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 сцен объектов с помощью метода трассировки лучей»

Студент ИУ7-51Б (группа) _____ (подпись, дата) Н.С. Могилин (И.О. Фамилия)

Руководитель курсового проекта _____ (подпись, дата) Д.Ю. Мальцева (И.О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **6**

1 Аналитическая часть **7**

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 Варнока 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

2 Конструкторская часть **12**

2.1 Требования к программному обеспечению 12

2.2 Формализация объектов синтезируемой сцены 12

2.3 Декомпозиция задачи 13

2.4 Алгоритм трассировки лучей 15

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

2.6 Вывод 21

3 Технологическая часть **22**

3.1 Средства реализации 22

3.2 Структура программы 22

3.3 Реализация алгоритмов 23

3.4 Интерфейс программного обеспечения 29
3.5 Вывод 31

4 Исследовательская часть 32
4.1 Вывод 32

ЗАКЛЮЧЕНИЕ 33

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 34

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 создания реалистичного изображения сцен трехмерных 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:

$$I_{\text{д}} = I_{\text{и}} cos\theta \cdot k_{\text{д}}, \tag{1.1}$$

где $I_{\text{и}}$ – интенсивность источника освещения, θ – угол между нормалью к поверхности в точке отражения и вектором, направленным из точки отражения к источнику света, $k_{\text{д}}$ -– коэффициент диффузного отражения.

Зеркальная составляющая явления отражения света описывается уравнением:

$$I_{\text{з}} = I_{\text{и}} cos^n\alpha \cdot k_{\text{з}}, \tag{1.2}$$

где $I_{\text{и}}$ – интенсивность источника освещения, α – угол падения луча, $k_{\text{з}}$ – коэффициент зеркального отражения, степень n косинуса характеризует концентрированность света в направлении отражения.

Рассеяние света в компьютерной графике обычно описывают следующей формулой:

$$I' = I_{\text{р}} k_{\text{р}}, \tag{1.3}$$

где $I_{\text{р}}$ – интенсивность рассеянного света, $k_{\text{р}}$ – 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:

$$I = I_{\text{р}} k_{\text{р}} + \frac{I_{\text{и}} cos\theta \cdot k_{\text{д}} + I_{\text{и}} cos^n\alpha \cdot k_{\text{з}}}{d + K}, \tag{1.4}$$

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].

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].

1.2.2 Алгоритм Варнока

Алгоритм Варнока основывается на рекурсивном разбиении окна на подокна всякий раз, когда оно не пусто, до окна размеров в 1 пиксель, после чего определяется ближайший к наблюдателю объект, пиксель закрашивается цветом этого объекта. Рекурсивность алгоритма является главным недостатком алгоритма: окно каждый раз делится на 4 подокна, если размер окна больше одного пикселя, или оно содержит несколько объектов, или пересекается с объектом.

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. Основной недостаток алгоритма – большой объем требуемой памяти [1].

1.2.4 Алгоритм обратной трассировки лучей

Метод прямой и обратной трассировки лучей заключается в том, что от момента испускания лучей источником света до момента попадания в камеру, траектории лучей отслеживаются, и рассчитываются пересечения лучей с лежащими на траектории объектами. При этом луч может быть поглощен, диффузно и/или зеркально отражен или, в случае прозрачности некоторых объектов, преломлен [**?**].

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 [3].

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 B, C, D вычислить усреднением нормалей к многоугольникам, то они будут одинаково ориентированы, то есть интенсивность в этих точках будет равной. При линейной интерполяции от B до D значение интенсивности получится постоянным, и поверхность на данном участке будет выглядеть плоской [3].

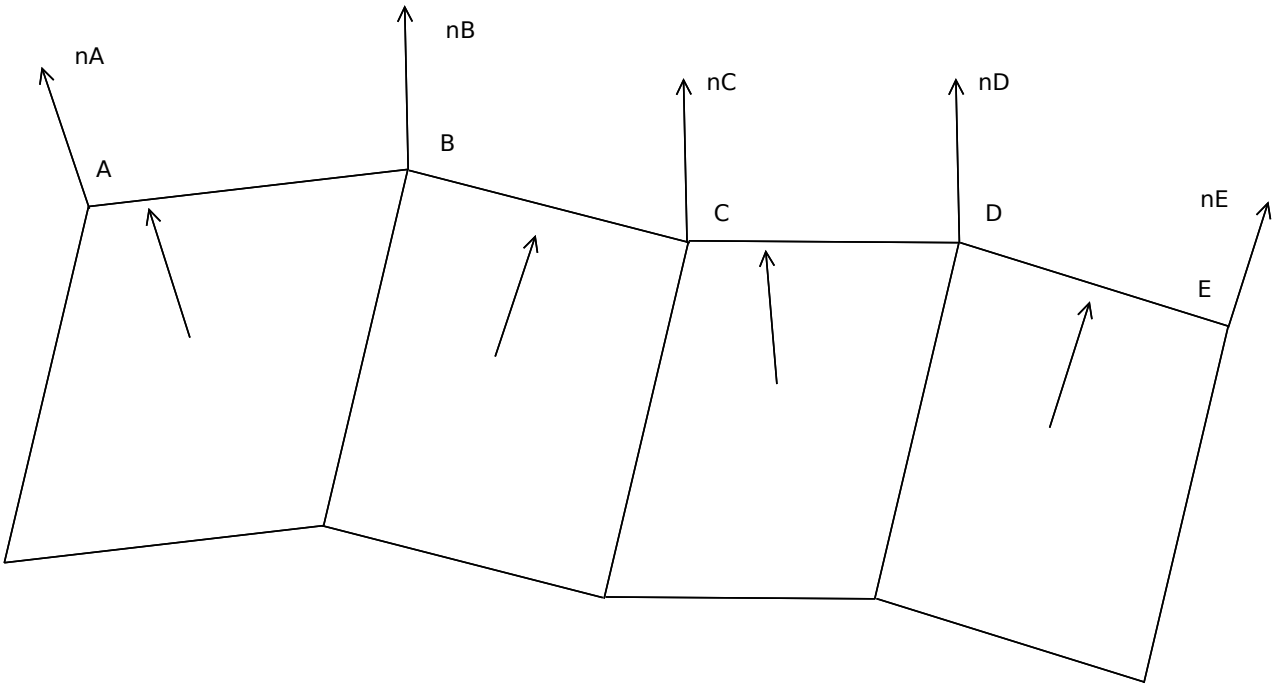

Рисунок 1.1 — Пример некорректной работы закраски Гуро

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 [3].

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 [4].

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.

Вывод

В данном разделе были рассмотрены основные методы построения реалистичных изображений. Для удаления невидимых ребер, учета отражения света и теней 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 плоскости;

— 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 нормалями в этих вершинах;

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

2.3 Декомпозиция задачи

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

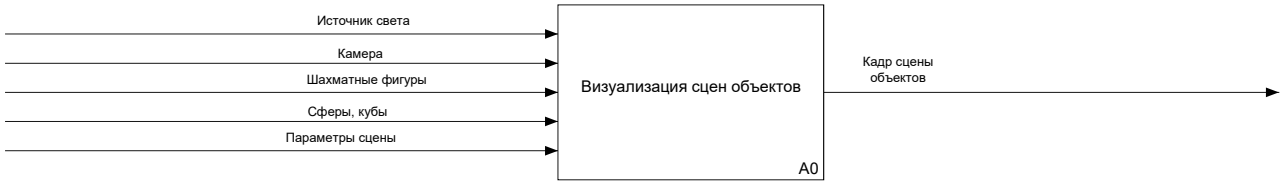

Рисунок 2.1 — Верхний уровень декомпозиции задачи

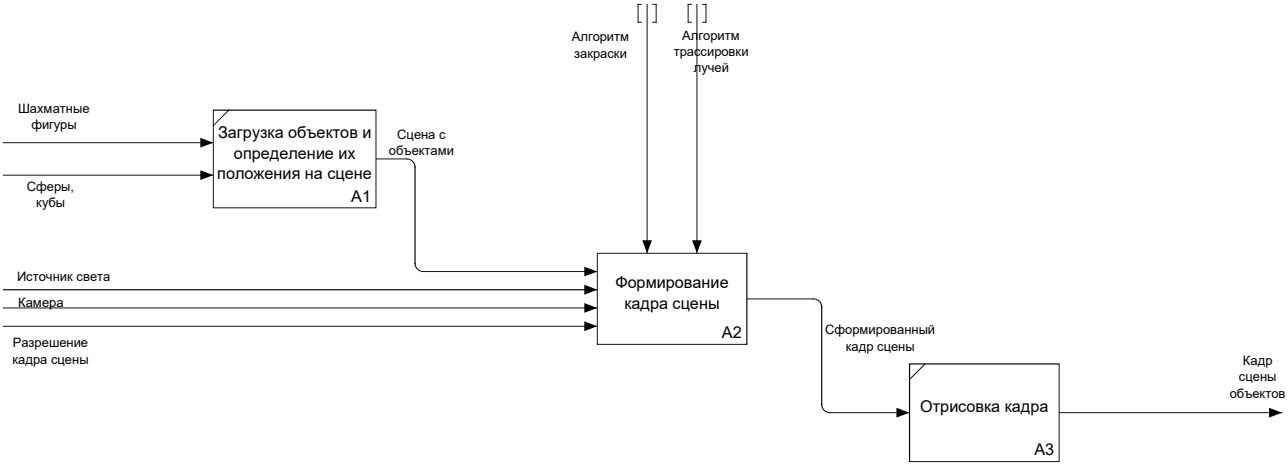

Рисунок 2.2 — Декомпозиция блока A0

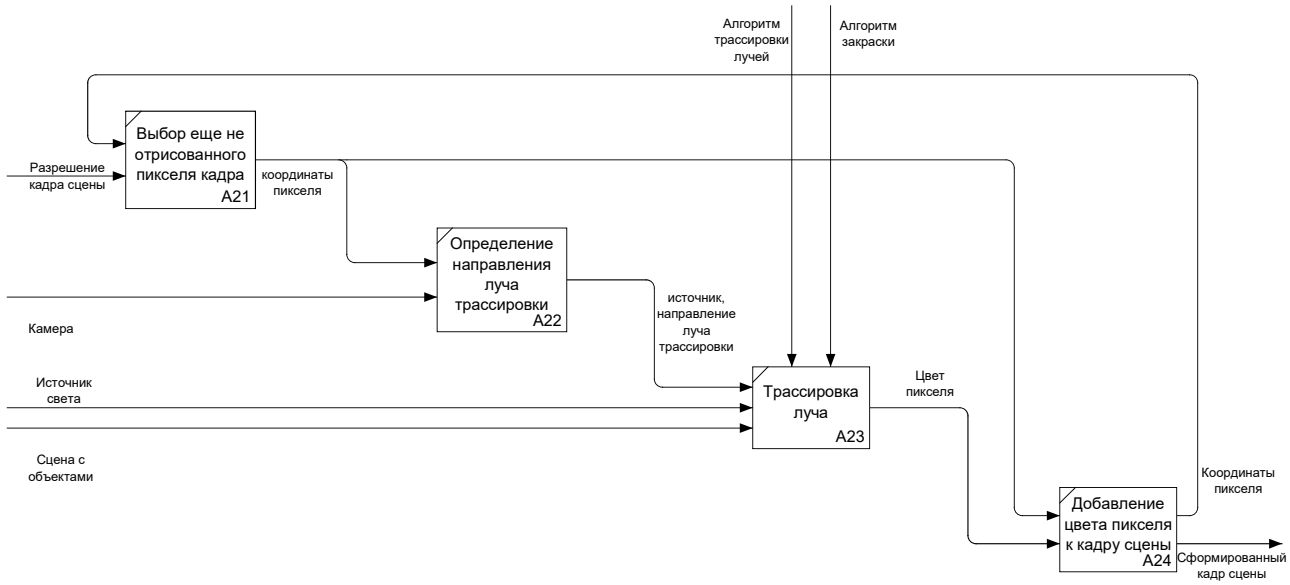

Рисунок 2.3 — Декомпозиция блока A2

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

2.4 Алгоритм трассировки лучей

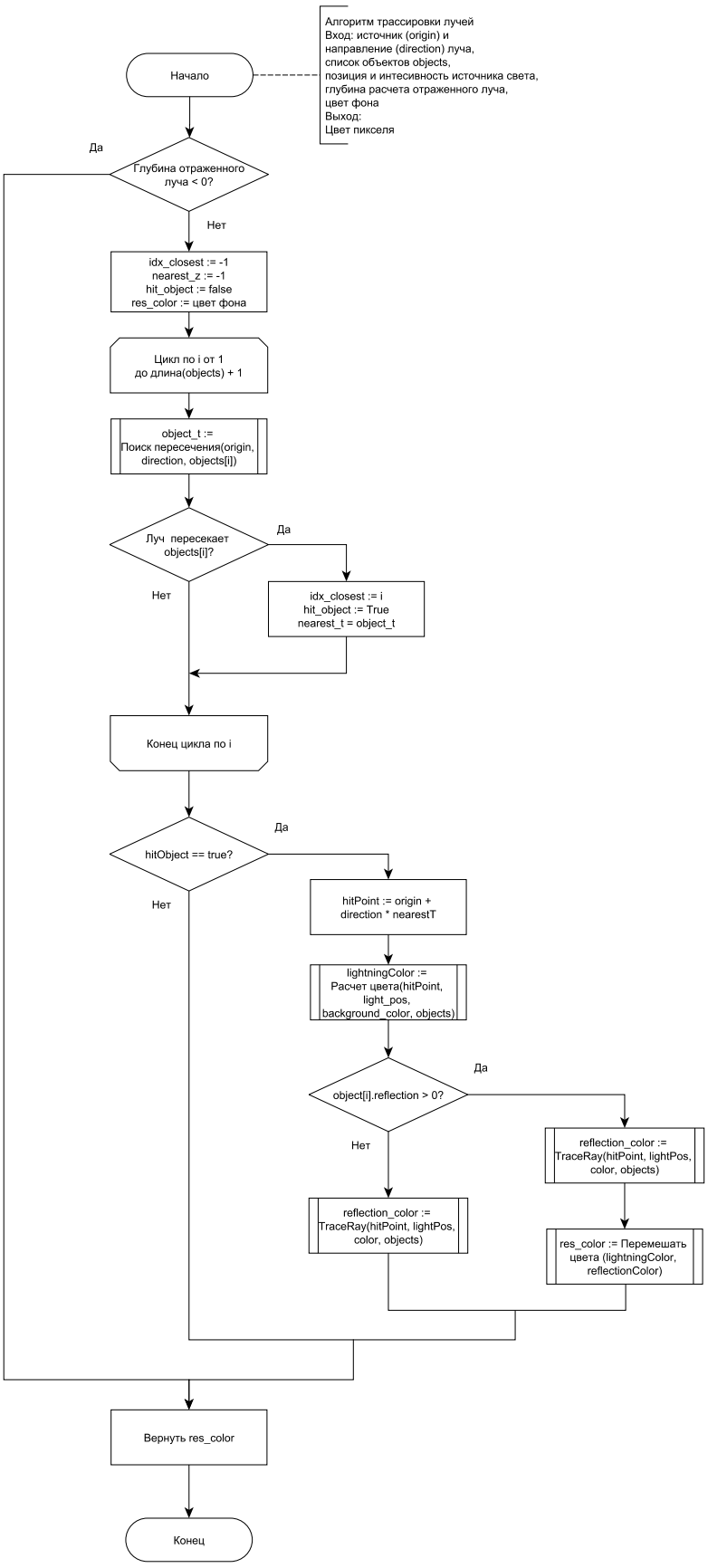

Рисунок 2.4 — Схема алгоритма трассировки лучей

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:

$$p(\vec{t}) = \vec{p_0} + \vec{u} \cdot t, \qquad t \geq 0. \tag{2.1}$$

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

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

Подставляя 2.1 в 2.2 получим

$$|\vec{p_0} + t \cdot \vec{u} - \vec{c}| - r = 0, \tag{2.3}$$

откуда получаем квадратное уравнение

$$A^2 \cdot t + B \cdot t + C = 0, \tag{2.4}$$

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

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

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

На рисунке 2.5 приведена схема алгоритма, реализующая описанные вычисления.

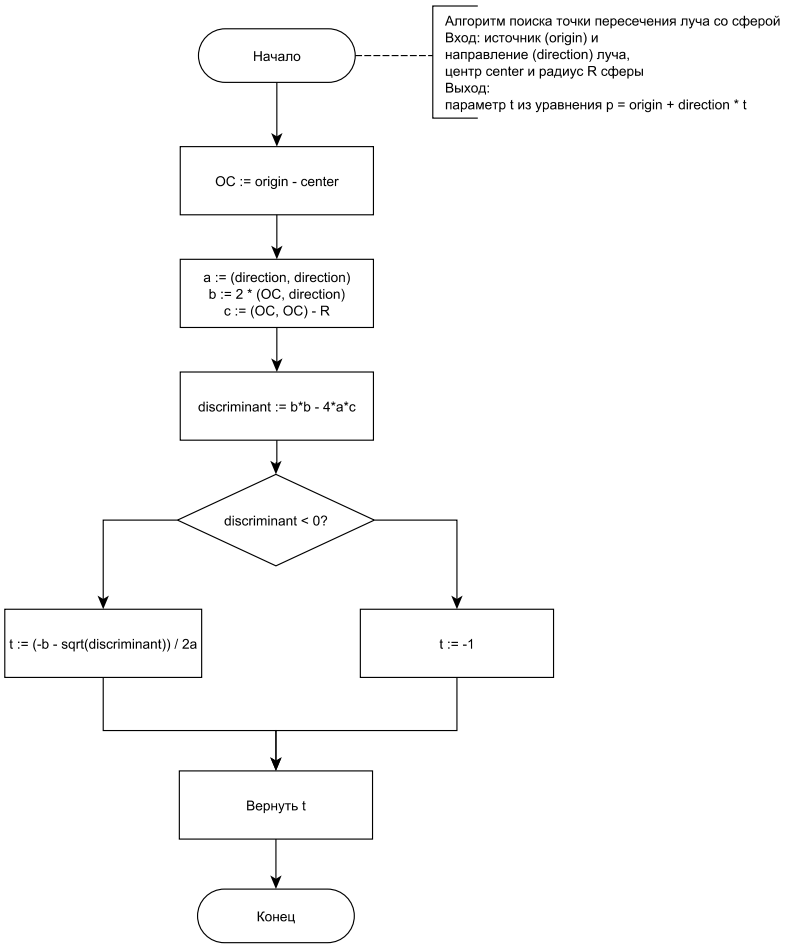

Рисунок 2.5 — Схема алгоритма нахождения точки пересечения луча и сферы

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 [5]. Алгоритм представлен на схемах 2.6-2.7.

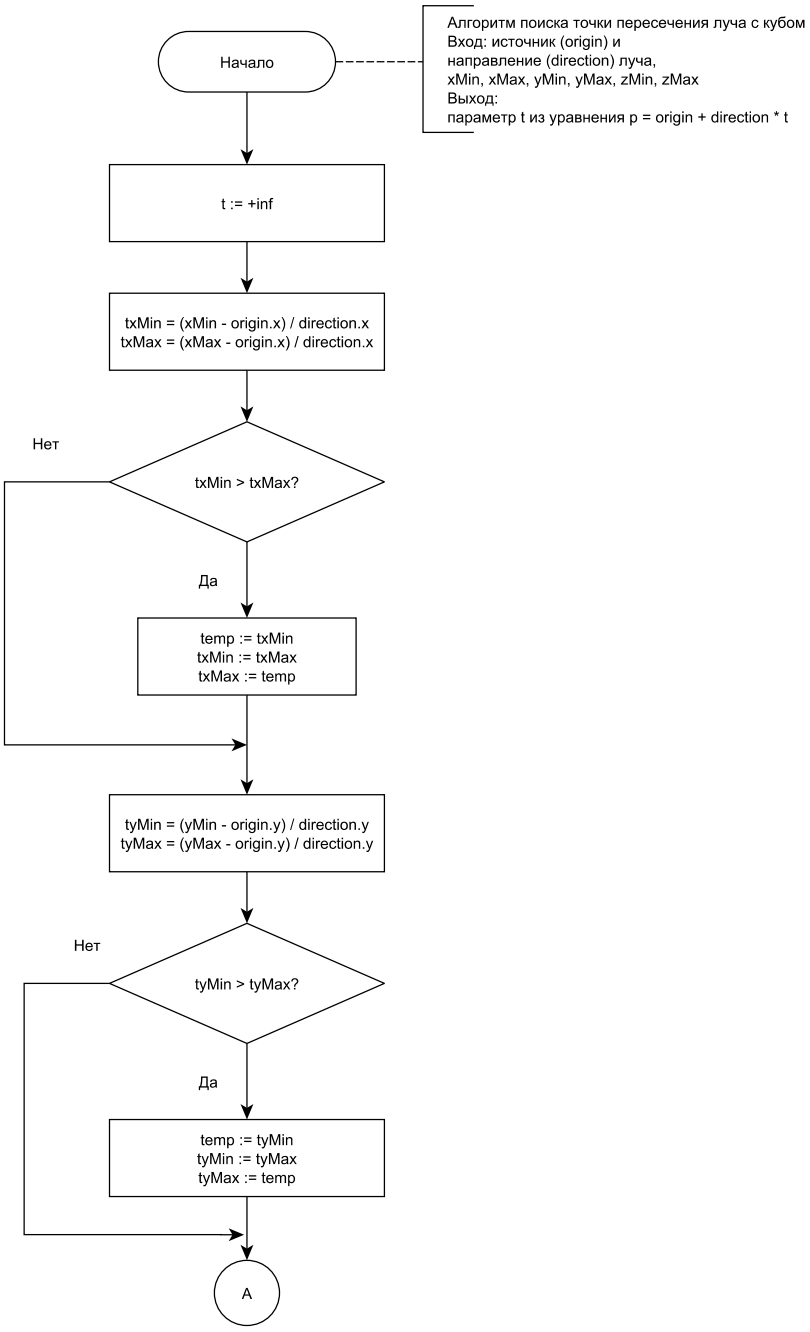

Рисунок 2.6 — Схема алгоритма нахождения точки пересечения луча и сферы

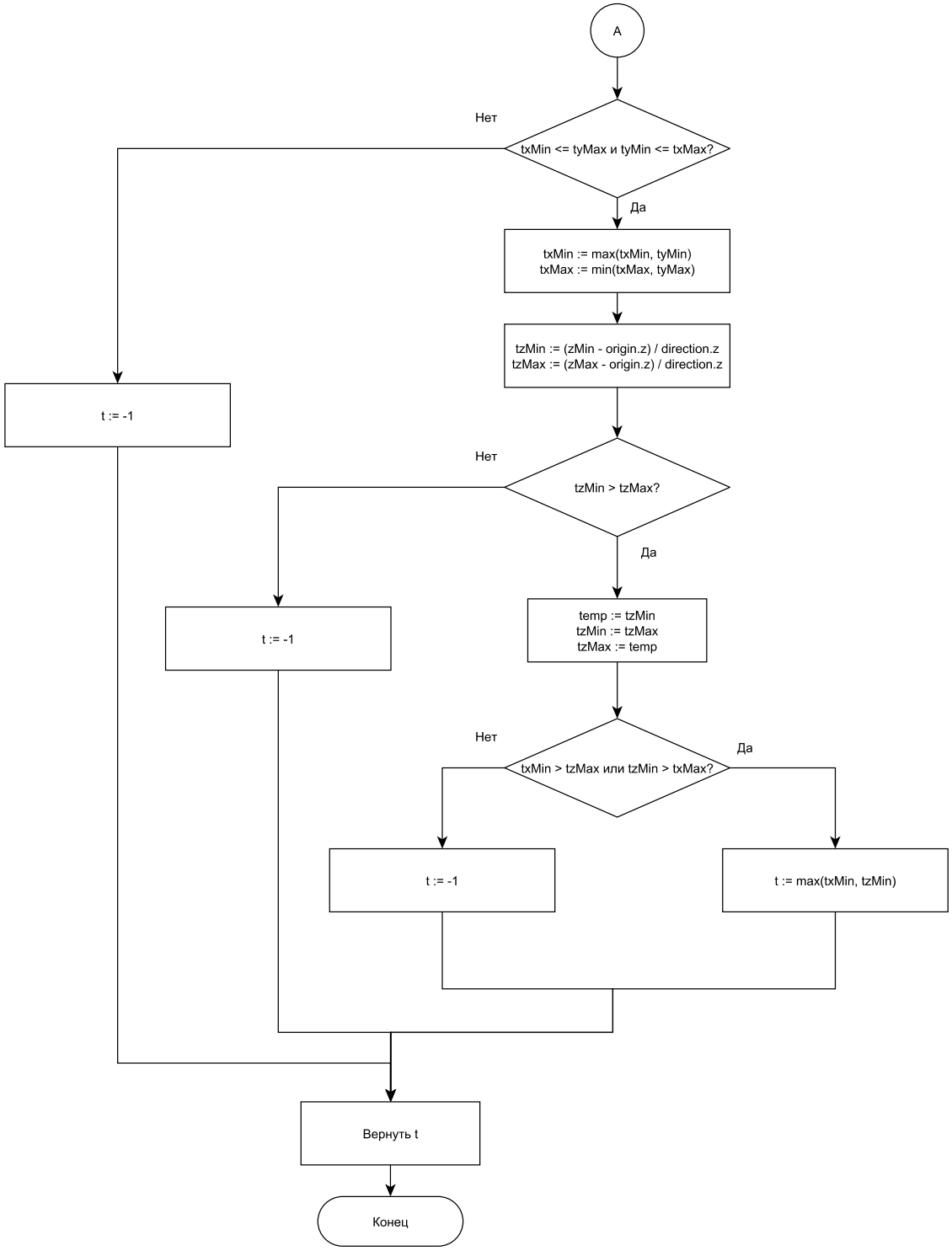

Рисунок 2.7 — Схема алгоритма нахождения точки пересечения луча и сферы

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 — Трумбора, позволяющий

решить задачу без предварительного вычисления уравнения плоскости, содержащей треугольник [6].

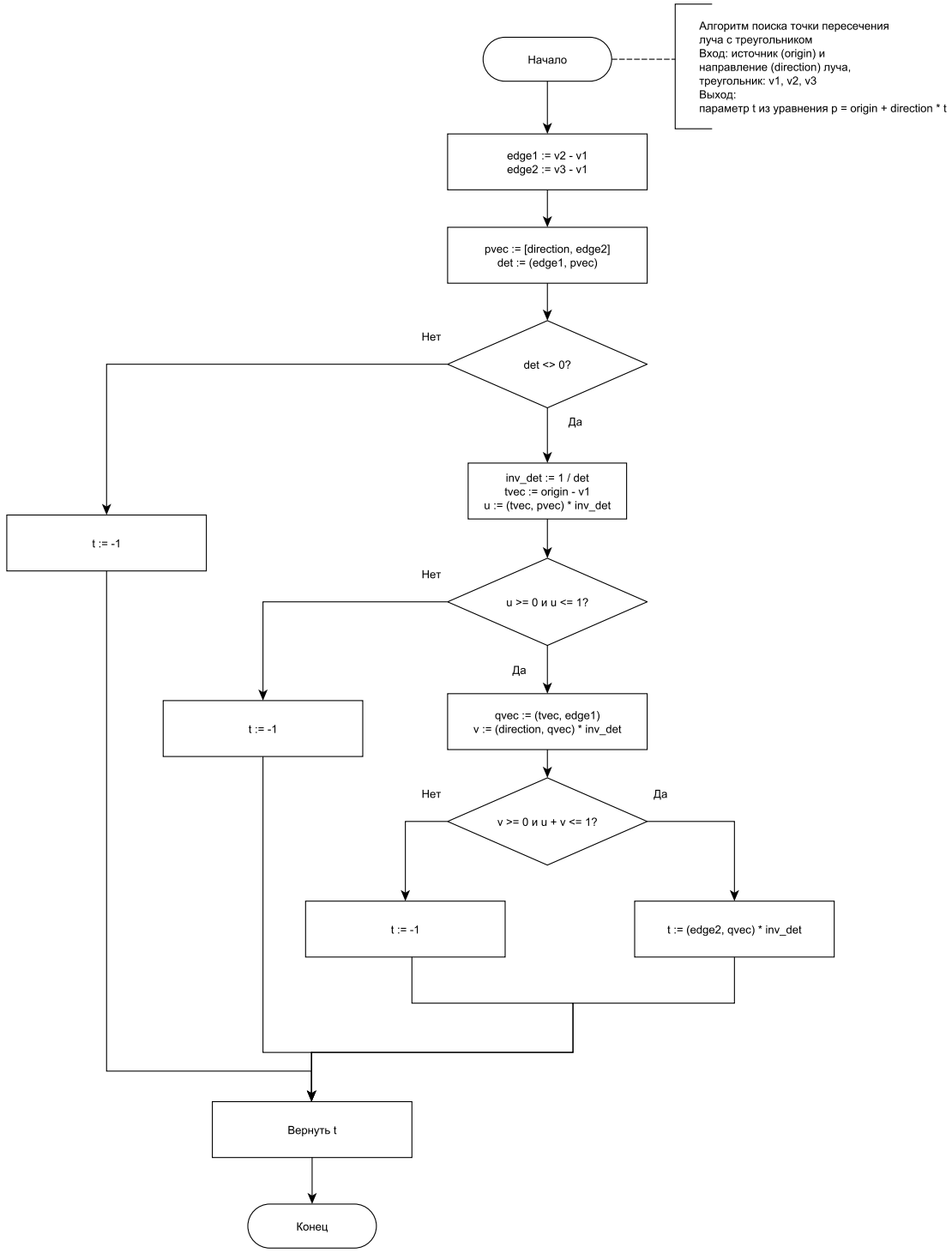

Рисунок 2.8 — Схема алгоритма нахождения точки пересечения луча и треугольника

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.

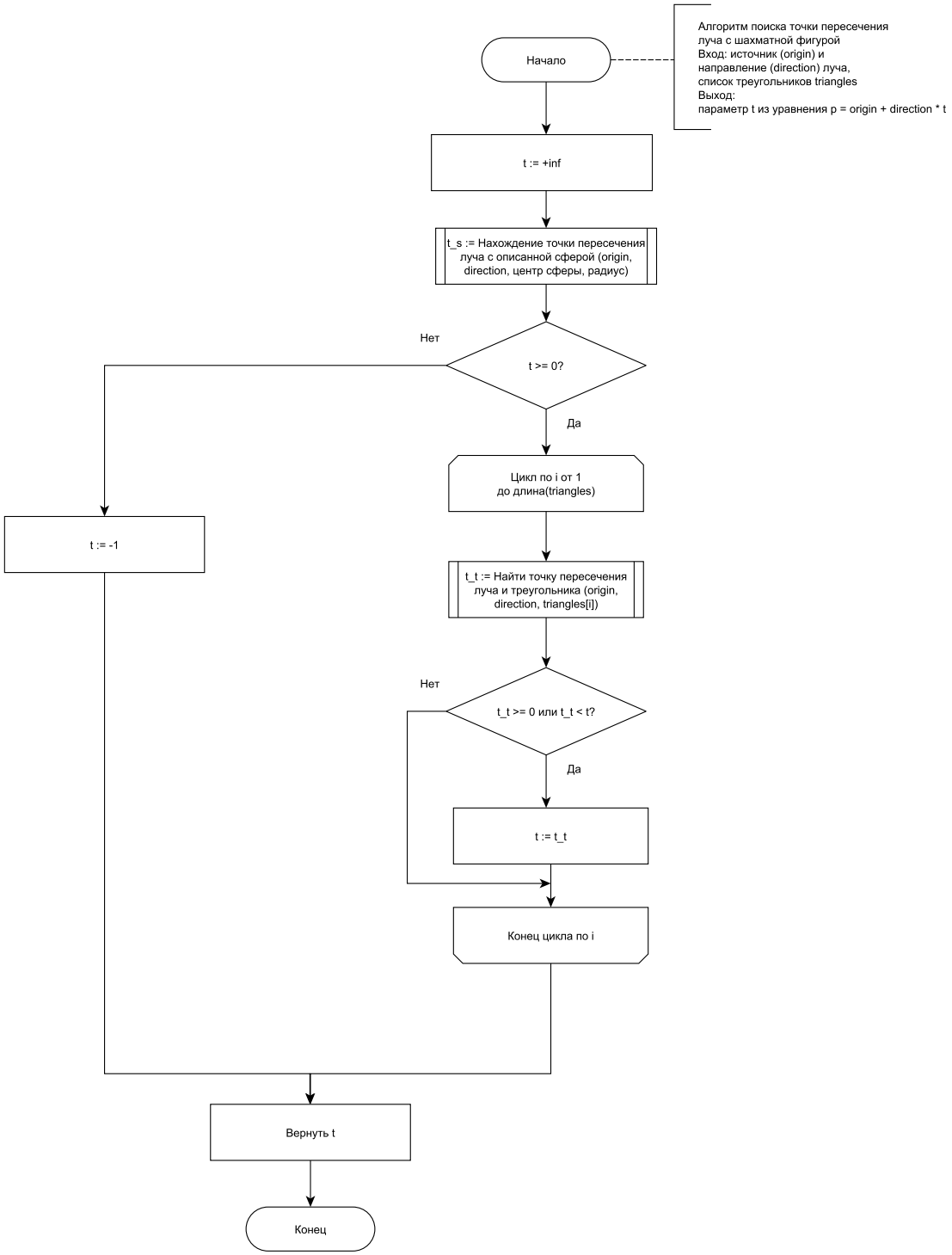

Рисунок 2.9 — Схема алгоритма нахождения точки пересечения луча и шахматной фигуры

2.6 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 [7], 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 [8], предоставляющая достаточный функционал для написания, профилирования и отладки написанной программы, равно как и реализация пользовательского графического интерфейса ПО.

3.2 Структура программы

На рисунках 3.1 - 3.2 представлена диаграмма разработанных классов.

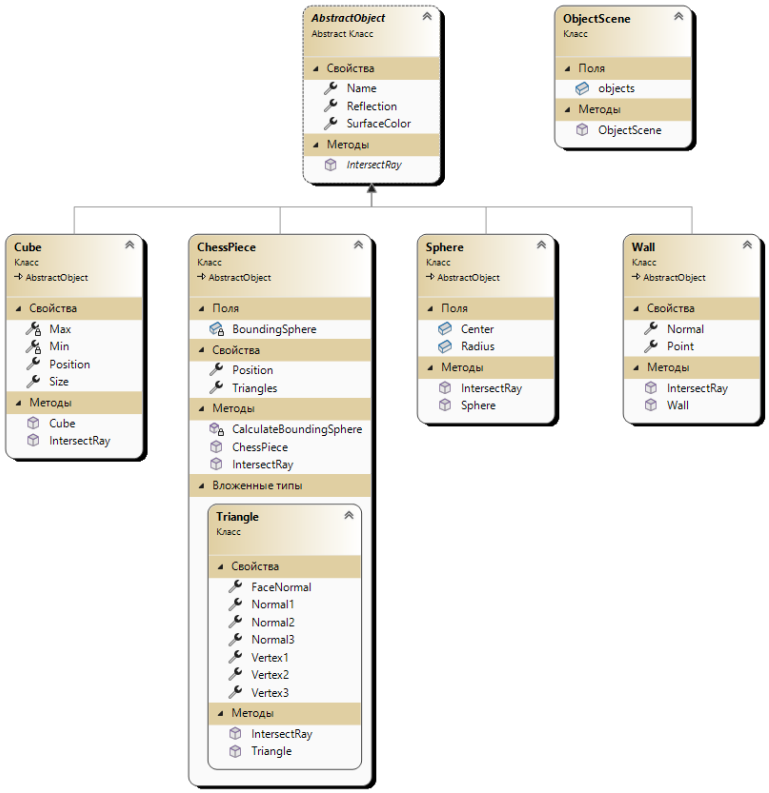

Рисунок 3.1 — Диаграмма классов объектов сцены

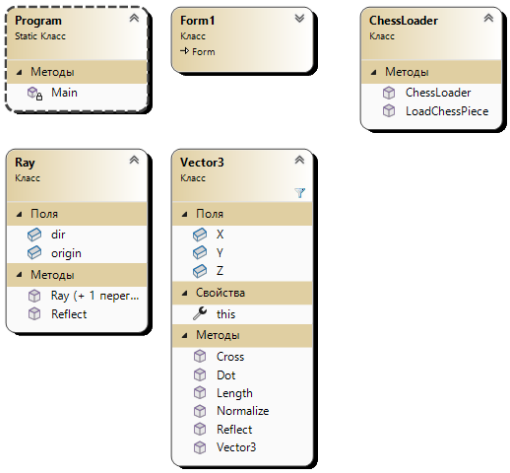

Рисунок 3.2 — Диаграммы вспомогательных классов

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.

3.3 Реализация алгоритмов

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

```
private Color TraceRay(Ray ray, ObjectScene scene, Vector3 lightPos,
    Color backgroundColor, int depth)
{
  if (depth <= 0)
    return backgroundColor;

  double closestDistance = double.MaxValue;
  Vector3 hitNormal = new Vector3(0, 0, 0);
  AbstractObject closestObject = null;

  foreach (var obj in scene.objects)
  {
    if (obj.IntersectRay(ray, out double dist, out Vector3 normal) &&
        dist < closestDistance)
    {
      closestDistance = dist;
      hitNormal = normal;
      closestObject = obj;
    }
  }

  if (closestObject == null)
    return backgroundColor;

  Vector3 hitPoint = ray.origin + ray.dir * closestDistance;
  Color objectColor = ((dynamic)closestObject).SurfaceColor;
  Color lightingColor = CalculateLighting(hitPoint, hitNormal,
     lightPos, objectColor, scene);

  if (closestObject.Reflection > 0)
  {
    Vector3 reflectionDir = ray.dir.Reflect(hitNormal);
    Color reflectionColor = TraceRay(new Ray(hitPoint, reflectionDir)
       , scene, lightPos, backgroundColor, depth - 1);
    lightingColor = MixColors(lightingColor, reflectionColor,
       closestObject.Reflection);
  }
  return lightingColor;
}
```

Листинг 3.1 — Алгоритм трассировки лучей

На листингах 3.2 - 3.6 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;
```

```
        }
      }
      return false;
    }
```

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

```
    public override bool IntersectRay(Ray ray, out double t, out
       Vector3 normal)
    {
      t = double.MaxValue;
      normal = new Vector3(0, 0, 0);

      double tMin = (Min.X - ray.origin.X) / ray.dir.X;
      double tMax = (Max.X - ray.origin.X) / ray.dir.X;

      if (tMin > tMax) (tMin, tMax) = (tMax, tMin);

      double tyMin = (Min.Y - ray.origin.Y) / ray.dir.Y;
      double tyMax = (Max.Y - ray.origin.Y) / ray.dir.Y;

      if (tyMin > tyMax) (tyMin, tyMax) = (tyMax, tyMin);

      if ((tMin > tyMax) || (tyMin > tMax))
      return false;

      if (tyMin > tMin) tMin = tyMin;
      if (tyMax < tMax) tMax = tyMax;

      double tzMin = (Min.Z - ray.origin.Z) / ray.dir.Z;
      double tzMax = (Max.Z - ray.origin.Z) / ray.dir.Z;

      if (tzMin > tzMax) (tzMin, tzMax) = (tzMax, tzMin);

      double epsilon = 1e-8;

      if (tMin > tzMax + epsilon || tzMin > tMax + epsilon)
      return false;

      tMin = Math.Max(tMin, tzMin);
      tMax = Math.Min(tMax, tzMax);

```

```
35        if (tMin < 0)
36        {
37          t = tMax; // If inside the cube, use farther hit
38          if (tMax < 0)
39          return false;
40        }
41        else
42        {
43          t = tMin;
44        }
45
46        Vector3 hitPoint = ray.origin + ray.dir * t;
47
48        if (Math.Abs(hitPoint.X - Min.X) < epsilon) normal = new Vector3
              (-1, 0, 0); // Left face
49        else if (Math.Abs(hitPoint.X - Max.X) < epsilon) normal = new
              Vector3(1, 0, 0);  // Right face
50        else if (Math.Abs(hitPoint.Y - Min.Y) < epsilon) normal = new
              Vector3(0, -1, 0); // Bottom face
51        else if (Math.Abs(hitPoint.Y - Max.Y) < epsilon) normal = new
              Vector3(0, 1, 0);  // Top face
52        else if (Math.Abs(hitPoint.Z - Min.Z) < epsilon) normal = new
              Vector3(0, 0, -1); // Front face
53        else if (Math.Abs(hitPoint.Z - Max.Z) < epsilon) normal = new
              Vector3(0, 0, 1);  // Back face
54
55        return true;
56      }
```

Листинг 3.4 — Алгоритм поиска точки пересечения луча с кубом

```
1       public bool IntersectRay(Ray ray, out double distance, out Vector3
            barycentricCoords)
2       {
3         barycentricCoords = new Vector3(0, 0, 0);
4         distance = 0;
5
6         Vector3 edge1 = Vertex2 - Vertex1;
7         Vector3 edge2 = Vertex3 - Vertex1;
8
9         Vector3 pvec = ray.dir.Cross(edge2);
10        double det = edge1.Dot(pvec);
```

```
11
12        const double epsilon = 1e-8;
13        if (Math.Abs(det) < epsilon)
14        {
15          return false; // Ray is parallel to the triangle
16        }
17
18        double invDet = 1.0 / det;
19
20        Vector3 tvec = ray.origin - Vertex1;
21        double u = tvec.Dot(pvec) * invDet;
22
23        if (u < 0.0 || u > 1.0)
24        {
25          return false;
26        }
27
28        Vector3 qvec = tvec.Cross(edge1);
29        double v = ray.dir.Dot(qvec) * invDet;
30
31        if (v < 0.0 || u + v > 1.0)
32        {
33          return false;
34        }
35
36        distance = edge2.Dot(qvec) * invDet;
37
38        if (distance > epsilon)
39        {
40          barycentricCoords = new Vector3(1 - u - v, u, v);
41          return true;
42        }
43
44        return false;
45      }
```

Листинг 3.5 — 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 3.6 — Алгоритм поиска точки пересечения луча с шахматной фигурой

3.4 Интерфейс программного обеспечения

На рисунке 3.3 приведена демонстрация интерфейса программы при ее запуске.

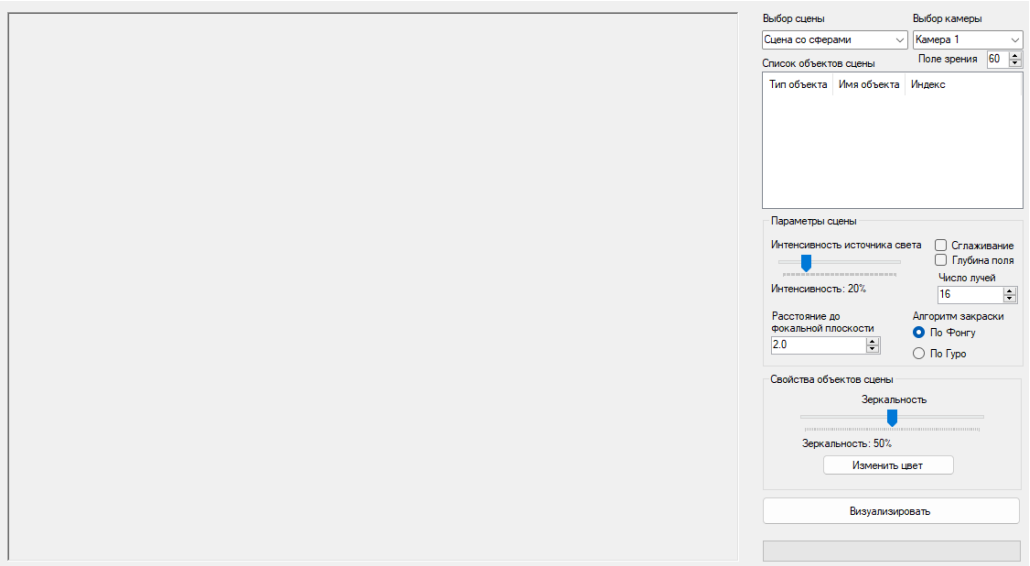

Рисунок 3.3 — Пользовательский интерфейс ПО

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 3.4.

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) и глубину поля (Depth Of Field), выбрать число лучей для двух описанных эффектов, изменить расстояние от камеры до фокальной плоскости (плоскость всегда перпендикулярна 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 3.5.

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).

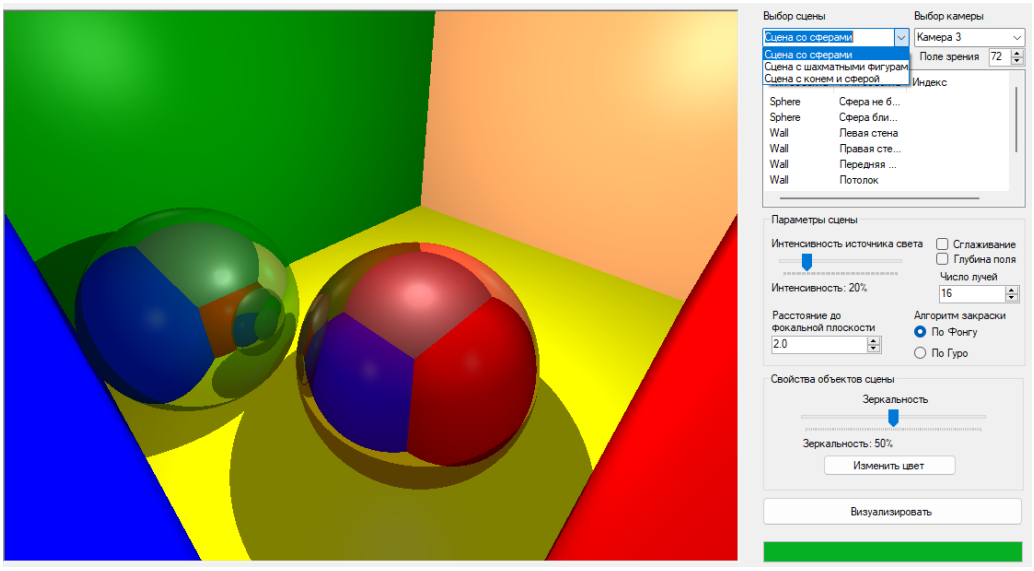

Рисунок 3.4 — Пример выбора сцены и камеры

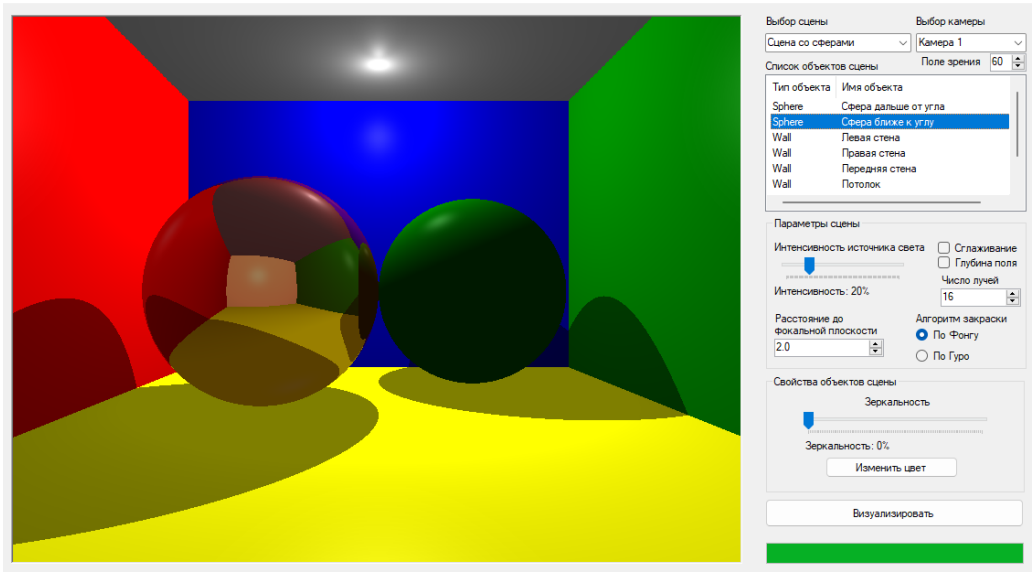

Рисунок 3.5 — Пример выбора объекта на сцене и изменения его зеркальности

3.5 Вывод

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

4 Исследовательская часть

4.1 Вывод

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Роджерс Д., Адамс Дж. Математические основы машинной графики. Москва: Мир, 2001. 500с.;

2. Алгоритм Робертса удаления невидимых граней [Электронный ресурс]. URL: https://cgraph.ru/node/495 (дата обращения 06.10.24);

3. Закрашивание. Рендеринг полигональных моделей [Электронный ресурс]. URL: https://intuit.ru/studies/professional_skill_improvements/1283/courses/70/lecture/21/08?page=2 (дата обращения 06.10.24);

4. Построение теней [Электронный ресурс]. URL: https://stratum.ac.ru/education/textbooks/kgrafic/additional/addit28.html (дата обращения 06.10.24);

5. Glassner, Andrew. An Introduction to Ray Tracing. New York: Academic Press, 1991. 359p.;

6. 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;

7. Заметки о выпуске C 7.3 [Электронный ресурс]. URL: https://docs.microsoft.com/en-us/visualstudio/releasenotes/vs2017-relnotes-v15.7#csharp (дата обращения 4.12.2024);

8. Microsoft Visual Studio 2022 [Электронный ресурс]. URL: https://visualstudio.microsoft.com/ru/vs/