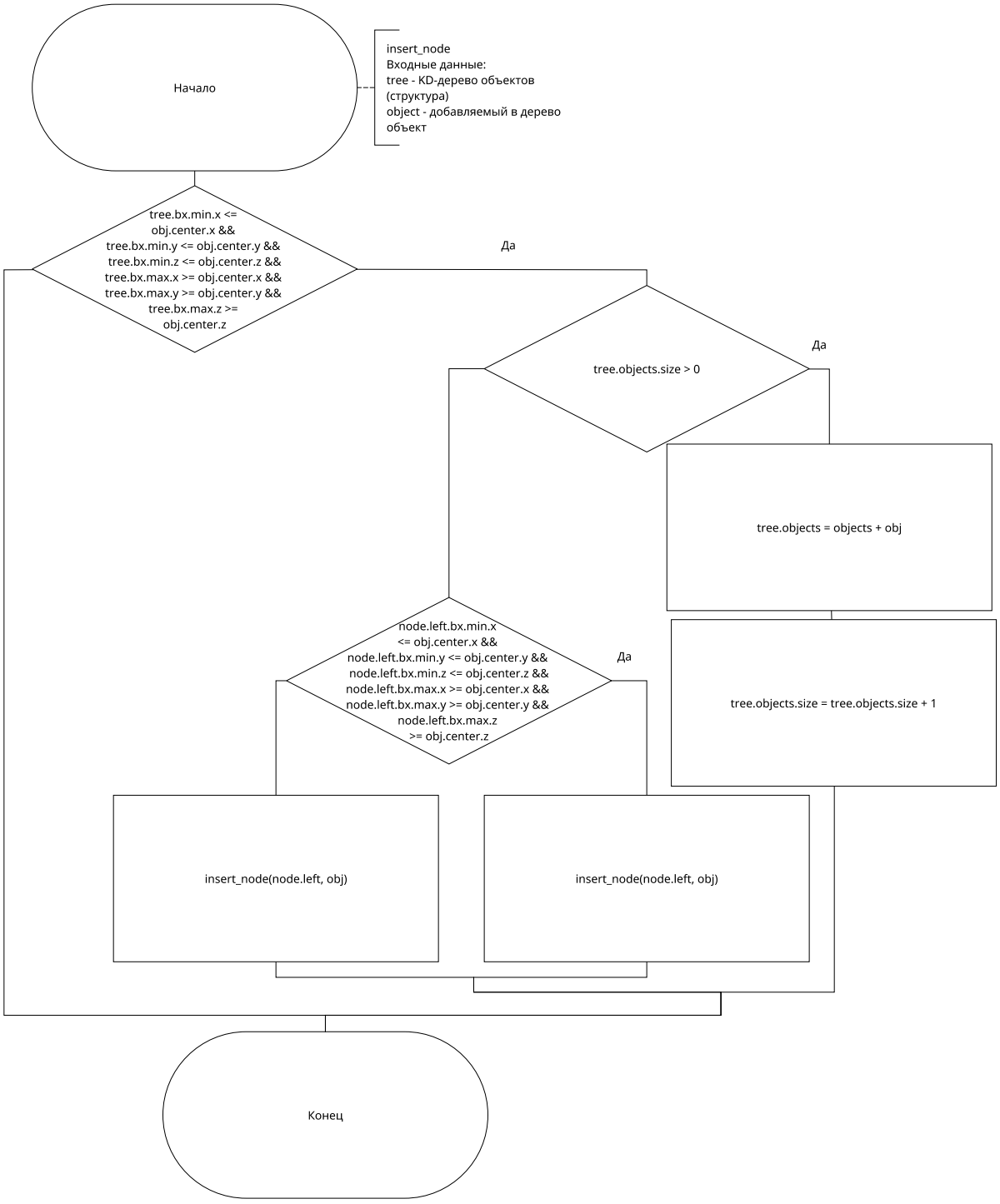

Рисунок 2.9 – Схема алгоритма добавления объекта в KD-дерево

2.3.4 Алгоритм удаления объекта из KD-дерева

Удаление из KD-дерева можно производить с учётом того, что будет удаляться шахматная фигура с доски согласно заданным пользователем координатам, иначе говоря, параметры ограничивающего параллелепипеда заранее известны.

Пусть ось z направлена вверх, а шахматная доска располагается парал-

лельно плоскости z = 0 и ниже плоскости относительно направления оси z плоскости z = 0, в то время как нижняя грань ограничивающих параллелепипедов расположена в плоскости z = 0. В таком случае, поскольку на доске не может быть в одной клетке более одной фигуры, то достаточно найти фигуру, нижняя грань которой соотносится с ведёнными параметрами клети на шахматной доске.

На рисунке 2.10 приведена схема алгоритма удаления объекта из KD-дерева.

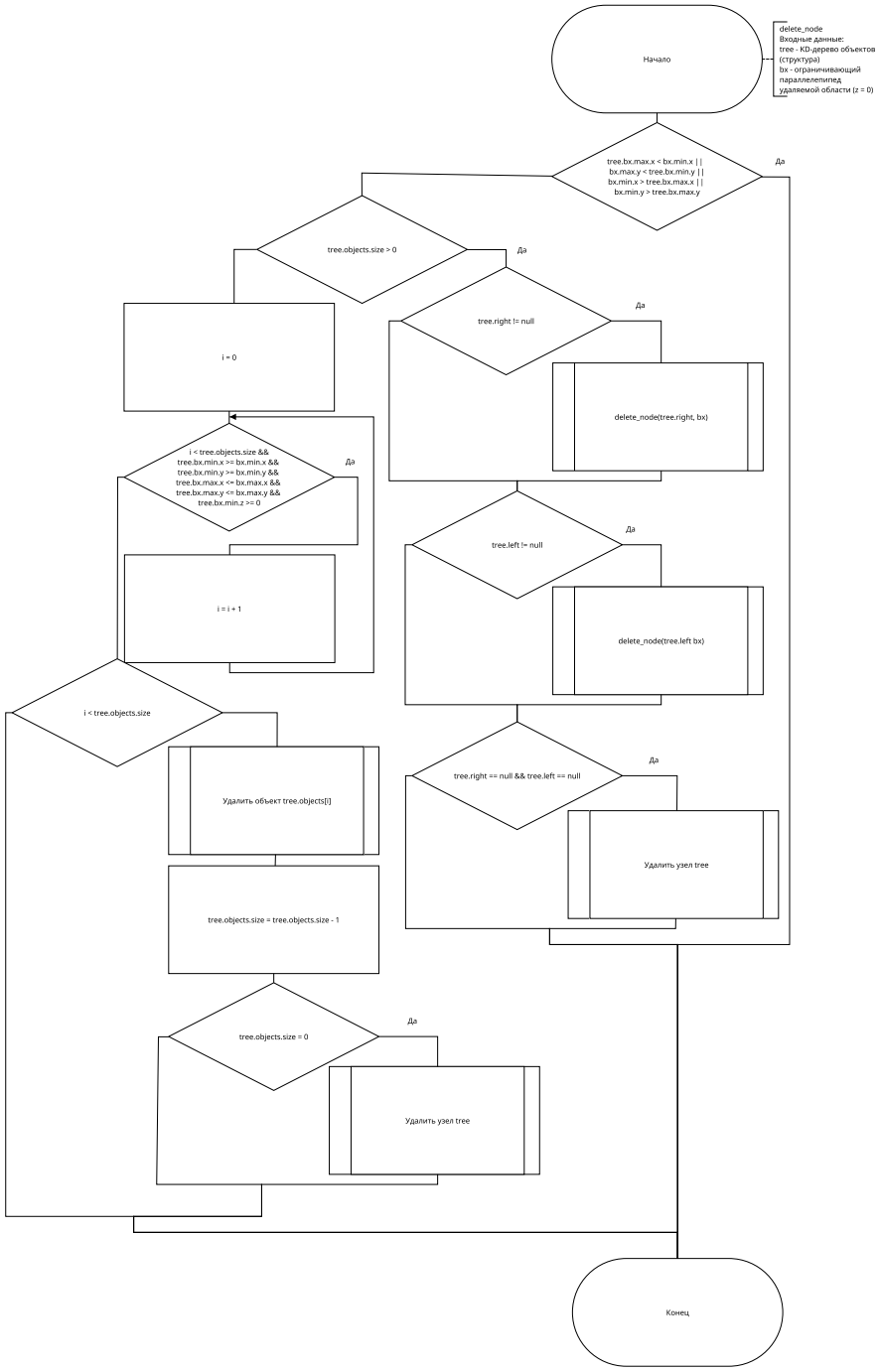

Рисунок 2.10 – Схема алгоритма добавления объекта в KD-дерево

2.3.5 Алгоритм поиска пересечения с плоскостью, ограниченной треугольником

2.4 Обратная трассировка лучей

На рисунке 2.11 приведена схемы алгоритмов отрисовки одного луча и перебора координат объекта-изображения, выводимого пользователю.

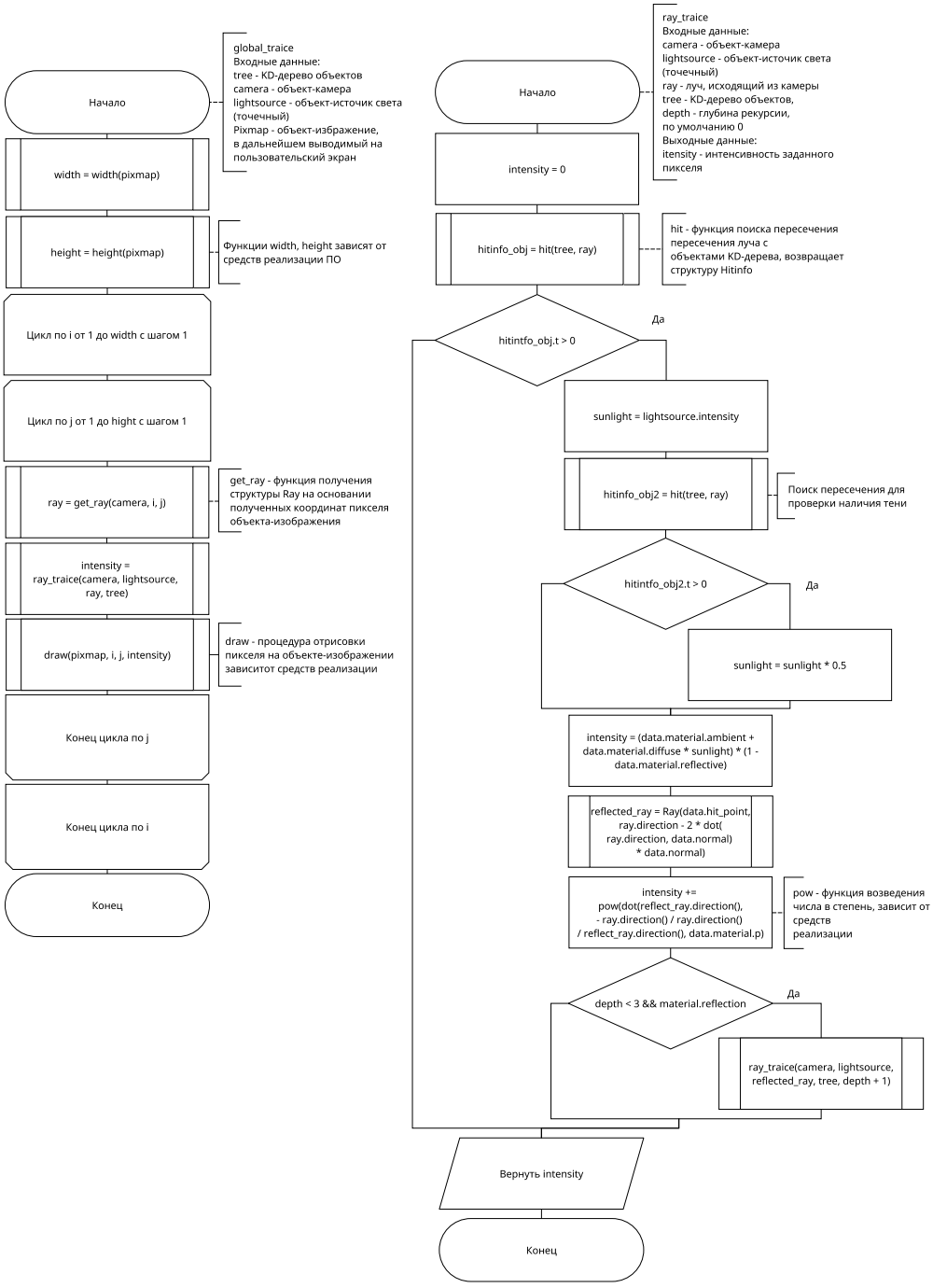

Рисунок 2.11 – Схемы алгоритмов отрисовки одного луча и перебора координат объекта-изображения, выводимого пользователю

Вывод

В данном разделе была будет формально описана декомпозиция разрабатываемого ПО с формальным описанием всех используемых алгоритмов.

3 Технологическая часть

В данном разделе будет описан и обоснован выбор средств реализации программного обеспечения и представлены детали реализации.

3.1 Средства реализации программного обеспечения

В качестве языка программирования был выбран язык c++ в силу следующих причин:

— имеется опыт разработки на данном языке;

— средствами языка можно реализовать все алгоритмы, выбранные в результате проектирования.

Так же использовались следующие библиотеки:

— кроссплатформленная библиотека Qt для разработки графического интерфейса, поскольку имеется опыт разработки, с использованием этой библиотеки;

— библиотека Intel® Threading Building Blocks (Intel® TBB), предназначенная для параллелизма на уровне инструкций во время выполнения для эффективного использования ресурсов процессора [9]. Использовалась как средство реализации параллельной версии алгоритма обратной трассировки лучей. Библиотека совместима в другими библиотеками потоков [9], из чего следует, что конфликта с потоками библиотеки Qt не будет.

— библиотека stl_reader, состоящая для одного заголовочного файла и предназначенная для чтения stl файлов, а также преобразования содержимого в пользовательские контейнеры [10]. Необходима в связи с надобностью, исходя из аналитической части, чтения stl-файлов заранее подготовленных моделей шахматных фигур.

Для ускорения сборки программного обеспечивания использовалась утилита cmake [11].

3.2 Структура программного обеспечивания

В разработанном программном обеспечении реализованы следующие классы.

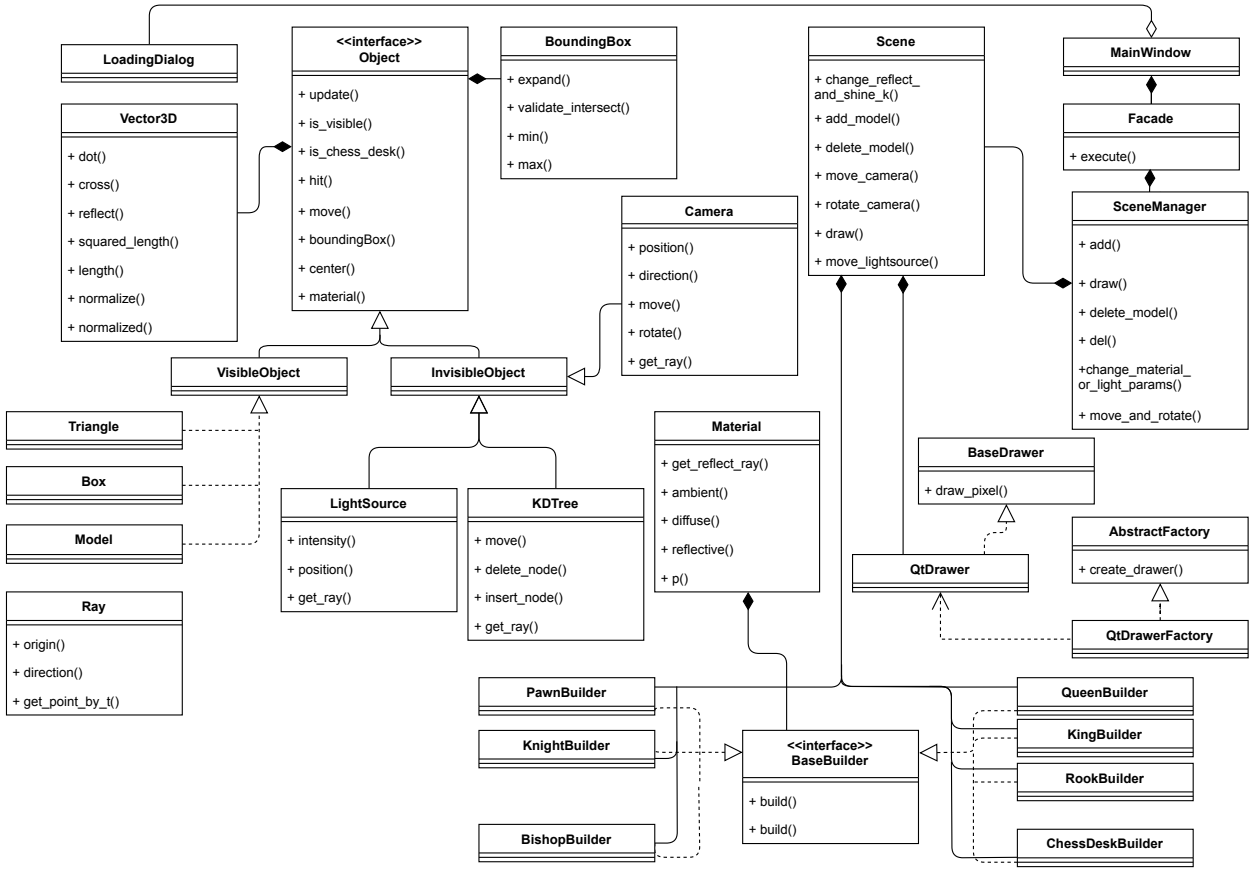

Рисунок 3.1 – Диаграмма классов (ч. 1)

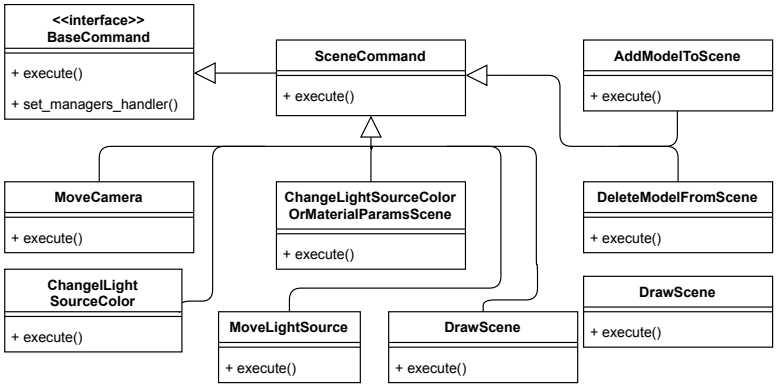

Рисунок 3.2 – Диаграмма классов (ч. 2)

Ниже приведено писание следующих основных классов, используемых в программном обеспечении.

— классы, связанные с объектами сцены;

  – Object – базовый класс объекта сцены, определяет интерфейс объекта;
  – Box – описывает параллелепипед, используется для создания «клетки» шахматной доски;
  – Triangle – описывает плоскость, ограниченную треугольником, единица декомпозиции модели;
  – Model – базовый класс модели сцены;
  – Camera – наблюдатель (камера);
  – LightSource – точечный источник света;

— классы, связанные с использованием модификации обратной трассировки;

  – KDTree – класс, используемый для построения KD-дерева;
  – BoundingBox – параллелепипед, использующийся в модификации для определения наличия пересечения луча, исходящего из камеры, с объектом;
  – Material – используется как контейнер спектральных характеристик модели;
  – Ray – хранит параметрическое описание прямой;
  – Vector3D – описывает вектор в декартовых координатах;

— классы, связанные с взаимодействием интерфейса и объектов сцены;

  – Scene – класс, описывающий объекты сцены, в одном из методов реализован основной алгоритм обратной трассировки лучей;
  – SceneManager – класс, используемый для инкапсулирования сцены;
  – Facade, *Command, SceneCommand - реализация паттернов команда и фасад для обеспечения интерфейса сцены в совокупности;
  – QtDrawer - класс, используемый для заполнения объекта QPixmap – экземпляр класса библиотеки Qt для работы с изображениями [12],

3.3 Интерфейс пользователя

Интерфейс пользователя включает в себя следующие объекты:

— область изображения сцены;

— список выбора шахматных фигур;

— списки выбора цвета и соответствующих координат при добавлении фигуры;

— списки выбора координат фигуры при её удалении;

— вкладки «Материал поверхности» и «Точечный источник света» для изменения спектральных характеристик соответствующих объектов;

— поля «Показатель качества полировки» (целое число, варьируется от 0 до 1000) и «Коэффициент диффузного отражения» (десятичная дробь, два знака после запятой, варьируется от 0.3 до 1);

— поле «Выбор интенсивности» для изменения интенсивности точечного источника освещения;

— Кнопка «Камера-Источник света» для выбора объекта, местоположение которого изменяется;

— Поля «dx», «dy», «dz» для выполнения переноса выбранного объекта (целые числа, варьируются -5000 до 5000) и поля «Ox», «Oy», «Oz» для выполнения поворота вокруг соответствующих осей (в случае выбора «Источника света» вкладка «Поворот» недоступная для заполнения).

На рисунке 3.3 представлен интерфейс пользователя.

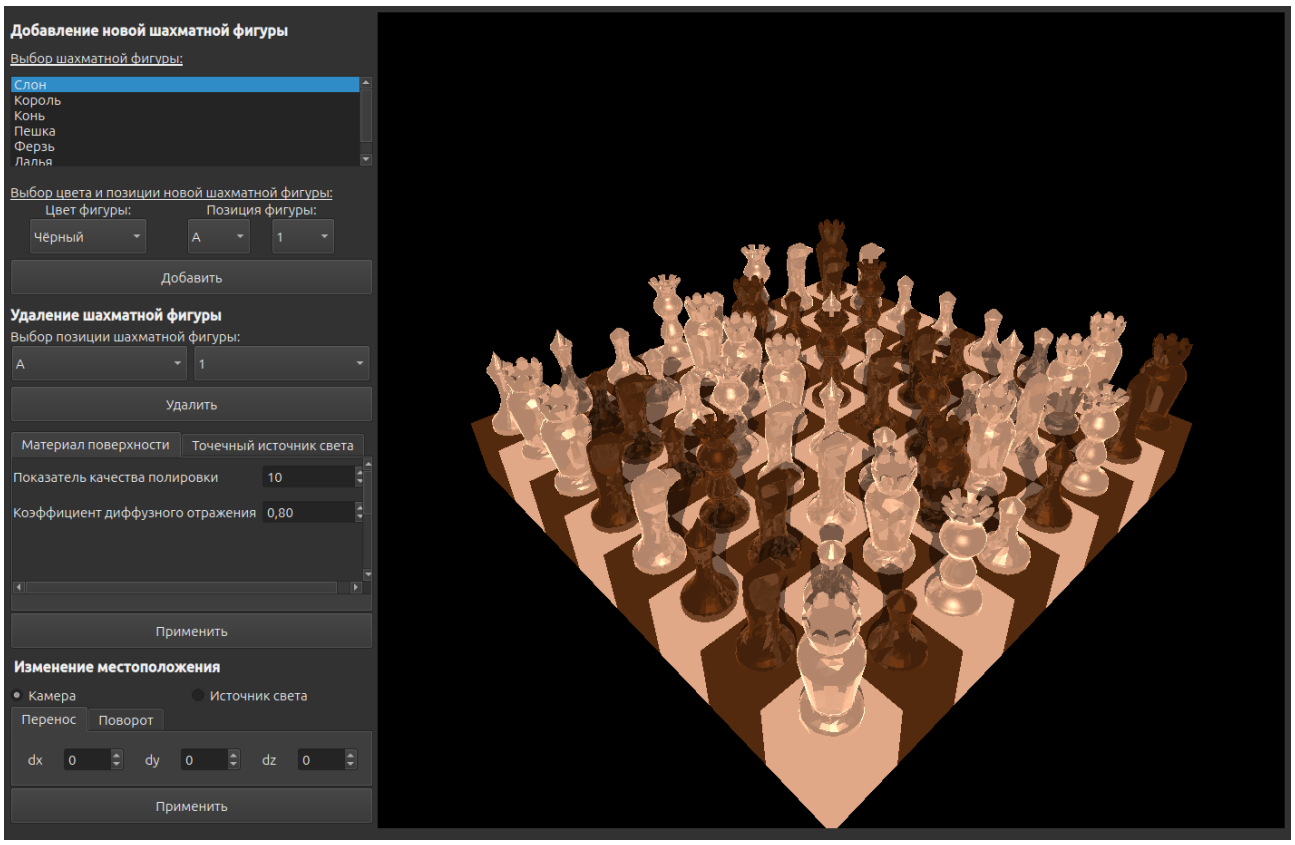

Рисунок 3.3 – Интерфейс пользователя

Вывод

В данном разделе была описана и обоснован выбор средств реализации программного обеспечения и представлены детали реализации.

4 Исследовательская часть

В данном разделе будет описана цель исследования по времени, технические характеристики устройства, на котором будет проводиться исследование по времени, а также приведено само исследование.

4.1 Цель исследования

В реализации алгоритма обратной трассировки лучей присутствует рекурсивная часть. В ходе выполнения программы, данная часть выполняется в том случае, если коэффициент отражения поверхности больше 0. В противном случае будет выполняться только линейная часть программы.

Цель исследования заключается в сравнении времени выполнения реализации алгоритма обратной трассировки лучей в случае, если коэффициент отражения равен 0, и в случае, если коэффициент отражения не равен 0, от количества шахматных фигур на сцене.

4.2 Технические характеристики устройства

Ниже приведены технические характеристики устройства, на котором проводилось тестирование:

— Операционная система: Ubuntu Mantic Minotaur (development branch);

— Оперативная память: 16 ГБ;

— Процессор: Intel® Core™ i7-10510U × 8.

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, окружением, а также непосредственно системой тестирования.

Использовался компилятор g++. Оптимизация отключалась с помощью использования опции компилятора $-O0$.

4.3 Время выполнения реализации алгоритма

Замеры времени выполнения для каждого количества шахматных фигур 10 раз. Максимальное количество фигур – 24, изменение количества

выполнялось с шагом 2.

В Таблице 4.1 приведены замеры времени работы (в тиках).

Таблица 4.1 – Замеры времени выполнения в тиках (среднее значение)

Количество фигур на сцене	Коэффициент отражения равен 0	Коэффициент отражения равен 1
0	36 928 591.6	37 165 577.5
2	63 915 518.8	74 191 396.7
4	81 624 786.9	97 750 875.7
6	87 910 836.0	114 321 536.9
8	103 859 896.7	129 753 559.5
10	120 782 833.2	171 895 533.1
12	129 186 274.5	173 975 459.4
14	131 764 314.4	183 770 128.0
16	138 377 645.3	193 480 551.0
18	157 251 043.0	229 007 486.8
20	182 764 486.7	273 964 928.9
22	193 834 301.5	299 999 849.2
24	211 815 320.6	327 839 202.1

На рисунке 4.1 представлены дискретные данные, полученные в результате замера процессорного времени (в тиках).

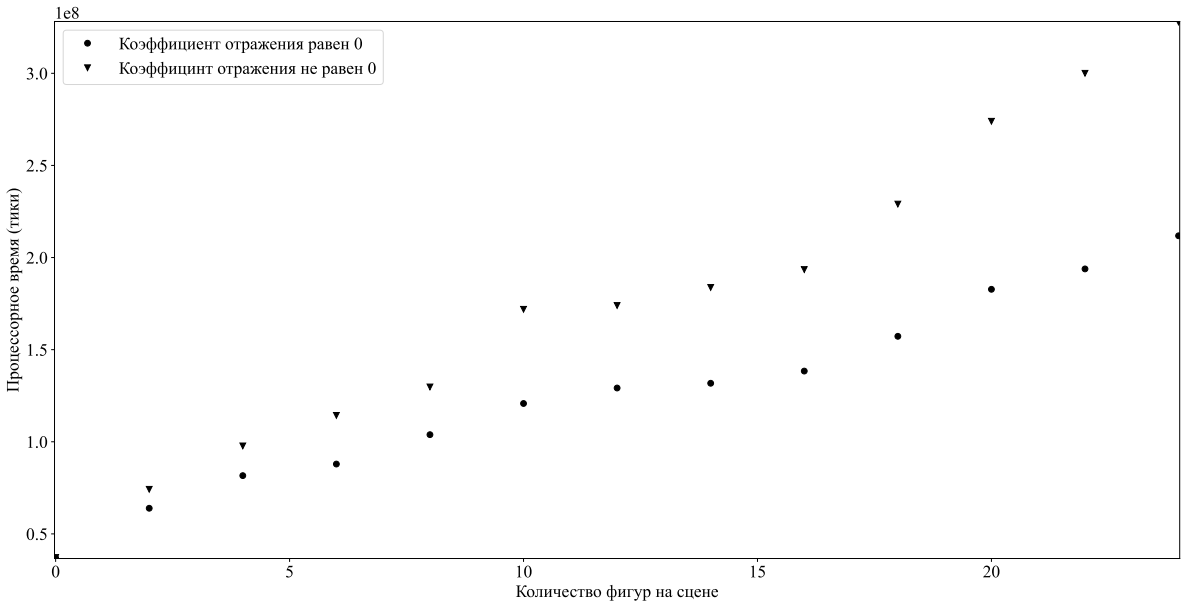

Рисунок 4.1 – Данные, полученные в результате замера процессорного времени (в тиках)

Зависимость времени работы (в тиках) от количества фигур на шахмат-

ной сцене можно аппроксимировать, используя метод наименьших квадратов. Минимальная сумма неявок δ (остаточная сумма квадратов) достигается при степени полинома $n = 6$ в случае данных при коэффициенте отражения равному 0, и степени полинома $n = 8$ в случае данных при коэффициенте отражения не равному 0. На рисунке 4.2 приведена зависимость времени (в тиках) от количества фигур на шахматной сцене при коэффициенте отражения, равному 0, и коэффициенте отражения, не равному 0. В первом случае $\delta = 1.48e + 14$, во втором случае – $\delta = 1.33e + 14$.

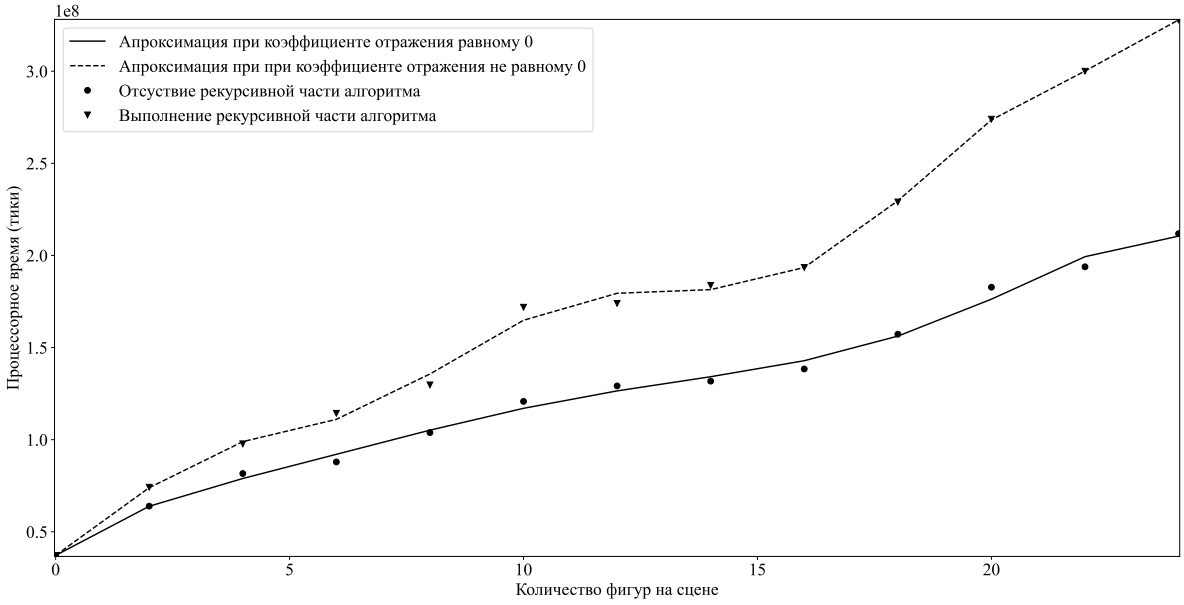

Рисунок 4.2 – Зависимость времени (в тиках) от количества фигур на шахматной сцене при коэффициенте отражения, равному 0, и коэффициенте отражения, не равному 0

Вывод

В данном разделе была описана цель исследования по времени, технические характеристики устройства, на котором будет проводиться исследование по времени, а также приведены результаты исследования.

Было проведено исследование зависимости времени выполнения (в тиках) реализации алгоритма обратной трассировки лучей в случае, если коэффициент отражения равен 0 (случай 1), и в случае, если коэффициент отражения отличен от 0 (случай 2).

В результате аппроксимации с использованием метода наименьших квадратов выяснилось, что зависимость времени (в тиках) от количества шахмат-

ных фигур в случае 1 можно выразить с использованием полинома степени $n = 6$, в случае 2 – полинома степени $n = 8$. В случае 1 минимальная сумма неявок $\delta = 1.48e + 14$, во втором случае – $\delta = 1.33e + 14$.

Поскольку порядок роста второй функции выше, чем первой, то можно сказать, что требуемое процессорное время для выполнения в случае 2 больше, чем в случае 1, независимо от прочих значений входных параметров.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы поставленная цель была достигнута: разработано программное обеспечение с пользовательским интерфейсом, реализовывающего моделирование статической сцены расстановки шахматных фигур на шахматной доске.

В ходе выполнения курсовой работы были решены все задачи.

— проанализирована предметная область, рассмотрены известные подходы и алгоритмы решения задачи синтеза изображения в контексте моделирования статической сцены;

— спроектировано программное обеспечение;

— выбраны средства реализации и разработано программное обеспечение;

— исследованы характеристики разработанного программного обеспечения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Конспект лекций по дисциплине «Компьютерная графика», Куров А. В. — 2023.
2. *Олейников М. И.* ЯЗЫК ПАРАМЕТРИЧЕСКОГО ОПИСАНИЯ ОПТИКО-ГЕОМЕТРИЧЕСКИХ СЦЕН В ЗАДАЧАХ ТЕХНИЧЕСКОГО ЗРЕНИЯ. — Санкт-Петербург : Известия ТулГУ. Технические науки. №12, 2022. — Режим доступа: https://cyberleninka.ru/article/n/yazyk-parametricheskogo-opisaniya-optiko-geometricheskih-stsen-v-zadachah-tehnicheskogo-zreniya (дата обращения: 7.10.2023).
3. *Дёмин А. Ю.* Основы компьютерной графики: учебное пособие. — Томск : Издательство Томского политехнического университета, 2011. — ISBN 973-5-0328-0145-4.
4. *Божко А. Н., Жук Д. М.* Компьютерная графика : учебное пособие. — Москва : МГТУ им. Баумана, 2007. — ISBN 978-5-7038-3015-4.
5. *Ульянов А. Ю.* МЕТОД ТРАССИРОВКИ ЛУЧЕЙ КАК ОСНОВНАЯ ТЕХНОЛОГИЯ ФОТОРЕАЛИСТИЧНОГО РЕНДЕРИНГА. — Екатеринбург : ФГАОУ ВПО «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина», 2015. — Режим доступа: https://s.fundamental-research.ru/pdf/2015/11-6/39706.pdf (дата обращения: 19.11.2023).
6. *Зори С. А.* Использование вычислительных возможностей графических систем для организации визуализации трехмерных сцен с использованием технологий объемного отображения. — Санкт-Петербург : Технические науки. №4 (153), 2014. — Режим доступа: https://cyberleninka.ru/article/n/ispolzovanie-vychislitelnyh-vozmozhnostey-graficheskih-sistem-dlya-organizatsii-vizualizatsii-trehmernyh-stsen-s-ispolzovaniem (дата обращения: 7.11.2023).
7. *К.Боресков А.* Программирование компьютерной графиких. — Москва : ДМК, 2019. — ISBN 978-5-97060-779-4.

8. *Роджерс Д.* Алгоритмические основы машинной графики. — Москва : Мир, 1989. — ISBN 5-03-000476-9.

9. Intel® Threading Building Blocks Tutorial [Электронный ресурс]. — Режим доступа: `https://www.inf.ed.ac.uk/teaching/courses/ppls/TBBtutorial.pdf` (дата обращения: 1.12.2023).

10. stl_reader [Электронный ресурс]. — Режим доступа: `https://sreiter.github.io/stl_reader/stl__reader_8h.html#details` (дата обращения: 3.11.2023).

11. CMake Reference Documentation [Электронный ресурс]. — Режим доступа: `https://cmake.org/cmake/help/latest/index.html` (дата обращения: 14.12.2023).

12. Qt Documentation [Электронный ресурс]. — Режим доступа: `https://doc.qt.io/` (дата обращения: 14.12.2023).