

2024 г.

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>6</b>
<b>1 Аналитическая часть</b>	<b>7</b>
1.1 Оптические эффекты, учитываемые при построении реалистичного изображения	7
1.2 Алгоритмы удаления невидимых линий и поверхностей	8
1.2.1 Алгоритм Робертса	8
1.2.2 Алгоритм Варнака	8
1.2.3 Алгоритм, использующий z-буфер	8
1.2.4 Алгоритм обратной трассировки лучей	8
1.3 Методы закраски	9
1.3.1 Простая закраска	9
1.3.2 Закраска методом Гуро	9
1.3.3 Закраска методом Фонга	10
1.4 Построение теней	10
1.5 Алгоритм симуляции эффекта глубины поля	11
<b>2 Конструкторская часть</b>	<b>12</b>
2.1 Требования к программному обеспечению	12
2.2 Формализация объектов синтезируемой сцены	12
2.3 Декомпозиция задачи	13
2.4 Алгоритм трассировки лучей	14
2.5 Алгоритмы нахождения точки пересечения луча с объектом сцены	16
2.5.1 Алгоритм нахождения точки пересечения луча со сферой	16
2.5.2 Алгоритм нахождения точки пересечения луча и куба	17
2.5.3 Алгоритм нахождения точки пересечения луча и шахматной фигуры	19
<b>3 Технологическая часть</b>	<b>22</b>
3.1 Средства реализации	22
3.2 Структура программы	22
3.3 Реализация алгоритмов	24
3.4 Функциональное тестирование	29
3.5 Интерфейс программного обеспечения	31
<b>4 Исследовательская часть</b>	<b>34</b>
4.1 Технические характеристики	34

4.2	Описание исследования . . . . .	34
4.3	Результаты исследования . . . . .	34
<b>ЗАКЛЮЧЕНИЕ . . . . .</b>		<b>37</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ . . . . .</b>		<b>38</b>
<b>ПРИЛОЖЕНИЕ А . . . . .</b>		<b>40</b>
<b>ПРИЛОЖЕНИЕ Б . . . . .</b>		<b>41</b>

# ВВЕДЕНИЕ

Задача построения реалистичных трехмерных сцен, учитывающих оптические свойства поверхностей и источников света, является центральной в компьютерной графике. В настоящей работе рассматриваются методы решения данной задачи.

Цель работы – разработка программного обеспечения (ПО) для визуализации сцен трехмерных объектов, позволяющего изменять параметры объектов.

Для достижения поставленной цели необходимо решить следующие задачи:

- 1) проанализировать предметную область, рассмотреть известные подходы и алгоритмы;
- 2) спроектировать ПО для построения реалистичных сцен;
- 3) реализовать выбранные алгоритмы для построения трехмерной сцены;
- 4) исследовать характеристики разработанного ПО.

# 1 Аналитическая часть

## 1.1 Оптические эффекты, учитываемые при построении реалистичного изображения

При построении реалистичных изображений необходимо учитывать такие эффекты оптики, как:

- отражающие свойства поверхностей: диффузное и зеркальное отражения;
- преломление части света при взаимодействии луча с прозрачной поверхностью;
- рассеяние света.

Поскольку в данной работе не используются прозрачные объекты, будем рассматривать только отражение и рассеяние света.

Диффузная составляющая явления отражения света от поверхностей описывается следующим уравнением:

$$I_d = I_n \cos \theta \cdot k_d, \quad (1.1)$$

где  $I_n$  – интенсивность источника освещения,  $\theta$  – угол между нормалью к поверхности в точке отражения и вектором, направленным из точки отражения к источнику света,  $k_d$  – коэффициент диффузного отражения.

Зеркальная составляющая явления отражения света описывается уравнением:

$$I_z = I_n \cos^n \alpha \cdot k_z, \quad (1.2)$$

где  $I_n$  – интенсивность источника освещения,  $\alpha$  – угол падения луча,  $k_z$  – коэффициент зеркального отражения, степень  $n$  косинуса характеризует концентрированность света в направлении отражения.

Рассеяние света в компьютерной графике обычно описывают следующей формулой:

$$I' = I_p k_p, \quad (1.3)$$

где  $I_p$  – интенсивность рассеянного света,  $k_p$  – коэффициент диффузного отражения рассеянного света.

Также необходимо учесть затухание света с расстоянием. Интенсивность света падает обратно пропорционально квадрату расстояния до источника, однако, как показывает практика, большей реалистичности можно добиться при линейном затухании [1]. В этом случае получаем следующий результат:

$$I = I_p k_p + \frac{I_n \cos \theta \cdot k_d + I_n \cos^n \alpha \cdot k_z}{d + K}, \quad (1.4)$$

где  $d$  – расстояние до источника освещения,  $K$  – произвольная константа.

## **1.2 Алгоритмы удаления невидимых линий и поверхностей**

Все алгоритмы удаления невидимых линий работают либо в пространстве объектов, либо в пространстве изображений. Рассмотрим наиболее известные из них.

### **1.2.1 Алгоритм Робертса**

Этот алгоритм работает в объектном пространстве и использует проекции объектов на картинную плоскость для анализа видимости. Алгоритм прежде всего удаляет из каждого тела те ребра или грани, которые экранируются самим телом. Затем каждое из видимых ребер каждого тела сравнивается с каждым из оставшихся тел для определения того, какая его часть или части, если таковые есть, экранируются этими телами. Поэтому вычислительная трудоемкость алгоритма Робертса растет теоретически как квадрат числа объектов [1, 2].

### **1.2.2 Алгоритм Варнака**

Алгоритм Варнака, работающий в пространстве изображений, реализует рекурсивное разбиение исходного окна на подокна для определения видимых объектов сцены. Если текущее окно содержит более одного объекта или пересекается с объектом, оно делится на четыре равные части. Процесс повторяется до тех пор, пока размер подокна не станет равен одному пикселю или оно не окажется полностью пустым. По завершении разбиения цвет пикселя определяется ближайшим видимым объектом. Основным недостатком алгоритма — высокая вычислительная сложность из-за рекурсивного характера, особенно при работе с деталями в высоком разрешении [3].

### **1.2.3 Алгоритм, использующий z-буфер**

Алгоритм основывается на хранении глубины (z-значения) каждого пикселя экрана и определении ближайшей к экрану точки при разложении многоугольников в растр. Главное преимущество алгоритма – его простота. Кроме того, этот алгоритм решает задачу об удалении невидимых поверхностей и делает тривиальной визуализацию пересечений сложных поверхностей – сцены могут быть любой сложности. Основным недостатком алгоритма – большой объем требуемой памяти [1].

### **1.2.4 Алгоритм обратной трассировки лучей**

Алгоритм трассировки лучей моделирует процесс формирования изображения путём симуляции распространения световых лучей в сцене. При прямой трассировке отслеживаются лучи, исходящие от источника света, что неэффективно, поскольку далеко не все лучи попадут в камеру, а значит, повлияют на изображение. По этой причине используется алгоритм обратной трассировки лучей. При обратной трассировке выполняется построение изображения, следя за

траекторией лучей, исходящих из камеры. Лучи проверяют пересечения с объектами сцены, и, если такое пересечение обнаружено, рассчитывается цветовая характеристика точки с учётом материалов объекта, освещения, отражений и преломлений. Для моделирования этих эффектов алгоритм порождает вторичные лучи, что позволяет создавать реалистичные изображения. Основным недостатком метода является его высокая вычислительная сложность [2, 3].

## **1.3 Методы закраски**

### **1.3.1 Простая закраска**

Если предположить, что источник света находится на бесконечности, то лучи света, падающие на поверхность, параллельны между собой. Если к этому добавить условие, что наблюдатель находится в бесконечно удаленной точке, то эффектом ослабления света с увеличением расстояния от источника также можно пренебречь. При таких вводных плоская грань во всех ее точках имеет одинаковую интенсивность освещения, поэтому она закрашивается одним цветом.

При закрашивании этим методом на стыке соседних граней неизбежно будут проявляться ребра, поскольку соседние грани с различными направлениями нормалей имеют разный цвет [4].

### **1.3.2 Закраска методом Гуро**

Один из способов устранения дискретности интенсивностей закрашивания был предложен Гуро. Его метод заключается в том, что используются не нормали к плоским граням, а нормали к аппроксимируемой поверхности, построенные в вершинах многогранника. После этого вычисляются интенсивности в вершинах, а затем во всех внутренних точках многоугольника выполняется билинейная интерполяция интенсивности. К недостаткам метода Гуро следует отнести то, что он хорошо работает только с диффузной моделью отражения. Форма бликов на поверхности и их расположение не могут быть адекватно воспроизведены при интерполяции на многоугольниках. Кроме того, есть проблема построения нормалей к поверхности. В алгоритме Гуро нормаль в вершине многогранника вычисляется путем усреднения нормалей к граням, примыкающим к этой вершине. Такое построение сильно зависит от характера разбиения.

Еще один недостаток закраски изображен на рисунке 1.1. Если нормали к вершинам В, С, D вычислить усреднением нормалей к многоугольникам, то они будут одинаково ориентированы, то есть интенсивность в этих точках будет равной. При линейной интерполяции от В до D значение интенсивности получится постоянным, и поверхность на данном участке будет выглядеть плоской [4].

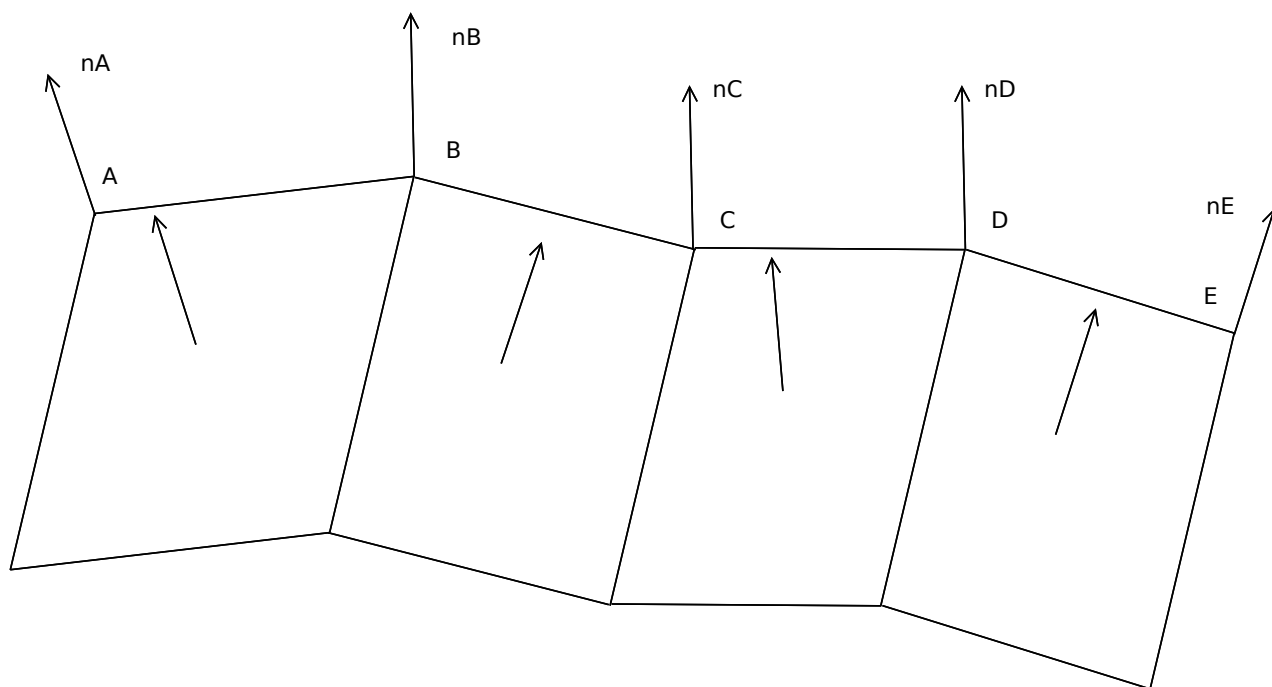


Рисунок 1.1 — Пример некорректной работы закрашки Гуро

### 1.3.3 Закраска методом Фонга

Фонг предложил вместо интерполяции интенсивностей произвести интерполяцию вектора нормали к поверхности на сканирующей строке. Этот метод требует больших вычислительных затрат, поскольку формулы интерполяции применяются уже к трем компонентам вектора нормали, но зато дает лучшую аппроксимацию кривизны поверхности. Поэтому зеркальные свойства поверхности воспроизводятся гораздо лучше.

Нормали к поверхности в вершинах многогранника вычисляются так же, как и в методе Гуро, после чего выполняется билинейная интерполяция в сочетании с построчным сканированием. После построения вектора нормали в очередной точке вычисляется интенсивность [4].

## 1.4 Построение теней

Тени делятся на два вида: собственные и проекционные. Определение собственных теней не представляет сложностей: точка наблюдателя совмещается с положением источника света и решается задача определения видимых граней. Видимые грани окажутся освещенными, невидимые – в тени.

Если источник света находится на бесконечном расстоянии, то при падении света на объект сцены будет образовываться полоса тени, которую можно задать уравнением. Если же источник света находится не на бесконечном расстоянии, возникает потребность нахождения проекции грани в тени на поверхность другого тела [5].

Стоит отметить, что в случае алгоритма обратной трассировки лучей задача построения теней сводится к тому, чтобы при нахождении точки пересечения луча с объектом сцены про-



вести еще один луч от этой точки к источнику света. Если на пути луча встречается преграда в виде поверхности какого-либо объекта сцены, точка находится в тени.

## **1.5 Алгоритм симуляции эффекта глубины поля**

Алгоритм симуляции эффекта глубины поля, то есть алгоритм, позволяющий изображать объекты, расположенные ближе к фокальной плоскости более четкими, чем те, что расположены дальше от нее, является модификацией алгоритма трассировки лучей. Работа алгоритма заключается в том, чтобы применять случайное отклонение начала луча трассировки от положения камеры в пространстве в рамках круга заданного радиуса, называемого апертурой. Круг при этом лежит в плоскости, параллельной картинной плоскости.

В рамках данного алгоритма также вводится понятие фокальной точки, задающейся фиксированным расстоянием от камеры по вектору направления взгляда для каждого пикселя кадра. Таким образом, модифицированный трассируемый луч, начало которого задается с помощью случайной точки круга заданного радиуса, направляется к фокальной точке [7].

## **Вывод**

В данном разделе были рассмотрены основные методы построения реалистичных изображений. Для удаления невидимых ребер, учета отражения света и теней, а также симуляции эффекта глубины поля в данной работе был выбран алгоритм обратной трассировки лучей в силу своей универсальности и реалистичности результата.

## **2 Конструкторская часть**

В данном разделе формализуются требования к программному обеспечению, объекты сцены и их структура, приводится декомпозиция задачи, а также рассматриваются схемы алгоритмов визуализации сцен объектов методом трассировки лучей.

### **2.1 Требования к программному обеспечению**

Пользовательский интерфейс разрабатываемой программы должен предоставлять следующие функциональные возможности:

- возможность выбора визуализируемой сцены из трех предложенных вариантов: сцена с двумя сферами, сцена с кубом, сферой и фигурой коня, сцена со всеми шахматными фигурами;
- возможность выбора объекта сцены путем нажатия на него мышкой или его выбора в списке объектов;
- возможность изменения цвета и коэффициента отражения выбранного объекта сцены;
- возможность выбора типа закраски объекта: по методу Гуро и по методу Фонга;
- возможность включения режима глубины поля, при этом возможно изменение расстояния от камеры до фокальной точки;
- возможность включения режима сглаживания изображения.

К разработанному ПО предъявляются следующие требования:

- чтение моделей шахматных фигур производится из объектных файлов при первом построении выбранной сцены в рамках одного запуска программы, модели шахматных фигур представлены полигональной сеткой;
- модели шара и куба представляются в программе аналитически;
- программа должна корректно обрабатывать ввод некорректных данных.

### **2.2 Формализация объектов синтезируемой сцены**

Синтезируемые сцены состоят из следующих типов объектов:

- точечный источник света, представимый положением в пространстве и интенсивностью;
- сфера, представимая точкой центра и радиусом;
- куб, грани которого параллельны координатным плоскостям. В программе представлен точкой центра и длиной стороны;
- шахматные фигуры: король, ферзь, ладья, слон, конь и пешка. Фигуры описываются точкой положения центра нижней грани, а также массивом треугольных полигонов, из которых составлена фигура. Треугольные полигоны, в свою очередь, описаны тремя вершинами и тремя нормальными в этих вершинах;

— камера, характеризующаяся положением в пространстве и направлением вектора наблюдения.

## 2.3 Декомпозиция задачи

В результате декомпозиции задачи были получены подзадачи, показанные на рисунках 2.1 - 2.3.

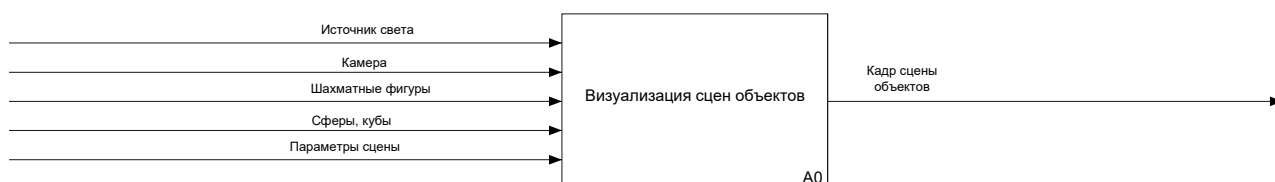


Рисунок 2.1 — Верхний уровень декомпозиции задачи

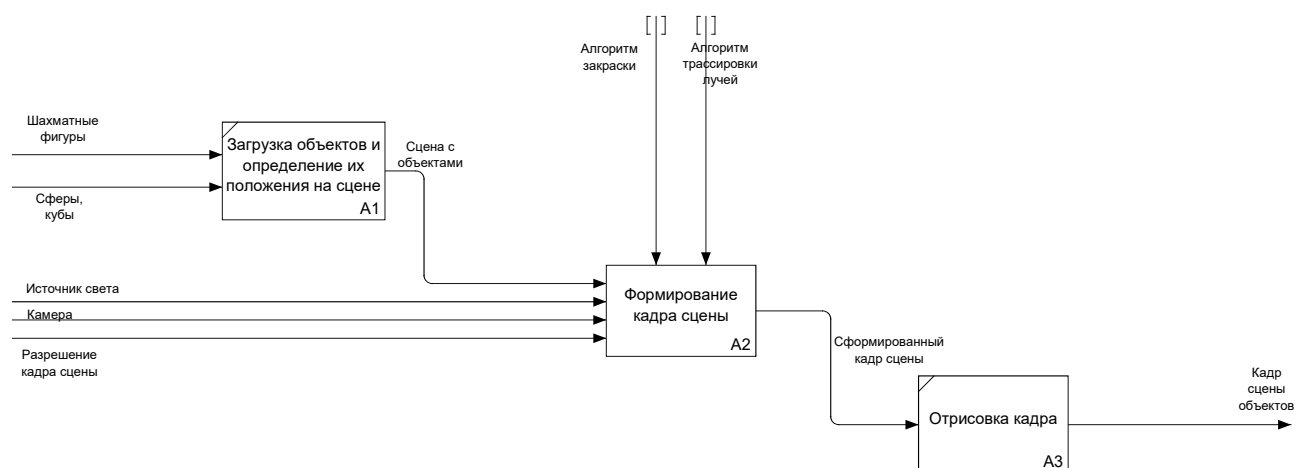


Рисунок 2.2 — Декомпозиция блока A0

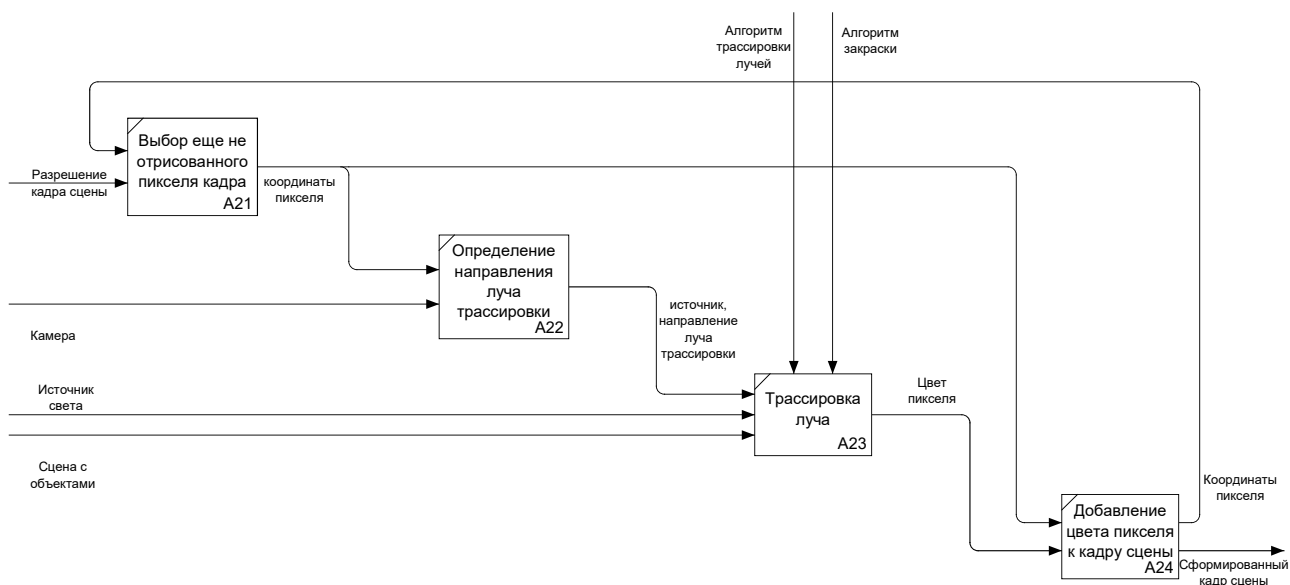


Рисунок 2.3 — Декомпозиция блока A2

Блок A23 трассировки лучей реализован в виде схемы алгоритма, приведенной на рисунке 2.4.

## 2.4 Алгоритм трассировки лучей

На рисунке 2.4 приведена схема алгоритма трассировки лучей.

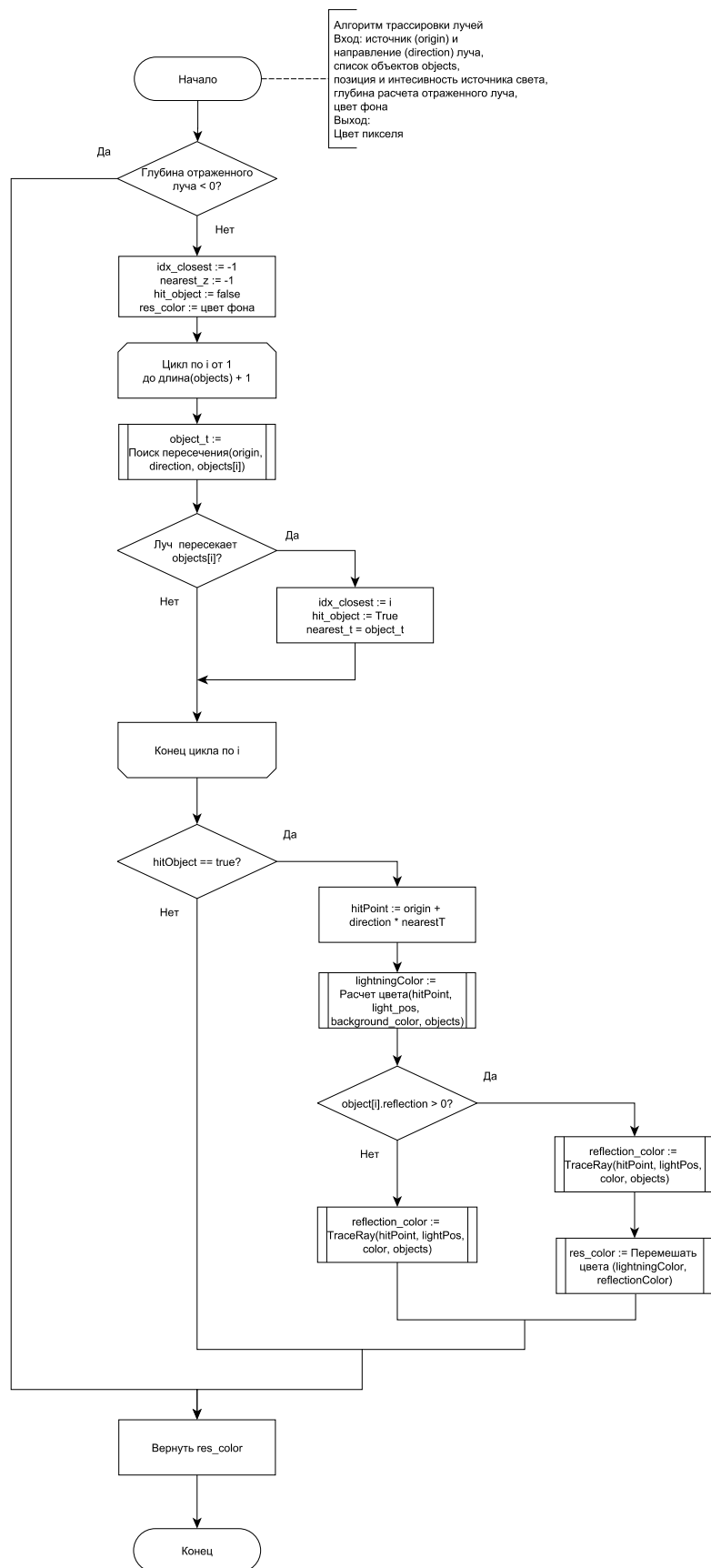


Рисунок 2.4 — Схема алгоритма трассировки лучей

## 2.5 Алгоритмы нахождения точки пересечения луча с объектом сцены

Как было описано выше, в данной работе синтезируемые объекты сцены представлены одним из следующих типов:

- сфера;
- куб;
- шахматные фигуры, поверхность которых состоит из треугольных полигонов.

Ниже описаны алгоритмы пересечения луча трассировки с каждым из этих типов объектов.

### 2.5.1 Алгоритм нахождения точки пересечения луча со сферой

Пусть  $\vec{c}$  – центр сферы,  $r$  – ее радиус. При известных  $\vec{p}_0$  и  $\vec{u}$  – точке начала и вектора направления луча соответственно, можно выразить точку луча следующим образом:

$$p(\vec{t}) = \vec{p}_0 + \vec{u} \cdot t, \quad t \geq 0. \quad (2.1)$$

Тогда для точки  $p(\vec{t})$  луча, лежащей на сфере, справедливо равенство

$$|p(\vec{t}) - \vec{c}| - r = 0. \quad (2.2)$$

Подставляя 2.1 в 2.2 получим

$$|\vec{p}_0 + t \cdot \vec{u} - \vec{c}| - r = 0, \quad (2.3)$$

откуда получаем квадратное уравнение

$$A^2 \cdot t + B \cdot t + C = 0, \quad (2.4)$$

где  $A = \vec{u}$ ,  $B = 2\vec{u}(\vec{p}_0 - \vec{c})$ ,  $C = (\vec{p}_0 - \vec{c})^2 - r^2$ .

Таким образом,  $t$  выражается как

$$t = \frac{-B \pm \sqrt{B^2 - 4C}}{2}, \quad (2.5)$$

при этом выбирается меньшее  $t$  (ближайшая точка) и проверяется на положительность.

На рисунке 2.5 приведена схема алгоритма, реализующая описанные вычисления.

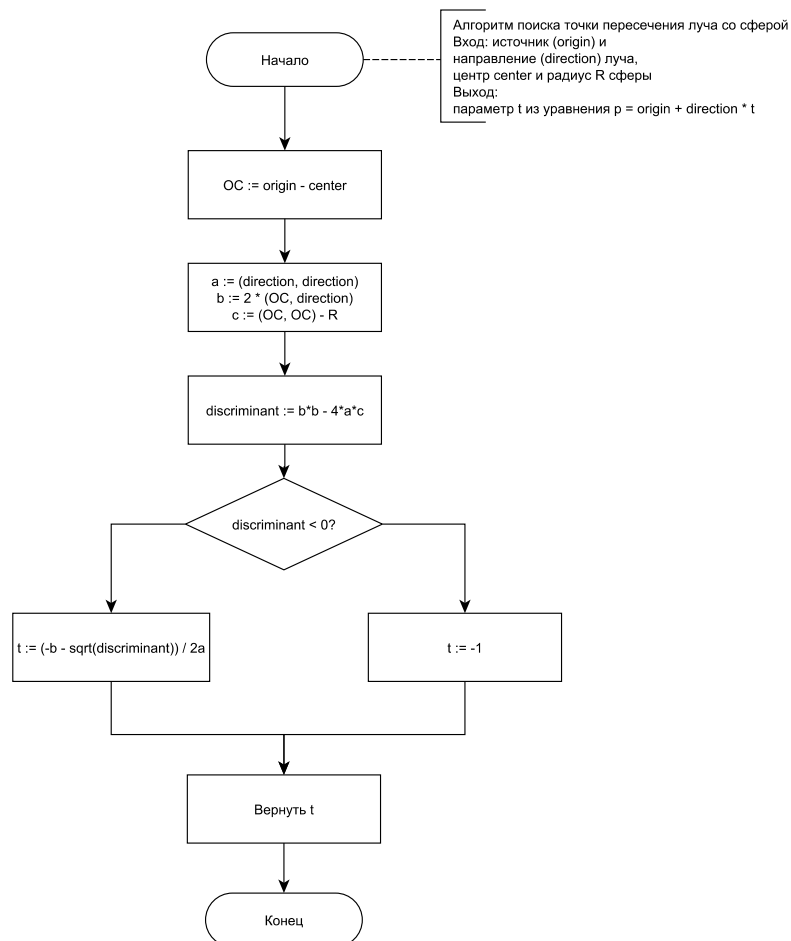


Рисунок 2.5 — Схема алгоритма нахождения точки пересечения луча и сферы

## 2.5.2 Алгоритм нахождения точки пересечения луча и куба

Алгоритм нахождения точки пересечения луча с кубом использует представление луча в форме, описанной формулой 2.1, а также идею последовательного обрезания луча каждой из граней куба [6]. Алгоритм представлен на схемах 2.6-2.7.

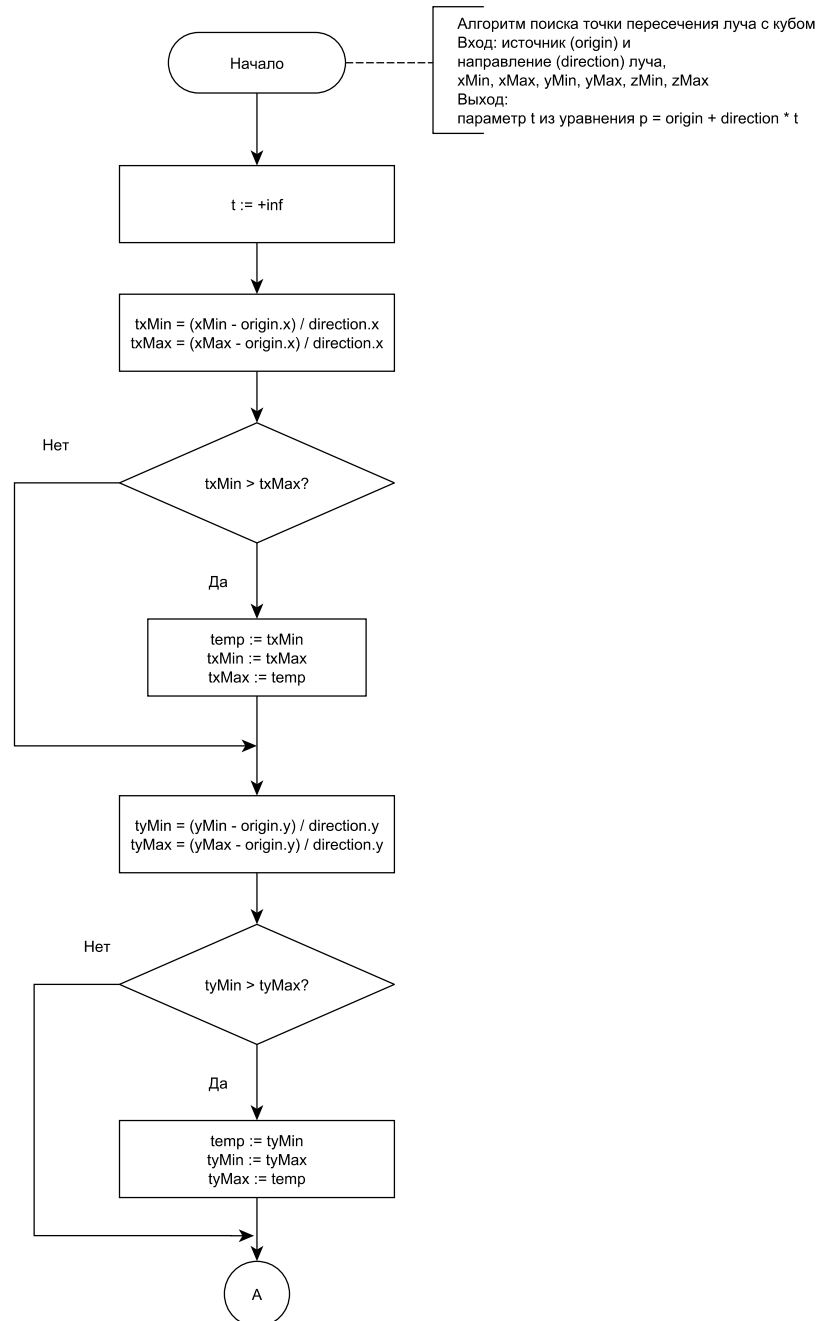


Рисунок 2.6 — Схема алгоритма нахождения точки пересечения луча и сферы



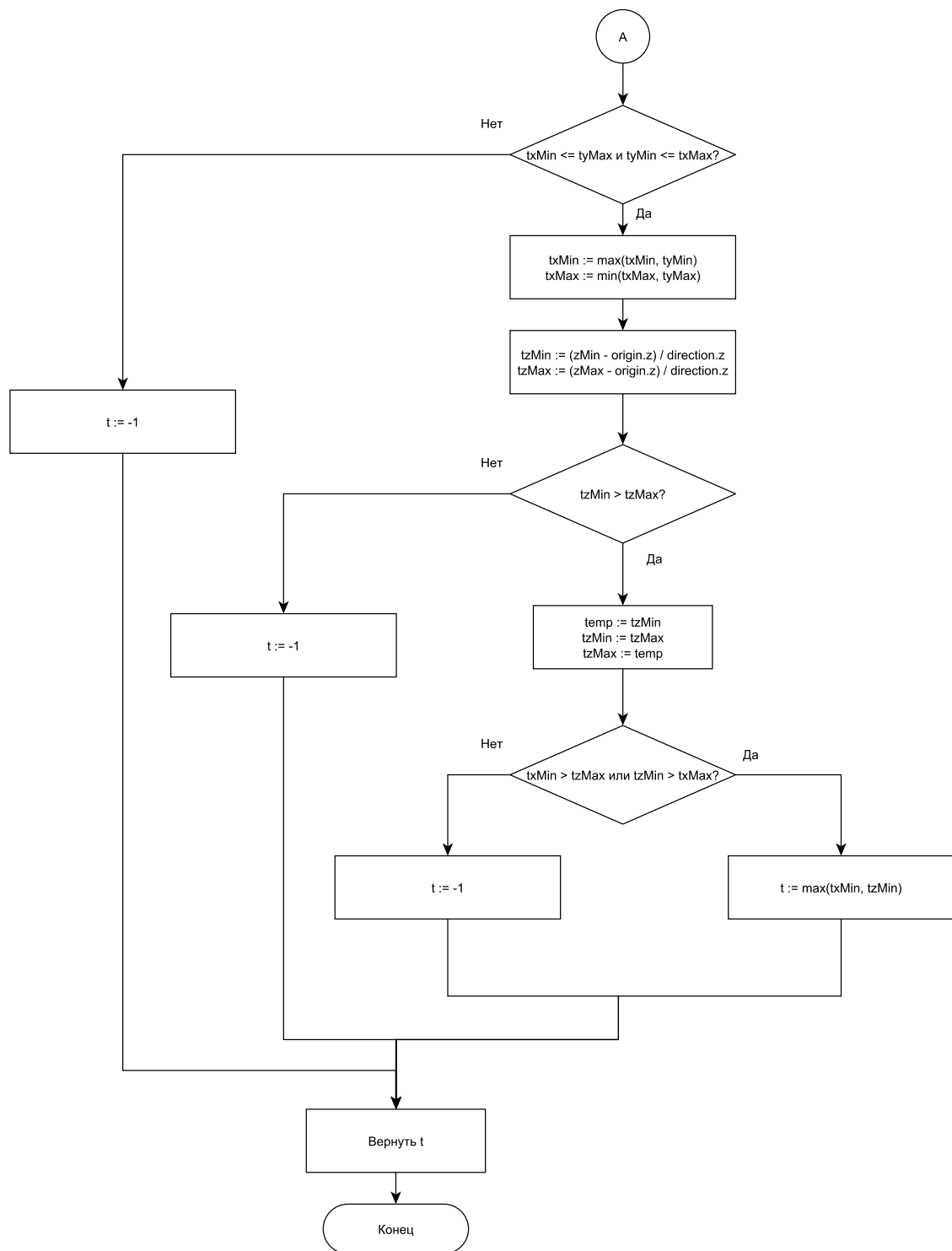


Рисунок 2.7 — Схема алгоритма нахождения точки пересечения луча и сферы

### 2.5.3 Алгоритм нахождения точки пересечения луча и шахматной фигуры

Поскольку модели шахматных фигур состоят из треугольных полигонов, задача нахождения точки пересечения луча и фигуры включает в себя подзадачу поиска точки пересечения с треугольником. Для решения задачи был выбран алгоритм Моллера — Трубора, позволяющий

решить задачу без предварительного вычисления уравнения плоскости, содержащей треугольник [8].

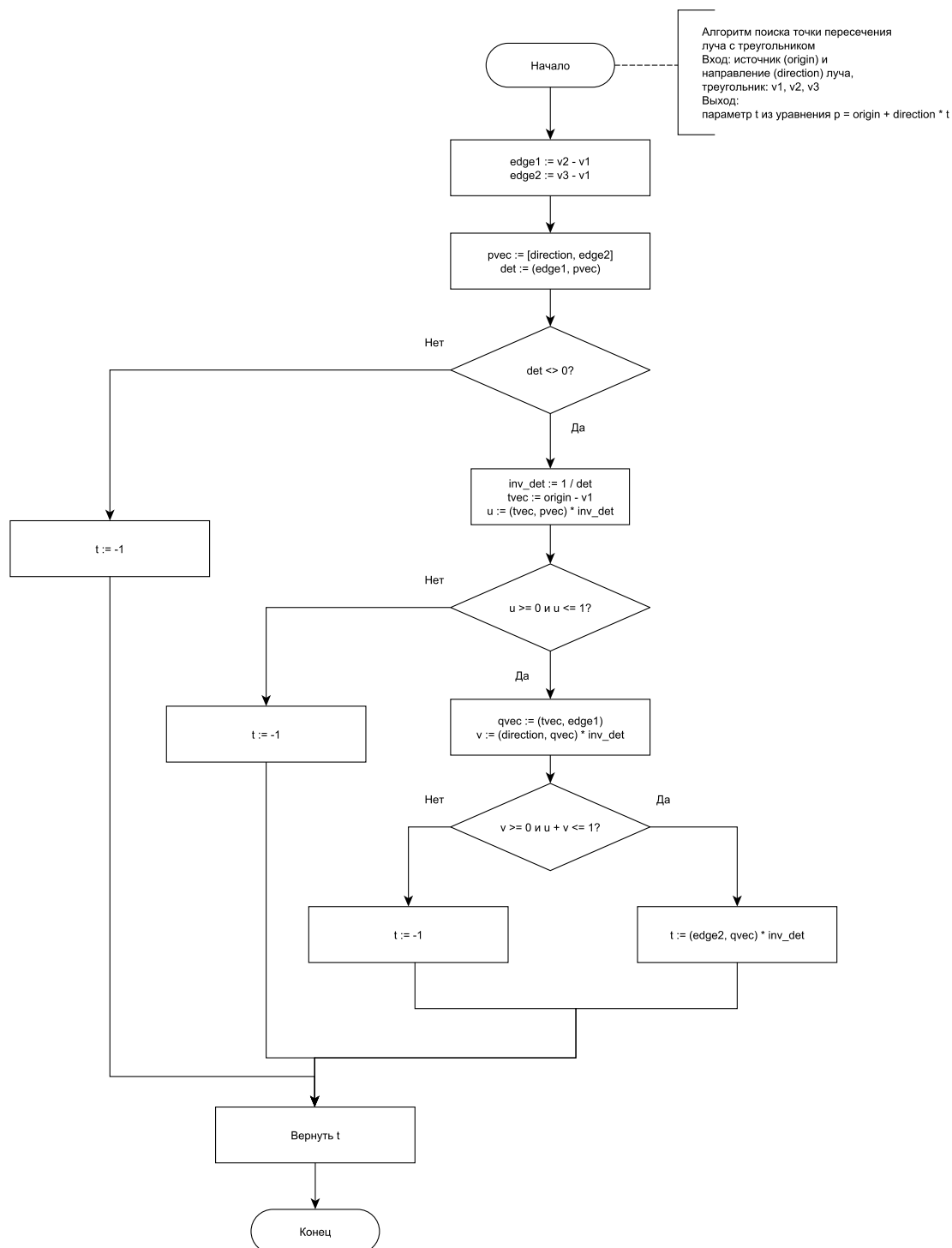


Рисунок 2.8 — Схема алгоритма нахождения точки пересечения луча и треугольника

Алгоритм нахождения точки пересечения луча и шахматной фигуры использует предварительный расчет описанной сферы при создании шахматной фигуры, и далее использует сферу, проверяя пересечение луча с описанной сферой прежде чем итерироваться по всем треугольникам фигуры: если луч не пересекает описанную сферу, то он не может пересечь фигуру. При описанной проверке используется алгоритм нахождения точки пересечения луча со сферой 2.5.

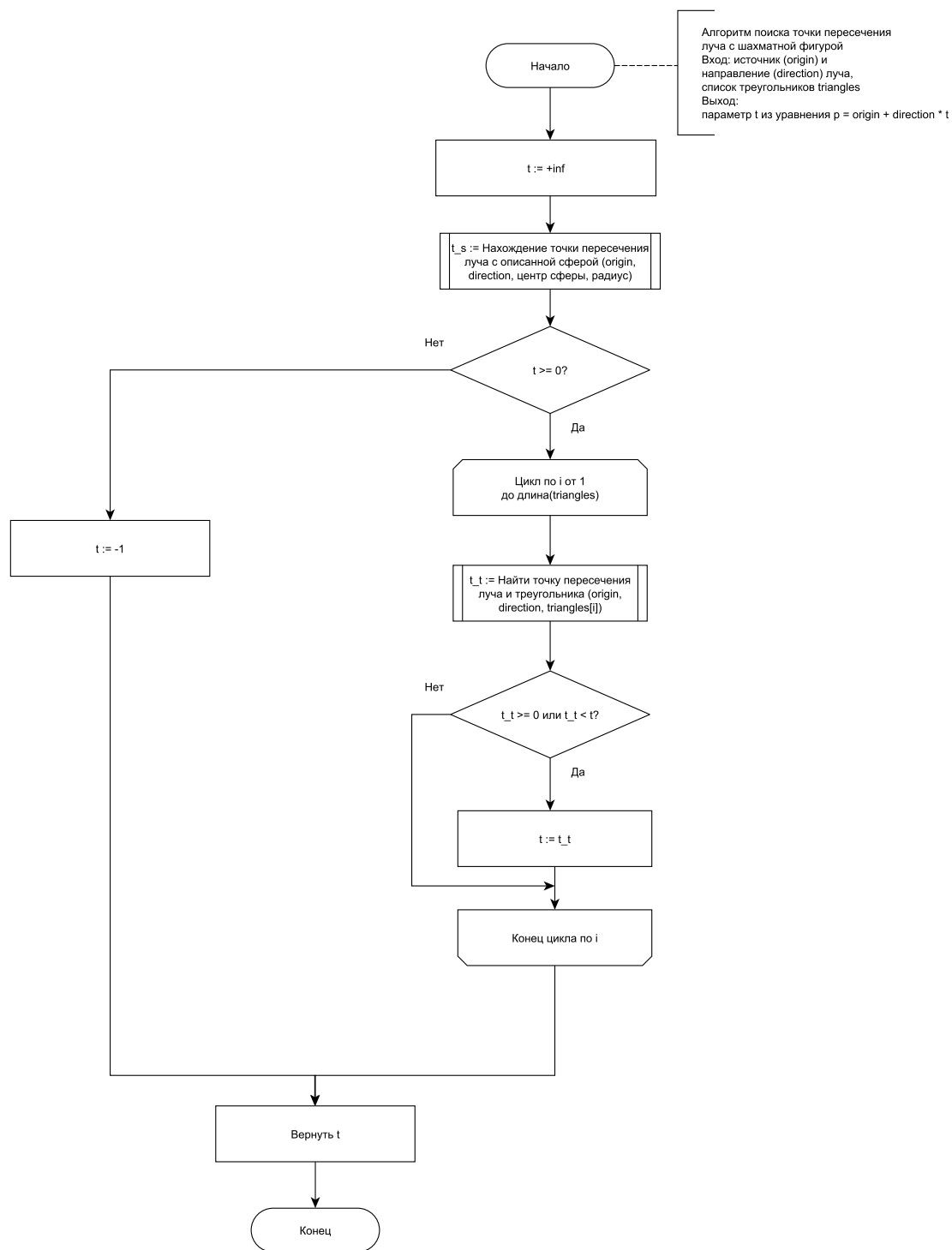


Рисунок 2.9 — Схема алгоритма нахождения точки пересечения луча и шахматной фигуры

## Вывод

В данном разделе были формализованы требования к реализуемому ПО, объекты сцены и их структура, приведена декомпозиция задачи, а также рассмотрены схемы алгоритмов визуализации сцены объектов методом трассировки лучей.

## **3 Технологическая часть**

В данной части рассматриваются описываются выбранные средства реализации, структура классов программы, а также приводятся листинги реализации алгоритмов и приводится демонстрационный пример интерфейса программы.

### **3.1 Средства реализации**

Для написания курсового проекта был выбран язык *C#* версии 7.3 [9], предоставляющий достаточный набор инструментов для реализации спроектированного ПО. В частности, язык поддерживает объектно-ориентированную модель разработки, что позволяет выделять отдельные сущности задачи в виде классов.

В качестве интегрированной среды разработки была выбрана Microsoft Visual Studio 2022 [10], предоставляющая достаточный функционал для написания, профилирования и отладки написанной программы, равно как и реализация пользовательского графического интерфейса ПО.

### **3.2 Структура программы**

На рисунке 3.1 представлена диаграмма разработанных классов.

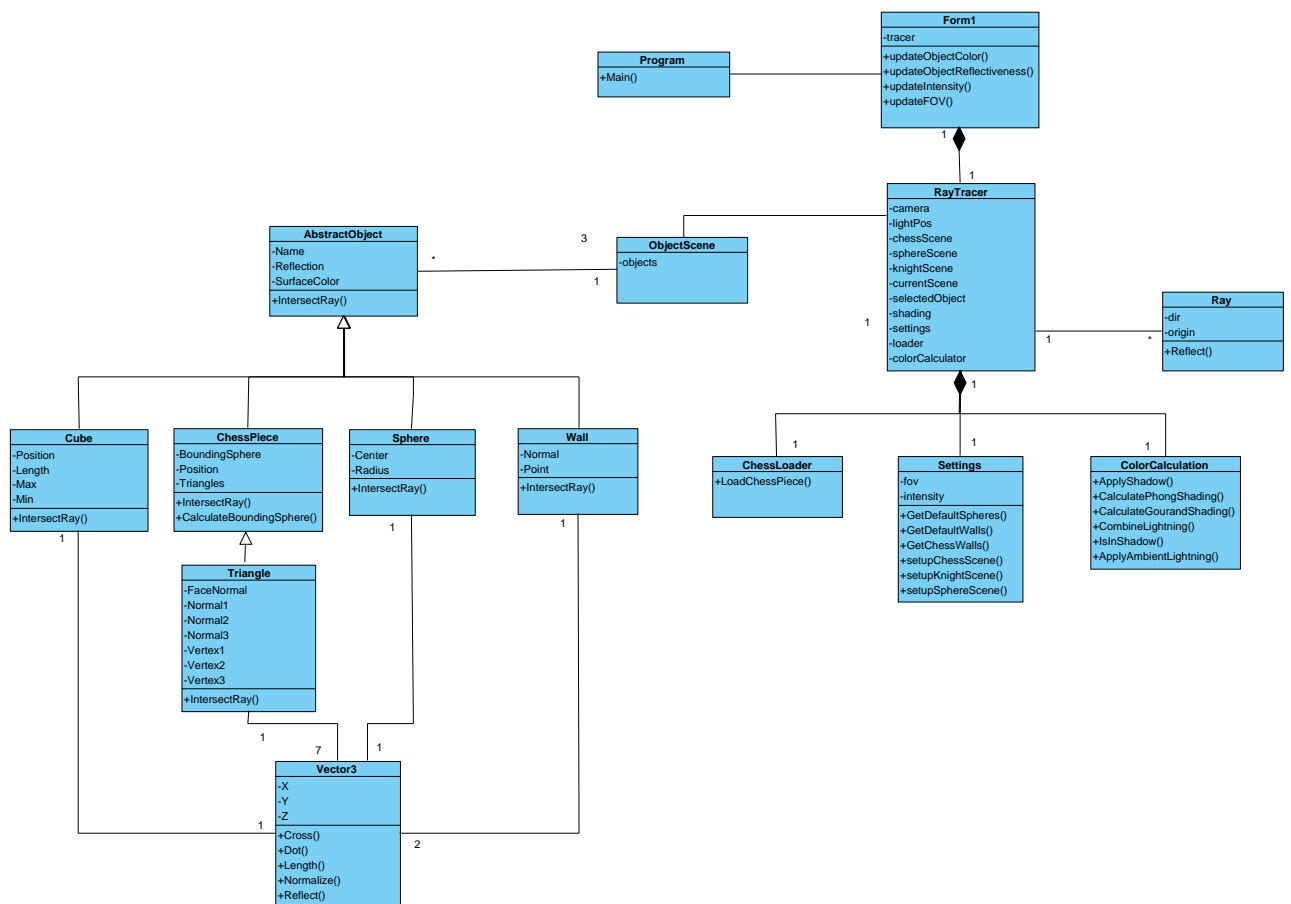


Рисунок 3.1 — Диаграмма классов объектов сцены

Разработанная программа состоит из следующих классов:

1) структурные классы программы

- Program – точка входа в программу;
- Form1 – класс, представляющий графический интерфейс программы;

2) базовые математические классы

- Ray – класс луча трассировки, также используется для представления камеры в программе;
- Vector3 – класс трехмерного вектора, поддерживающий математические операции над векторами;

3) классы объектов сцены

- AbstractObject – абстрактный класс, определяющий общий для всех объектов интерфейс;
- Cube – класс, представляющий куб в качестве объекта сцены;
- ChessPiece – класс, представляющий шахматную фигуру в качестве объекта сцены;
- Triangle – класс для представления треугольных полигонов, из которых состоят шахматные фигуры;
- Sphere – класс, представляющий сферу в качестве объекта сцены;

- Wall – класс, представляющий стену в качестве объекта сцены;
- 4) ObjectScene – класс, представляющий сцену объектов. Содержит список всех объектов сцены.
- 5) вспомогательные классы
  - ChessLoader – класс, предназначенный для загрузки моделей шахматных фигур из объектных файлов;
  - Settings – класс с настройками для трассировки лучей;
  - ColorCalculation – класс, реализующий логику реализации выбранной модели освещения;
- 6) RayTracing – класс, реализующий основную логику приложения.

### **3.3 Реализация алгоритмов**

В соответствии со схемой, изображенной на рисунке 2.4 был реализован алгоритм трассировки лучей, приведенный на листинге 3.1.

```

1 private Color TraceRay(Ray ray, ObjectScene scene, Vector3 lightPos,
2   Color backgroundColor, int depth)
3 {
4   if (depth <= 0)
5     return backgroundColor;
6
7   double closestDistance = double.MaxValue;
8   Vector3 hitNormal = new Vector3(0, 0, 0);
9   AbstractObject closestObject = null;
10
11   foreach (var obj in scene.objects)
12   {
13     if (obj.IntersectRay(ray, out double dist, out Vector3 normal) &&
14       dist < closestDistance)
15     {
16       closestDistance = dist;
17       hitNormal = normal;
18       closestObject = obj;
19     }
20   }
21
22   if (closestObject == null)
23     return backgroundColor;
24
25   Vector3 hitPoint = ray.origin + ray.dir * closestDistance;
26   Color objectColor = ((dynamic)closestObject).SurfaceColor;
27   Color lightingColor = CalculateLighting(hitPoint, hitNormal,
28     lightPos, objectColor, scene);
29
30   if (closestObject.Reflection > 0)
31   {
32     Vector3 reflectionDir = ray.dir.Reflect(hitNormal);
33     Color reflectionColor = TraceRay(new Ray(hitPoint, reflectionDir)
34       , scene, lightPos, backgroundColor, depth - 1);
35     lightingColor = MixColors(lightingColor, reflectionColor,
36       closestObject.Reflection);
37   }
38   return lightingColor;
39 }

```

Листинг 3.1 — Алгоритм трассировки лучей

На листингах 3.2 - 3.5, 4.1 приведены алгоритмы пересечения объектов сцены с лучом.

```
1 public override bool IntersectRay(Ray ray, out double t, out
   Vector3 Normal)
2 {
3     Vector3 oc = ray.origin - Center;
4     double a = ray.dir.Dot(ray.dir);
5     double b = 2.0 * oc.Dot(ray.dir);
6     double c = oc.Dot(oc) - Radius * Radius;
7     double discriminant = b * b - 4 * a * c;
8     Normal = null;
9
10    if (discriminant < 0)
11    {
12        t = 0;
13        return false;
14    }
15
16    t = (-b - Math.Sqrt(discriminant)) / (2.0 * a);
17
18    if (t > 0)
19        Normal = (ray.origin + ray.dir * t - Center).Normalize();
20
21    return t > 0;
22 }
```

Листинг 3.2 — Алгоритм поиска точки пересечения луча со сферой



```

1      public override bool IntersectRay(Ray ray, out double t, out
      Vector3 normal)
2      {
3          t = 0;
4          normal = new Vector3(0, 0, 0);
5
6          double denom = Normal.Dot(ray.dir);
7          if (Math.Abs(denom) > 1e-6)
8          {
9              t = (Point - ray.origin).Dot(Normal) / denom;
10             if (t >= 0)
11             {
12                 normal = Normal;
13                 return true;
14             }
15         }
16         return false;
17     }

```

Листинг 3.3 — Алгоритм поиска точки пересечения луча с плоскостью (стеной)

Реализация алгоритма пересечения луча трассировки с кубом представлена в листинге 4.1 (см. приложение Б).

```

1      public bool IntersectRay(Ray ray, out double distance, out Vector3
      barycentricCoords)
2      {
3          barycentricCoords = new Vector3(0, 0, 0);
4          distance = 0;
5          Vector3 edge1 = Vertex2 - Vertex1;
6          Vector3 edge2 = Vertex3 - Vertex1;
7          Vector3 pvec = ray.dir.Cross(edge2);
8          double det = edge1.Dot(pvec);
9          const double epsilon = 1e-8;
10         if (Math.Abs(det) < epsilon)
11             return false;
12
13         double invDet = 1.0 / det;
14
15         Vector3 tvec = ray.origin - Vertex1;
16         double u = tvec.Dot(pvec) * invDet;
17
18         if (u < 0.0 || u > 1.0)
19             return false;
20
21         Vector3 qvec = tvec.Cross(edge1);
22         double v = ray.dir.Dot(qvec) * invDet;
23
24         if (v < 0.0 || u + v > 1.0)
25             return false;
26
27         distance = edge2.Dot(qvec) * invDet;
28
29         if (distance > epsilon)
30         {
31             barycentricCoords = new Vector3(1 - u - v, u, v);
32             return true;
33         }
34
35         return false;
36     }

```

Листинг 3.4 — Алгоритм поиска точки пересечения луча с треугольным полигоном

```

1      public override bool IntersectRay(Ray ray, out double distance, out
        Vector3 interpolatedNormal)
2      {
3          distance = double.MaxValue;
4          interpolatedNormal = new Vector3(0, 0, 0);
5
6          if (!BoundingSphere.IntersectRay(ray, out double sphereDist, out
            Vector3 Normal))
7          {
8              return false;
9          }
10
11         bool hit = false;
12         foreach (var triangle in Triangles)
13         {
14             if (triangle.IntersectRay(ray, out double dist, out Vector3
                barycentricCoords) && dist < distance)
15             {
16                 distance = dist;
17                 hit = true;
18
19                 interpolatedNormal =
20                     barycentricCoords.X * triangle.Normal1 +
21                     barycentricCoords.Y * triangle.Normal2 +
22                     barycentricCoords.Z * triangle.Normal3;
23
24                 interpolatedNormal = interpolatedNormal.Normalize();
25             }
26         }
27
28         return hit;
29     }

```

Листинг 3.5 — Алгоритм поиска точки пересечения луча с шахматной фигурой

### 3.4 Функциональное тестирование

В таблице 3.1 приведены функциональные тесты для программы, визуализирующей трехмерные сцены объектов. Все тесты пройдены программой успешно.

Таблица 3.1 — Функциональные тесты (начало)

№	Состояние программы до теста	Изменяемый параметр/событие	Ожидаемый результат
1	—	Нажатие кнопки «Визуализировать»	Визуализация сцены с двумя сферами и первой камерой
2	—	Выбор сцены «Сцена с шахматными фигурами»; Нажатие кнопки «Визуализировать»	Визуализация сцены с шахматными фигурами и первой камерой
3	—	Выбор сцены «Сцена с конем и сферой»; Нажатие кнопки «Визуализировать»	Визуализация сцены с конем, кубом и сферой
4	Выбрана и визуализирована сцена с двумя сферами	Выбор камеры №2 Нажатие кнопки «Визуализировать»	Визуализация сцены с двумя сферами и второй камерой
5	Выбрана и визуализирована сцена с двумя сферами	Выбор камеры №3 Нажатие кнопки «Визуализировать»	Визуализация сцены с двумя сферами и третьей камерой
6	Выбрана и визуализирована сцена с двумя сферами и третьей камерой	Изменение поля зрения на 90 градусов Нажатие кнопки «Визуализировать»	Визуализация сцены с двумя сферами и третьей камерой с углом обзора 90 градусов
7	Выбрана и визуализирована сцена с конем и сферой и третьей камерой	Выбор фигуры коня нажатием левой кнопки мыши; Изменение цвета коня на желтый; Нажатие кнопки «Визуализировать»	Визуализация сцены с желтым конем и сферой с третьей камерой
8	Выбрана и визуализирована сцена с конем и сферой и третьей камерой	Выбор куба в списке объектов; Изменение зеркальности куба на 95%; Нажатие кнопки «Визуализировать»	Визуализация сцены с конем, зеркальным кубом и сферой с третьей камерой

Таблица 3.1 — Функциональные тесты (окончание табл. 3.1)

№	Состояние программы до теста	Изменяемый параметр/событие	Ожидаемый результат
9	Выбрана и визуализирована сцена с двумя сферами и первой камерой	Увеличение интенсивности света до 50%; Нажатие кнопки «Визуализировать»	Визуализация сцены с двумя сферами с первой камерой и с интенсивностью освещения 50% от максимальной
10	Выбрана и визуализирована сцена с двумя сферами и первой камерой	Включение сглаживания; Изменение числа лучей на 64; Нажатие кнопки «Визуализировать»	Визуализация сцены с двумя сферами с первой камерой и с сглаживаем
11	Выбрана и визуализирована сцена с двумя сферами и первой камерой	Включение глубины поля; Изменение расстояния до фокальной точки на 8; Нажатие кнопки «Визуализировать»	Визуализация сцены с двумя сферами с первой камерой и с эффектом глубины поля
12	Выбрана и визуализирована сцена с двумя сферами и первой камерой	Изменение типа закраски на метод Гуро; Нажатие кнопки «Визуализировать»	Визуализация сцены с двумя сферами с первой камерой с помощью закраски Гуро

### 3.5 Интерфейс программного обеспечения

На рисунке 3.2 приведена демонстрация интерфейса программы при ее запуске.

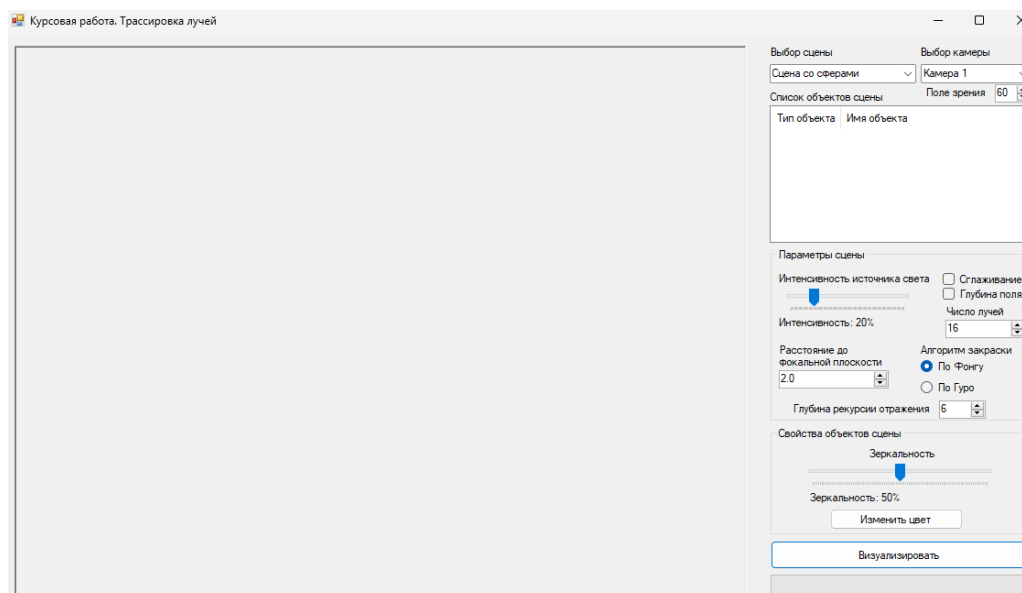


Рисунок 3.2 — Пользовательский интерфейс ПО

В левой части окна приложения расположена графическая область в виде прямоугольника для отображения результатов визуализации. На любой объект сцены можно произвести нажатие левой кнопки мыши, после чего выбранный объект будет выделен в окне «Список объектов сцены» и объект будет доступен для изменения параметров.

Пользователю на выбор предоставляются три варианта сцены, на каждом из них возможен выбор трех вариантов расположения камер. Изменяемым параметром камеры является ее поле зрения. Пример выбора сцены и камеры приведен на рисунке 3.3.

Ниже перечисленных элементов графического интерфейса расположено окно «Список объектов сцены», который заполняется наименованиями объектов сцены при нажатии кнопки «Визуализировать». В данном элементе можно выбирать объекты сцены нажатием левой кнопки мыши на нужную строку для изменения свойств выбранного объекта.

Группа параметров «Параметры сцены» позволяет настроить общие для всех объектов параметры. Пользователю предоставлена возможность изменить интенсивность источника света, включить сглаживание (Anti-Aliasing) [11] и глубину поля (Depth Of Field), выбрать число лучей для двух описанных эффектов, изменить расстояние от камеры до фокальной точки (расстояние измеряется по вектору направлению взгляда), а также выбрать алгоритм закраски – по методу Фонга или по методу Гуро.

Группа параметров «Свойства объектов сцены» позволяет настроить зеркальность поверхности выбранного объекта и его цвет. Пример выбора объекта сцены и изменения его зеркальности приведен на рисунке 3.4.

Для вступления в силу любого из перечисленных изменений параметров необходимо нажать кнопку «Визуализировать», после чего дождаться визуализации сцены (под кнопкой расположена полоса статуса для отображения прогресса визуализации).

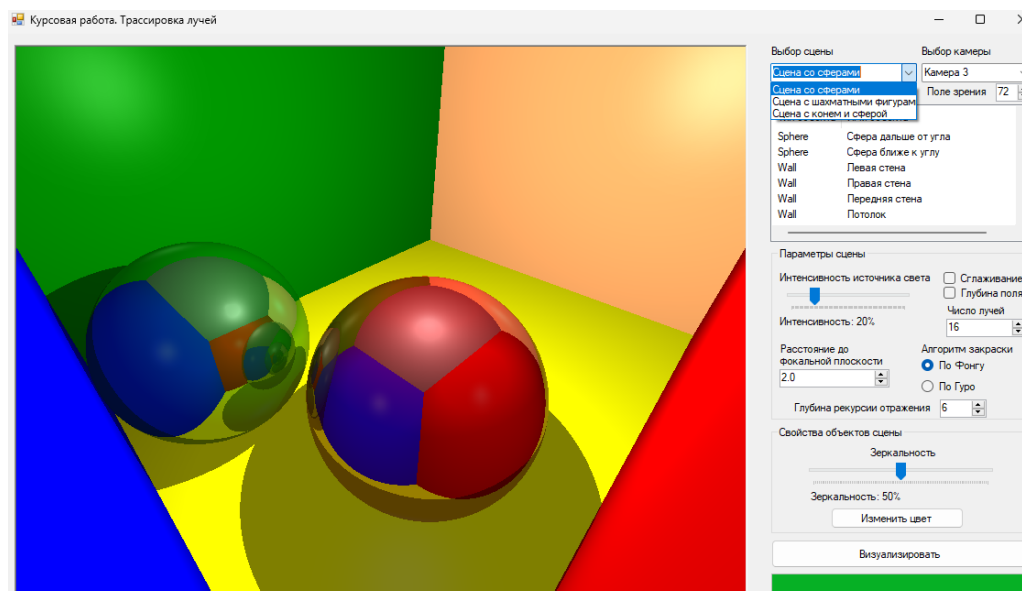


Рисунок 3.3 — Пример выбора сцены и камеры

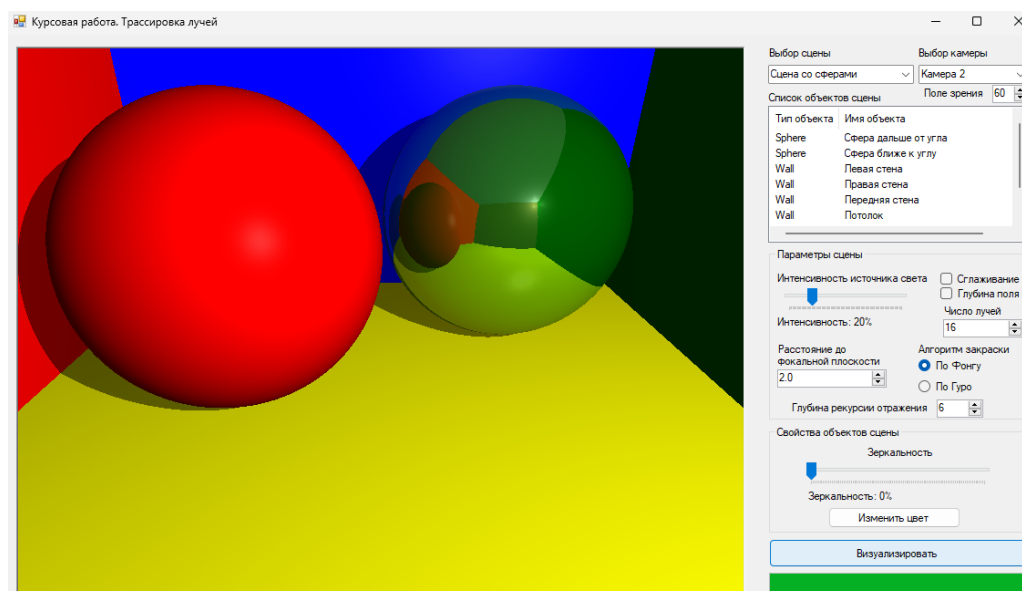


Рисунок 3.4 — Пример выбора объекта на сцене и изменения его зеркальности

## Вывод

В данном разделе были выбраны средства реализации, описана структура классов программы, а также приведено описание интерфейса ПО.

## 4 Исследовательская часть

В данном разделе приведены технические характеристики устройства, на котором проводились замеры времени выполнения программного обеспечения, а также результаты исследования.

### 4.1 Технические характеристики

Исследование по замерам времени выполнения программы проводилось на персональном компьютере, оснащённом следующими компонентами:

- процессор Intel Core i7-12700KF с 12 ядрами: 8 производительных и 4 энергоэффективных, – с суммарным числом потоков, равным 20 [12];
- оперативной памятью объёмом 32 Гб;
- операционная система Windows 11 Home 64-разрядная [13].

### 4.2 Описание исследования

В рамках данного исследования проводятся замеры времени растеризации сцены в зависимости от числа используемых потоков. В исследовании участвуют две сцены: сцена с двумя сферами с коэффициентами отражения, равными 0.5, а также сцена с одной сферой с таким же коэффициентом зеркального отражения.

Для каждой сцены и каждого числа потоков рассчитывается и приводится медианное время по 50 замерам.

В рамках исследования по замерам времени использовался класс *Stopwatch* [14] пространства имен *System.Diagnostics* [15] языка программирования C# [9].

### 4.3 Результаты исследования

Результаты исследования зависимости времени растеризации сцены объектов в зависимости от числа используемых потоков, приведены в таблице 4.1.



Таблица 4.1 — Результаты исследования зависимости медианного времени растеризации сцены объектов в зависимости от числа используемых потоков

Число потоков	Сцена с одной сферой	Сцена с двумя сферами
1	626.01	694.57
2	206.82	225.64
4	206.73	225.20
8	206.62	225.17
12	206.26	225.14
16	206.16	224.97
20	205.92	223.79
32	207.12	225.35
64	207.43	225.96

Результаты визуализации полученных данных приведены на рисунках 4.1 - 4.2.

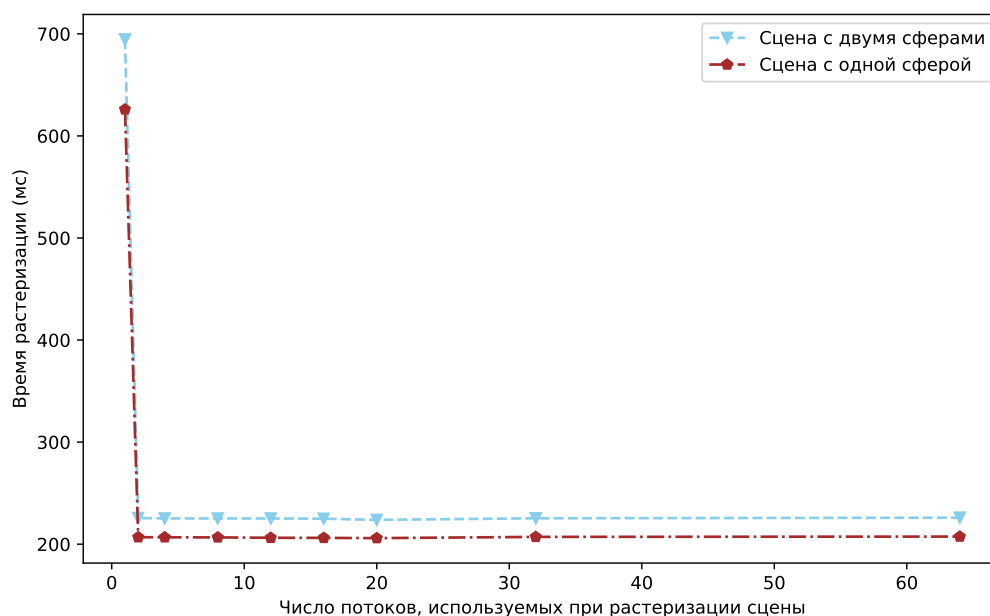


Рисунок 4.1 — График зависимости медианного времени растеризации сцены от числа используемых потоков

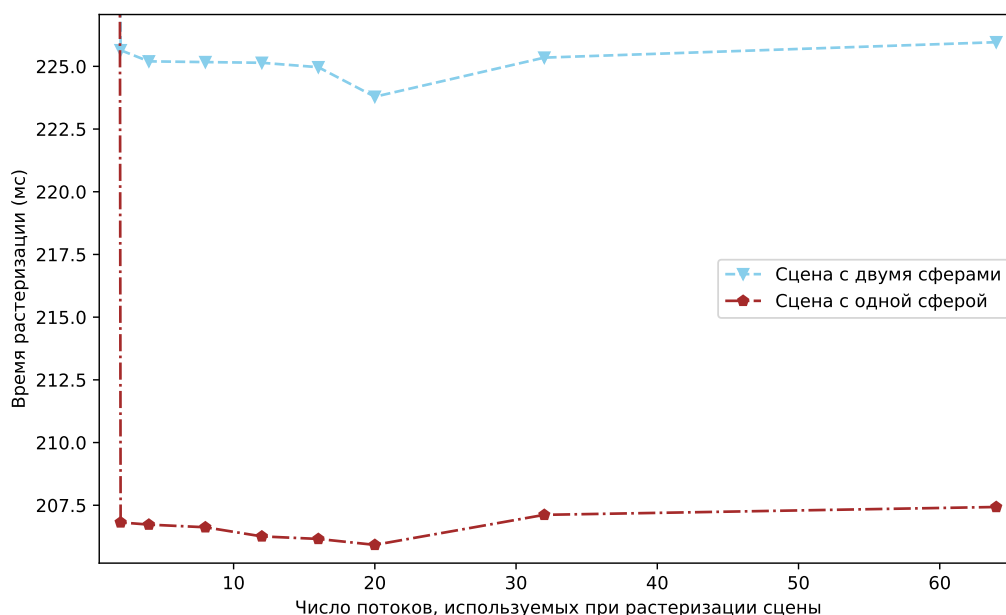


Рисунок 4.2 — График зависимости медианного времени растеризации сцены от числа используемых потоков (при числе потоков от 2 до 64)

По результатам проведения замеров было установлено, что наиболее эффективным числом потоков для растеризации сцены является 20, это число совпало с числом потоков процессора. Использование большего числа потоков, чем позволяет процессор, приводит к увеличению времени выполнения программы, поскольку потоки выстраиваются в очередь, а также происходят более частые переключения контекста. При этом использование хотя бы двух потоков приводит к росту временной эффективности на 67-68% относительно однопоточной версии программы. Результаты времени выполнения для 2 и 20 потоков различаются не более, чем на 0.5%.

В ходе замеров было также выявлено, что при увеличении числа сфер на сцене от 1 до 2 время растеризации сцены увеличилось в среднем на 8.4%.

## Вывод

В данном разделе было описано исследование замеров времени, описаны технические характеристики компьютера, на котором проводилось исследование, а также приведены результаты замеров.

Результаты замерного эксперимента совпали с ожидаемыми: наиболее эффективным числом потоков является число потоков, аппаратно поддерживаемых процессором. Также в ходе эксперимента было подтверждено, что при увеличении числа объектов на сцене растет и время растеризации сцены (в случае с увеличением числа сфер от 1 до 2 рост составил 8.4%).

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта была достигнута его цель: было разработано программное обеспечение для визуализации сцен трехмерных объектов с изменяемыми параметрами.

В ходе работы были решены все поставленные задачи:

- рассмотрены основные алгоритмы удаления невидимых линий и поверхностей, методы закраски, методы построения теней. На основании их анализа были выбраны алгоритмы и методы, наиболее подходящие для решения поставленной задачи;
- была проведена декомпозиция задачи, разработаны диаграммы функционального моделирования и схемы реализуемых алгоритмов;
- спроектированное ПО было реализовано выбранными средствами;
- было проведено исследование временной эффективности реализованного ПО.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Роджерс Д., Адамс Дж. Математические основы машинной графики. — Москва: Мир, 2001. — 500с.;
2. Шишкин Е. В., Боресков А. В. Компьютерная графика. Динамика, реалистичные изображения. — Москва: ДИАЛОГ-МИФИ, 1996. — 288с;
3. Польский С. В. Компьютерная графика: учебно-методическое пособие. — Москва: ГОУ ВПО МГУЛ, 2008. — 38с.;
4. Закрашивание. Рендеринг полигональных моделей [Электронный ресурс]. URL: [https://intuit.ru/studies/professional\\_skill\\_improvements/1283/courses/70/lecture/21/08?page=2](https://intuit.ru/studies/professional_skill_improvements/1283/courses/70/lecture/21/08?page=2) (дата обращения 06.10.24);
5. Построение теней [Электронный ресурс]. URL: <https://stratum.ac.ru/education/textbooks/kggraphic/additional/addit28.html> (дата обращения 06.10.24);
6. Glassner, Andrew. An Introduction to Ray Tracing. — New York: Academic Press, 1991. — 359p.;
7. Depth Of Field. The University of Washington [Электронный ресурс]. URL: <https://courses.cs.washington.edu/courses/csep557/99au/projects/trace/depthoffield.doc> (дата обращения 06.12.24);
8. Fast Minimum Storage Ray/Triangle Intersection [Электронный ресурс]. URL: <https://web.archive.org/web/20170517125238/http://www.cs.virginia.edu/~gfx/Courses/2003/ImageSynthesis/papers/Acceleration/Fast%20MinimumStorage%20RayTriangle%20Intersection.pdf> (дата обращения 04.12.2024);
9. Заметки о выпуске C# 7.3 [Электронный ресурс]. URL: <https://docs.microsoft.com/en-us/visualstudio/releasenotes/vs2017-relnotes-v15.7#cssharp> (дата обращения 04.12.2024);
10. Microsoft Visual Studio 2022 [Электронный ресурс]. URL: <https://visualstudio.microsoft.com/ru/vs/> (дата обращения 04.12.2024);
11. Сглаживание [Электронный ресурс]. URL: <https://www.nvidia.com/content/Control-Panel-Help/vLatest/ru-ru/mergedProjects/nv3dRUS/antialiasing.htm> (дата обращения 04.12.2024);
12. Технические характеристики процессора Intel Core i7-12700KF [Электронный ресурс]. URL: <https://www.intel.com/content/www/us/en/products/sku/134595/intel-core-i712700kf-processor-25m-cache-up-to-5-00-ghz/specifications.html> (дата обращения 04.12.2024);

13. Windows 11 Home [Электронный ресурс]. URL: <https://www.microsoft.com/ru-ru/software-download/windows11> (дата обращения 04.12.2024);
14. Документация класса Stopwatch пространства имен System.Diagnostics [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/dotnet/api/system.diagnostics.stopwatch?view=net-9.0&viewFallbackFrom=net-7.3> (дата обращения 04.12.2024);
15. Документация пространства имен System.Diagnostics [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/dotnet/api/system.diagnostics?view=net-7.3> (дата обращения 04.12.2024);

# ПРИЛОЖЕНИЕ А

Презентация к курсовой работе содержит \_\_ слайдов.

# ПРИЛОЖЕНИЕ Б

```
1 public override bool IntersectRay(Ray ray, out double t, out
   Vector3 normal)
2 {
3     t = double.MaxValue;
4     normal = new Vector3(0, 0, 0);
5     double tMin = (Min.X - ray.origin.X) / ray.dir.X;
6     double tMax = (Max.X - ray.origin.X) / ray.dir.X;
7
8     if (tMin > tMax) (tMin, tMax) = (tMax, tMin);
9
10    double tyMin = (Min.Y - ray.origin.Y) / ray.dir.Y;
11    double tyMax = (Max.Y - ray.origin.Y) / ray.dir.Y;
12
13    if (tyMin > tyMax) (tyMin, tyMax) = (tyMax, tyMin);
14    if ((tMin > tyMax) || (tyMin > tMax)) return false;
15
16    if (tyMin > tMin) tMin = tyMin;
17    if (tyMax < tMax) tMax = tyMax;
18
19    double tzMin = (Min.Z - ray.origin.Z) / ray.dir.Z;
20    double tzMax = (Max.Z - ray.origin.Z) / ray.dir.Z;
21
22    if (tzMin > tzMax) (tzMin, tzMax) = (tzMax, tzMin);
23
24    double epsilon = 1e-8;
25
26    if (tMin > tzMax + epsilon || tzMin > tMax + epsilon)
27        return false;
28
29    tMin = Math.Max(tMin, tzMin);
30    tMax = Math.Min(tMax, tzMax);
```

Листинг 4.1 — Алгоритм поиска точки пересечения луча с кубом (начало)

```

1      if (tMin < 0)
2      {
3          t = tMax;
4          if (tMax < 0)
5              return false;
6      }
7      else
8          t = tMin;
9
10     Vector3 hitPoint = ray.origin + ray.dir * t;
11     if (Math.Abs(hitPoint.X - Min.X) < epsilon) normal = new Vector3
        (-1, 0, 0);
12
13     else if (Math.Abs(hitPoint.X - Max.X) < epsilon) normal = new
        Vector3(1, 0, 0);
14     else if (Math.Abs(hitPoint.Y - Min.Y) < epsilon) normal = new
        Vector3(0, -1, 0);
15     else if (Math.Abs(hitPoint.Y - Max.Y) < epsilon) normal = new
        Vector3(0, 1, 0);
16     else if (Math.Abs(hitPoint.Z - Min.Z) < epsilon) normal = new
        Vector3(0, 0, -1);
17     else if (Math.Abs(hitPoint.Z - Max.Z) < epsilon) normal = new
        Vector3(0, 0, 1);
18     return true;
19 }

```

Листинг 4.1 — Алгоритм поиска точки пересечения луча с кубом (конец)