



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

K KУРСОВОЙ РАБОТЕ

на тему:

*«Моделирование геометрических тел простой формы с
изменяемым коэффициентом отражения»*

Студент ИУ7-54Б
(Группа)

(Подпись, дата)

Разин А.
(И. О. Фамилия)

Руководитель курсовой работы

(Подпись, дата)

Кузнецова О. В.
(И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 Аналитическая часть	5
1.1 Модель визуализации примитивов	5
1.1.1 Формализация объектов сцены	5
1.1.2 Определение множества алгоритмов визуализации	6
1.1.3 Визуализация примитивов	8
1.2 Анализ моделей отражения	10
1.2.1 Модель Ламберта	11
1.2.2 Модель Фонга	12
1.2.3 Выбор модели отражения	13
1.3 Анализ алгоритмов визуализации	14
1.3.1 Алгоритм трассировки лучей	14
1.3.2 Трассировка лучей в пространстве изображения	16
1.3.3 Трассировка пути	18
1.3.4 Сравнение алгоритмов	21
2 Конструкторская часть	24
2.1 Требования к программному обеспечению	24
2.2 Выбор типов и структур данных	24
2.3 Общий алгоритм построения изображения	25
2.4 Алгоритм обратной трассировки лучей	26
3 Технологическая часть	29
3.1 Средства Реализации	29
3.2 Реализация алгоритмов	29
3.3 Интерфейс программного обеспечения	30
3.4 Тестирование	32
3.4.1 Реалистичность отображаемых примитивов	33
3.4.2 Смена блика	33
3.4.3 «Рекурсивное» отражение примитивов	34
3.4.4 Отражение нескольких примитивов	34

4 Исследовательская часть	36
4.1 Технические характеристики	36
4.2 Временные характеристики	36
ЗАКЛЮЧЕНИЕ	40
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	43
ПРИЛОЖЕНИЕ А	44

ВВЕДЕНИЕ

Компьютерная графика — совокупность методов и способов преобразования информации в графическое представление при помощи ЭВМ [1]. Конечным продуктом графики является изображение, представляющее собой визуальное представление данных, поданных для отображения [2].

При необходимости получения фотoreалистичных изображений, необходимо принимать во внимание условия освещения и оптические свойства материалов, задействованных в сцене. При расчете освещения данных изображений, необходимо учитывать отраженную и преломленную составляющую света, для реалистичной визуализации наиболее важной составляющей является отраженная часть света [3; 4]. Таким образом, для получения реалистичного изображения, необходимо произвести расчет интенсивности отраженной составляющей света.

Целью данной работы является разработка и создание программного обеспечения, моделирующего отражения от геометрических тел.

Для достижения поставленной цели требуется решить следующие задачи:

1. формализовать представление объектов сцены и описать их;
2. проанализировать алгоритмы построения реалистичных изображений и теней;
3. выбрать наилучшие алгоритмы для достижения цели из рассмотренных;
4. проанализировать полученную модель взаимодействия света с объектами;
5. выбрать программные средства для реализации модели;
6. реализовать полученную модель и создать интерфейс;
7. провести замеры времени построения кадра от количества и типа примитивов на сцене.

1 Аналитическая часть

В данной части работы будет рассмотрена модель, по которой будет производиться визуализация, а также будут определены модель отражения и алгоритм визуализации примитивов.

1.1 Модель визуализации примитивов

В данной части работы происходит формализация объектов сцены и описание метода их визуализации.

1.1.1 Формализация объектов сцены

На визуализируемой сцене могут находиться следующие объекты.

1. Точечный источник света

Данный источник света излучает свет во всех направлениях, интенсивность света убывает при удалении от источника. Источник характеризуется положением в пространстве и интенсивностью. При расчете отражений будет использоваться интенсивность источника, для расчета интенсивности пикселей. Цвет свечения будет описываться через значения RGB.

2. Сфера

Сфера описывается радиусом и координатами ее центра.

3. Куб

Для описания куба необходимо несколько параметров.

- Координаты центра.
- Размеры куба по каждой из осей.
- Угол поворота по каждой из осей.

4. Конус

Конус описывается следующими параметрами.

- Половинный угол при вершине осевого сечения конуса.

- Высота конуса.
- Координаты вершины конуса.
- Вектор высоты конуса.

5. Цилиндр

Цилиндр описывается несколькими параметрами.

- Координаты центра первого основания цилиндра.
- Координаты центра второго основания цилиндра.
- Радиус цилиндра.

6. Камера

Камера описывается несколькими параметрами.

- Координатами своего положения.
- Системой координат камеры (описывается 3 взаимноперпендикулярными векторами).

Каждый из примитивов также должен описываться своим цветом в формате RGB, а также коэффициентами рассеянного, диффузного, зеркального отражения.

1.1.2 Определение множества алгоритмов визуализации

Существует простая (локальная) модель освещения, то есть модель освещения в которой учитывается свет, попадающий в рассматриваемую точку только от источника света. Также выделяют глобальную модель освещения, в которой также учитывается интенсивность света, отраженного от других поверхностей. Для получения реалистичных отражений стоит использовать глобальную модель освещения [2].

Приведем пример отражения лучей:

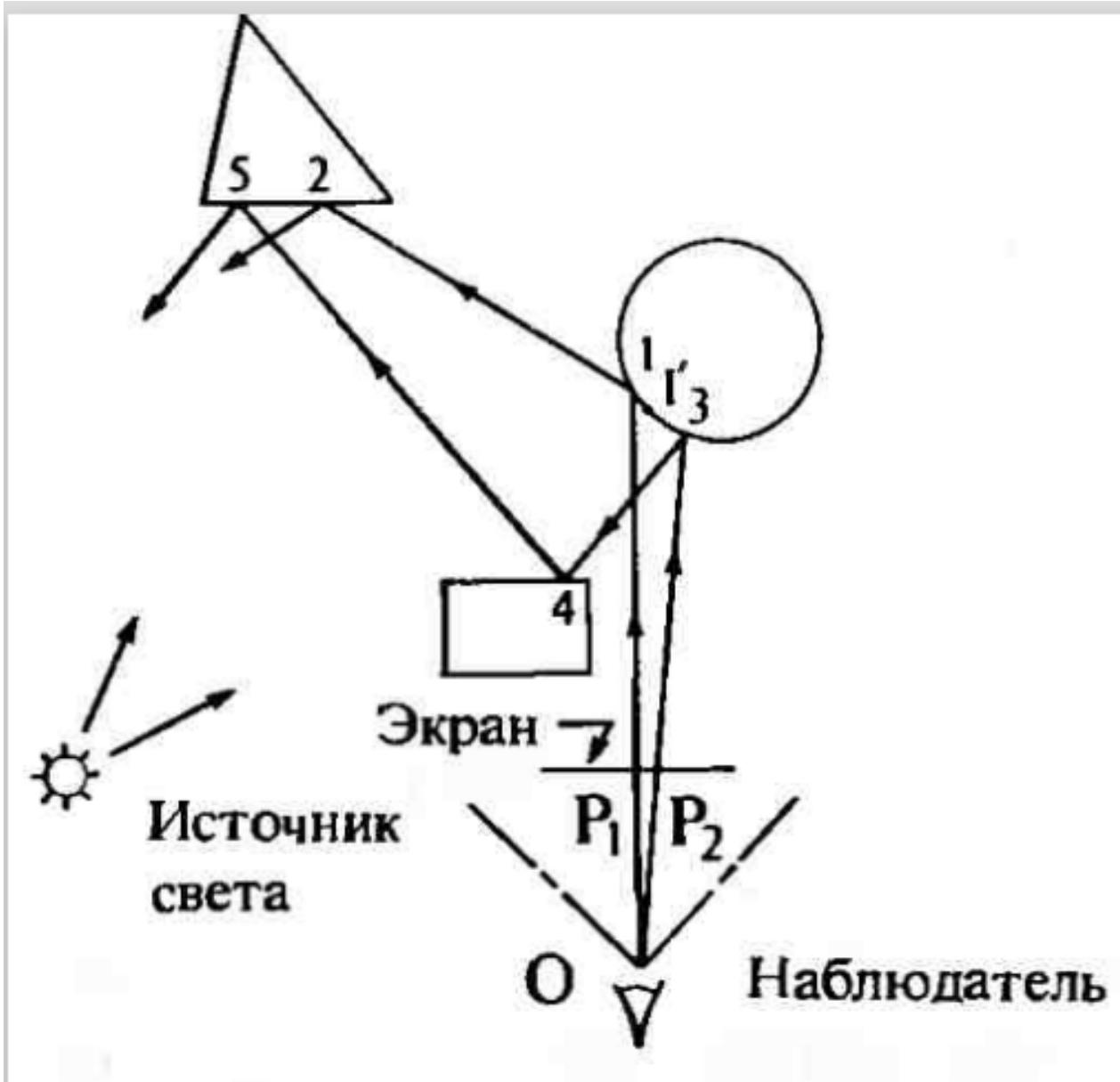


Рисунок 1.1 – Пример трассировки луча

На рисунке 1.1 призма, загороженная от наблюдателя параллелепипедом, становится видимой из-за отражения в сфере. Точка 5 видима, так как отражается от обратной стороны параллелепипеда в точке 4 к точке 3 на сфере, а затем к наблюдателю. Таким образом, при создании глобальной модели освещения, алгоритмы, основанные на удалении невидимых поверхностей не будут давать изображения необходимого качества. Глобальная модель освещения является частью алгоритмов выделения видимых поверхностей путем трассировки лучей, то есть для визуализации отражений необходимо использовать трассировку лучей [2].

1.1.3 Визуализация примитивов

Для визуализации примитивов методом трассировки лучей необходимо определить точку пересечения луча с каждым примитивом сцены или выявить ее отсутствие.

Необходимо ввести уравнение самого луча (1.1).

$$P(t) = \vec{E} + t\vec{D}, t \geq 0 \quad (1.1)$$

Также уравнение 1.1 имеет запись

$$\begin{aligned} x(t) &= x_E + tx_D, \\ y(t) &= y_E + ty_D, \\ z(t) &= z_E + tz_D. \end{aligned} \quad (1.2)$$

Таким образом луч определяется: точкой обзора — $\vec{E} = (x_E, y_E, z_E)$ и вектором направления — $\vec{D} = (x_D, y_D, z_D)$. Значение t определяет конкретную точку на луче: в случае если $t \geq 0$, точка на луче находится после точки обзора, иначе — за. Таким образом для поиска ближайшей точки пересечения, необходимо найти наименьшее неотрицательное значение t [2; 5].

Уравнение сферы Сфера с единичным радиусом может быть задана следующим образом:

$$\vec{P} \cdot \vec{P} = 1. \quad (1.3)$$

Для получения условия пересечений достаточно подставить уравнение луча из (1.1) в (1.3) и решить полученное уравнение относительно t .

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (1.4)$$

После проведенных преобразований будет получена формула (1.4), где:

1. $a = \vec{D} \cdot \vec{D}$;
2. $b = 2\vec{E} \cdot \vec{D}$;
3. $c = \vec{E} \cdot \vec{E} - 1$.

В случае если вещественные решения уравнения (1.4) отсутствуют, то пе-

пересечения луча также отсутствуют, если только одно решение - существует одно пересечение и т. д. Если значение отрицательное то точка пересечения находится за точкой наблюдения и не рассматривается [5].

Уравнение цилиндра Определим $extr_a$, как первое основание цилиндра, $extr_b$ — как второе, r — радиус цилиндра. С помощью полученных данных вычисляем ось координат цилиндра $V = extr_b - extr_a$. Для того чтобы точка P принадлежала цилинду необходимо, чтобы для нее выполнялось выражение (1.5).

$$\begin{cases} A = \vec{C} + \vec{V} \cdot m, \\ (\vec{P} - \vec{A}) \cdot \vec{V} = 0, \\ |P - A| = r. \end{cases} \quad (1.5)$$

В системе (1.5), m — определяет ближайшую точку на оси цилиндра до точки пересечения с лучом. После решения системы уравнений (1.5) и подстановки выражения 1.1, будет получено значение m (см. 1.6).

$$m = \vec{D} \cdot \vec{V} \cdot t + (\vec{O} - \vec{C}) \cdot \vec{V}. \quad (1.6)$$

После получения значения m , возможна запись уравнения (1.7).

$$|\vec{P} - \vec{C} - \vec{V} \cdot m| = r^2 \quad (1.7)$$

Введем $\vec{X} = \vec{E} - \vec{C}$. При решении данного квадратного уравнения относительно t будут получены следующие коэффициенты:

$$\begin{cases} a = \vec{D} \cdot \vec{D} - (\vec{D} \cdot \vec{D}), \\ \frac{b}{2} = \vec{D} \cdot (\vec{X}) - ((\vec{D} \cdot \vec{V}) \cdot (\vec{X} \cdot \vec{V})), \\ c = \vec{X} \cdot \vec{X} - (\vec{X} \cdot \vec{V})^2 - r^2. \end{cases} \quad (1.8)$$

После получения коэффициентов из системы (1.8) и их подстановки в (1.4), будут получены значения t [6].

Уравнение конуса

Определим координаты конуса как C , вектор высоты конуса как вектор V , θ как половинный угол при вершине осевого сечения конуса. Таким образом

каждая точка X на конусе, может определяться с помощью формулы (1.9).

$$(\vec{X} - \vec{C}) \cdot \vec{V} = \|\vec{X} - \vec{C}\| \cos \theta \quad (1.9)$$

В случае, если точка на луче принадлежит конусу будет получена система (1.10).

$$\begin{cases} \vec{P} = \vec{E} + t\vec{D} \\ \frac{((\vec{P}-\vec{C}) \cdot \vec{V})^2}{(\vec{P}-\vec{C}) \cdot (\vec{P}-\vec{C})} = \cos^2 \theta \end{cases} \quad (1.10)$$

После подстановки значения P из системы (1.10), в нижнее уравнение системы и упрощения полученного уравнения будет получено квадратное уравнение относительно t .

$$\begin{cases} a = (\vec{D} \cdot \vec{V})^2 - \cos^2 \theta \\ b = 2 \cdot ((\vec{D} \cdot \vec{V}) \cdot ((\vec{E} - \vec{C}) \cdot \vec{V}) - \vec{D} \cdot (\vec{E} - \vec{C}) \cos \theta^2) \\ c = ((\vec{E} - \vec{C}) \cdot \vec{V})^2 - \vec{C} \vec{E} \cdot \vec{C} \vec{E} \cos^2 \theta \end{cases} \quad (1.11)$$

После подстановки данных значений в выражение (1.4), будет получены значения t [7].

1.2 Анализ моделей отражения

Свет отраженный от объекта может быть диффузным и зеркальным. Диффузное отражение происходит, когда свет поглощается поверхностью, а затем вновь испускается, отражение равномерно рассеивается по всем направлениям и положение наблюдателя не имеет значения. Зеркальное отражение происходит от внешней поверхности объекта, оно является направленным и зависит от положения наблюдателя. Так как отражение происходит от внешней части объекта, то отраженный свет сохраняет свойства падающего, например в случае если белый свет отражается от красного тела, отраженный свет также будет нести в себе часть красного цвета [2].

Для расчета интенсивности света данных отражений существует несколько моделей [2]:

1. модель Ламберта;
2. модель Фонга.

1.2.1 Модель Ламберта

В данной модели рассматривается диффузная составляющая отражения.

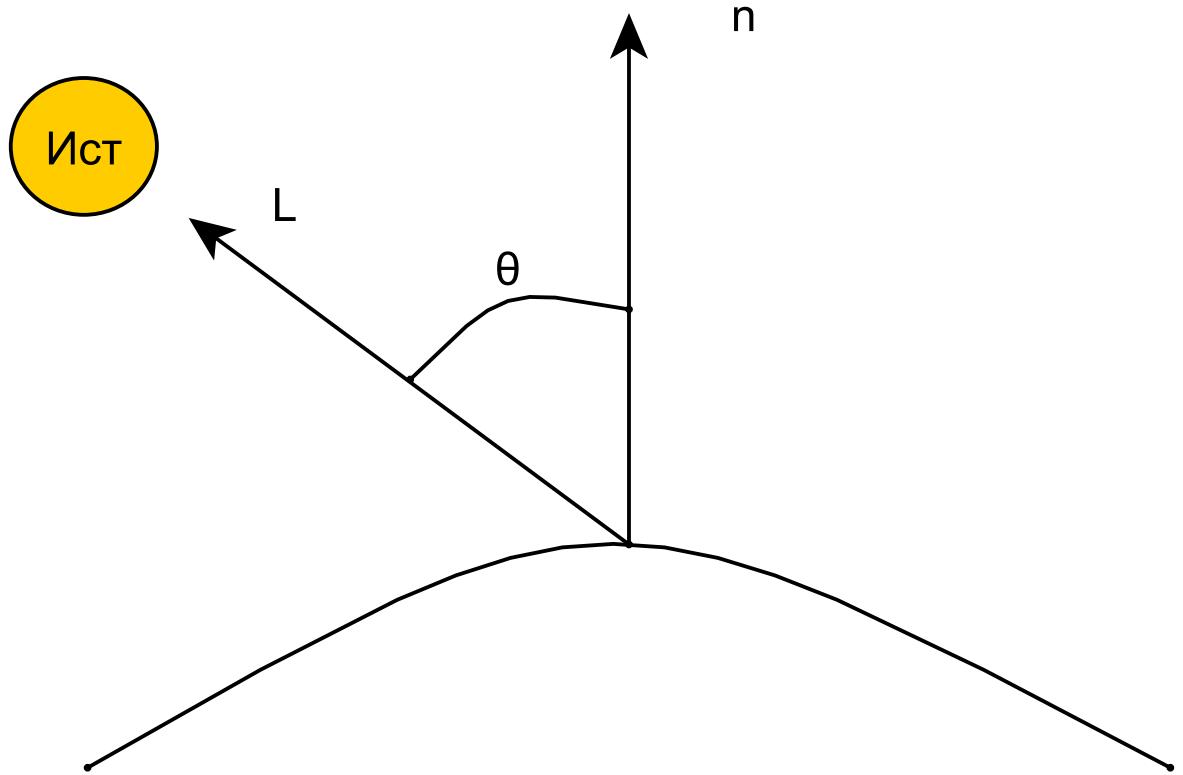


Рисунок 1.2 – Модель Ламберта

Считается, что интенсивность отраженного света пропорциональна косинусу угла между направлением света и нормалью к поверхности:

$$I = k_a I_a + I_l k_l \cos \theta \quad 0 \leq \theta \leq \pi/2. \quad (1.12)$$

В формуле (1.12):

1. k_a, k_d - коэффициенты рассеянного, диффузного отражения соответственно;
2. I_a, I_l - интенсивность рассеянного и диффузного отражения;
3. θ - угол между нормалью к поверхности и направлением света.

Заметим что значения приведенных коэффициентов лежат на отрезке от 0 до 1 [2].

Однако интенсивность света должна убывать с увеличением расстояния от источника до объекта, эмпирически было выведено следующее соотношение:

$$I = k_a I_a + \frac{I_l k_l \cos \theta}{d + K}. \quad (1.13)$$

В данном случае добавлены d, K , в случае если точка наблюдения на бесконечности, то d - определяется положением объекта, ближайшего к точке наблюдения, то есть он будет освещаться с максимальной интенсивностью источника, а дальние объекты - с уменьшенной. Таким образом K - произвольная постоянная [2].

1.2.2 Модель Фонга

Данная модель также учитывает зеркальную составляющую отражения

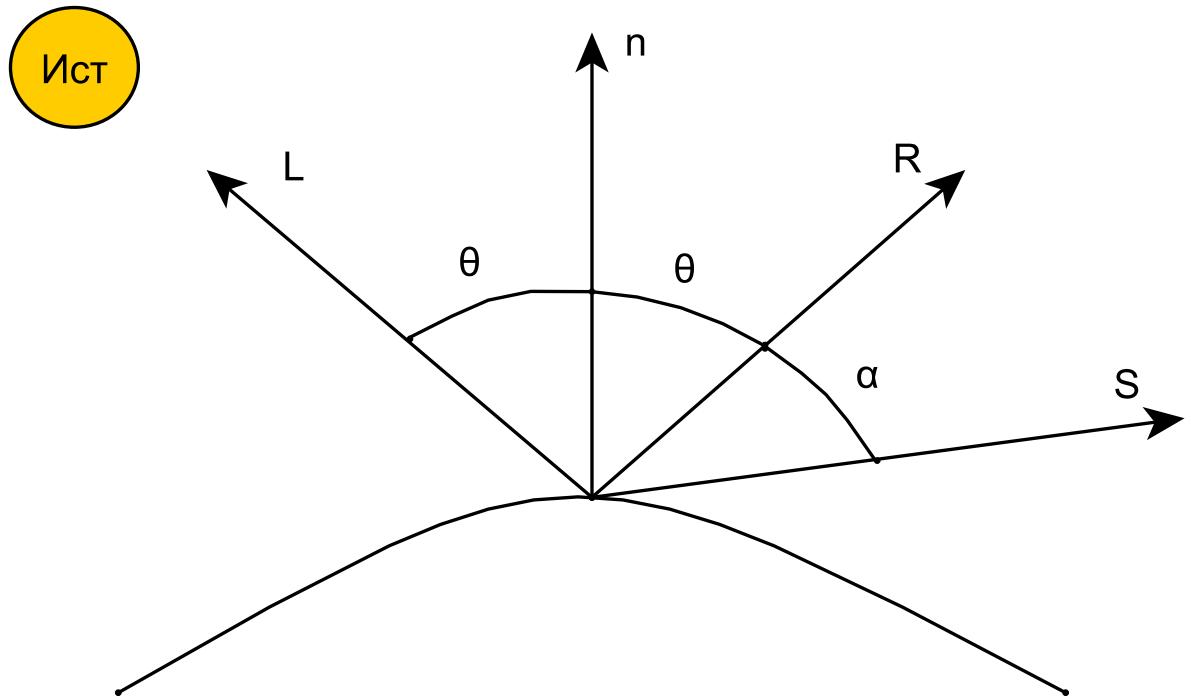


Рисунок 1.3 – Модель Фонга

Зеркальная составляющая отражения имеет следующий вид:

$$I_s = I_l \omega(i, \lambda) \cos^n \alpha. \quad (1.14)$$

В формуле (1.14) символы соответственно означают:

1. $\omega(i, \lambda)$ - кривая отражения, показывающая отношение зеркально отраженного света к падающему, как функцию угла падения i и длины волны λ ;
2. α - угол между отраженным лучом и вектором, проведенным из точки падения луча в точку наблюдения;
3. n - степень, аппроксимирующая пространственное распределение отраженного света;
4. I_l - интенсивность падающего луча.

Функция $\omega(i, \lambda)$ сложна, так что ее заменяют константой k_s , получаемой экспериментально [2].

Таким образом формула принимает следующий вид:

$$I = k_a I_a + k_d I_l (\hat{n} \cdot \hat{L}) + k_s I_l (\hat{S} \cdot \hat{R})^n. \quad (1.15)$$

В данном случае косинусы вычисляются с помощью скалярного произведения нормированных векторов:

1. \hat{n} - вектор нормали поверхности в точке падения;
2. \hat{L} - вектор падающего луча;
3. \hat{S} - вектор наблюдения;
4. \hat{R} - вектор отражения.

Символ $\hat{\cdot}$ означает, что данный вектор нормированный [2].

1.2.3 Выбор модели отражения

Так как для выполнения поставленной цели необходимо рассматривать зеркальное отражение, наиболее подходящей является модель отражения Фонга.

1.3 Анализ алгоритмов визуализации

При построении реалистичного изображения необходимо с полироваными поверхностями необходимо визуализировать отражения света от тел. Существуют множество подходов для создания реалистичных изображений:

1. трассировка световых лучей (англ. Ray tracing);
2. трассировка пути (англ. Path tracing);
3. трассировка лучей в пространстве изображения (англ. Screen reflections).

1.3.1 Алгоритм трассировки лучей

В реальной жизни объекты являются видимыми, в случае если они отражают свет от источника, после чего данные лучи света попадают в человеческий глаз. Аналогичная идея заложена в данном способе создания изображения — необходимо отследить движение лучей света. Отслеживать путь всех лучей света не стоит, так как это неэффективно (малое число лучей попадут в наблюдателя), при построении изображения внимание следует уделять объектам видимыми со стороны наблюдателя. В таком случае можно отслеживать лучи света, исходящие из точки наблюдения, т. е. производить трассировку лучей в обратном направлении. В данном случае лучи стоит проводить через центры пикселей изображения, считается, что наблюдатель находится на бесконечности, из-за чего все лучи параллельны оси OZ [2; 8].

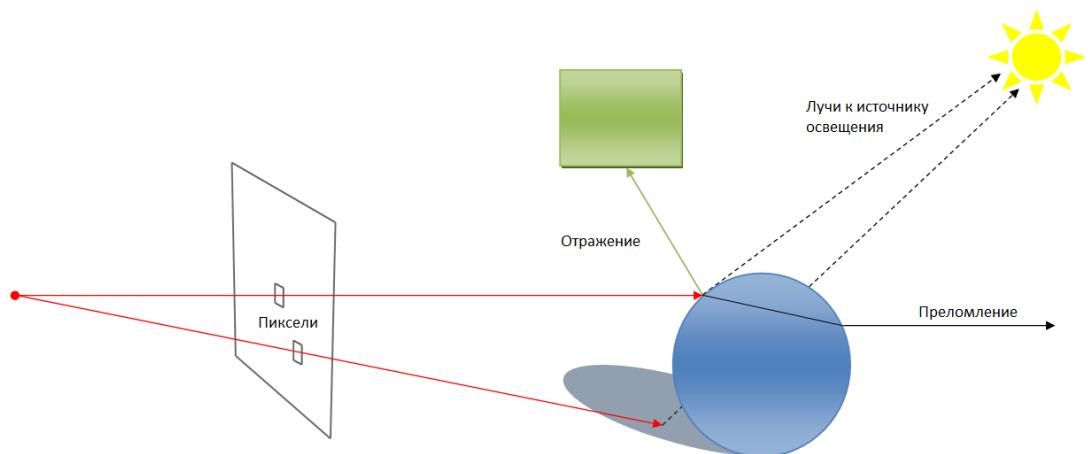


Рисунок 1.4 – Пример трассировки луча

Первые работы принадлежат Уиттеду и Кэю. Алгоритм Уиттеда более общий и часто используется. Уиттед пользуется моделью, в которой диффузная и зеркальная составляющие отражения рассчитываются подобно локальной модели (пример приведен на рисунке 1.2) [2].



Рисунок 1.5 – Расчет зеркального отражения луча в алгоритме Уиттеда

На рисунке 1.5 луч \mathbf{V} падает на поверхность в точку \mathbf{Q} , после чего отражается в направлении \mathbf{r} и преломляется в направлении \mathbf{p} . В данном случае:

1. I_t - интенсивность света, проходящего по преломленному лучу \mathbf{p} ;
2. η - показатели преломления сред (влияют на направление преломленного луча);
3. \hat{S}, \hat{R} - полученные вектора наблюдения и отражения;
4. \hat{L}_j - Вектор к источнику света j .

Тогда наблюдаемая интенсивность \mathbf{I} выражается формулой:

$$I = k_a I_a + k_d \sum_j I_{l_j} (\hat{n} \cdot \hat{L}_j) + k_s \sum_j I_{l_j} (\hat{S} \cdot \hat{R}_j)^n + k_s I_s + k_t I_t. \quad (1.16)$$

В формуле (1.16) соответственно означают:

1. k_a, k_d, k_s - коэффициенты рассеянного, диффузного, зеркального отражения соответственно;
2. k_t - коэффициент пропускания;
3. n - степень пространственного распределения Фонга;

В данном случае знак $\hat{\cdot}$ означает что данный вектор нормализован. Значения коэффициентов определяются внешней средой, свойствами материала объектов и длиной волн света. Таким образом возможно рассчитать интенсивность света для отраженной и преломленной части луча. После чего полученные вычисления необходимо выполнить еще раз для отраженного и преломленного луча и т. д., а также сложить полученные интенсивности. Теоретически свет может отражаться бесконечно, так что стоит ограничить число рассматриваемых отражений либо определенным числом, либо не рассматривать лучи с интенсивностью меньше определенного значения [2].

1.3.2 Трассировка лучей в пространстве изображения

Обычно при необходимости расчета отражений и теней уже известны объекты, которые находятся на сцене. При использовании алгоритма трассировки лучей в пространстве изображения (англ. Screen-space reflections, SSR), используется информация о имеющихся объектах из-за чего число рассматриваемых объектов сокращается. Асимптотическая оценка данного алгоритма аналогична асимптотической оценке алгоритма трассировке лучей, однако в данном случае будут анализироваться только видимые объекты [9].

Перед началом алгоритма требуется информация для каждого пикселя:

1. координата Z nearest к наблюдателю поверхности;
2. нормаль данной поверхности.

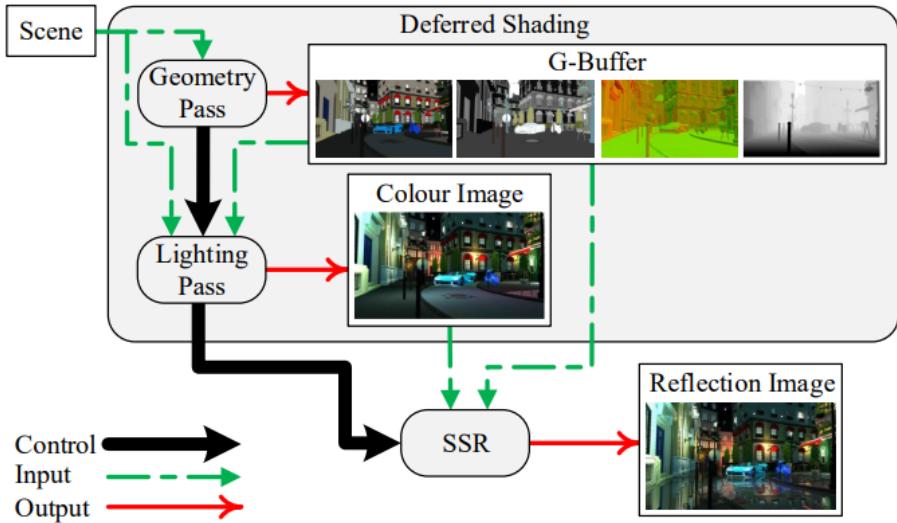


Рисунок 1.6 – Поток данных при использовании SSR

До начала работы самого алгоритма необходимо подготовить данные, что происходит в два этапа:

1. геометрический проход (англ. Geometry pass);
2. световой проход (англ. Lightning pass).

На картинке 1.6 используется понятие **G-buffer**, этот буфер содержит все необходимые данные для начала работы алгоритма, данные для данного буфера будут получены после геометрического прохода. В общем случае он содержит для каждого пикселя:

1. нормали к видимым поверхностям;
2. значение z наименее удаленной видимой фигуры;
3. свойства материалов, значимые для трассировки света (коэффициенты диффузного и зеркального отражения).

При световом проходе для каждого пикселя выбираются источники, которые влияют на его интенсивность. Работа SSR аналогична работе алгоритма ray tracing, однако информация о видимых объектах уже получена и будут рассматриваться только они. Из-за этого, если часть объекта не видима то изображение будет не корректным, как, например, на картинке 1.7 [9; 10].



Рисунок 1.7 – Некорректный расчет отражений при использовании SSR

1.3.3 Трассировка пути

В обратной трассировке лучей считается, что отражение луча идеально, и через каждый пиксель экрана трассируется один луч, однако ввиду данных допущений изображения не являются полностью физически корректными [11]. Данный алгоритм вычисляет интенсивность освещенности с помощью метода Монте-Карло, так как отражения лучей имеют случайный характер [12]. Для реализации метода Монте Карло для каждого пикселя генерируется несколько лучей, после чего результирующее значение их интенсивностей усредняется [13].

Каждый раз, когда луч пересекается с поверхностью, выпускается теневой и случайный отраженный луч. Теневой луч — луч, с помощью которого учитывается прямое освещение (свет излучаемый источником света) для данной точки выборки. Данный луч проводится из точки пересечения к объекту, являющимся источником света в данной сцене. В случае если данный луч пересекает объект сцены, то в данной точке интенсивность прямого света равна 0. Однако в таком случае объект может получить свет от других объектов сцены с помощью отраженных лучей (непрямое освещение). При пересечении трассируемого луча с объектами, в случае если точка пересечения освещена прямым освещением, интенсивность данного освещения будет учитываться при расчете интенсивности света исходного луча. Таким образом при каждом столкновении луча с примитивом учитывается прямое и непрямое освещение, которое будет учитываться при расчете интенсивности отраженного луча. Из-за того что направление лучей рассчитывается случайно, при малом коли-

честве генерируемых лучей на пиксель лучи не пересекут исходные объекты, что образует шумы, которые представлены на картинке 1.8 [11; 14]. Для избежания шумов необходимо увеличить число генерируемых лучей на 1 пиксель. Из-за необходимости в генерации большого количества лучей данный алгоритм трудозатратнее предыдущих и не позволяет наблюдать сцену в реально времени из-за шумов [14]. Трассировка пути позволяет получить более реалистичное изображение, на картинках 1.9 и 1.10 представлены два изображения, полученные с помощью трассировки лучей и трассировок пусты соответственно [15].

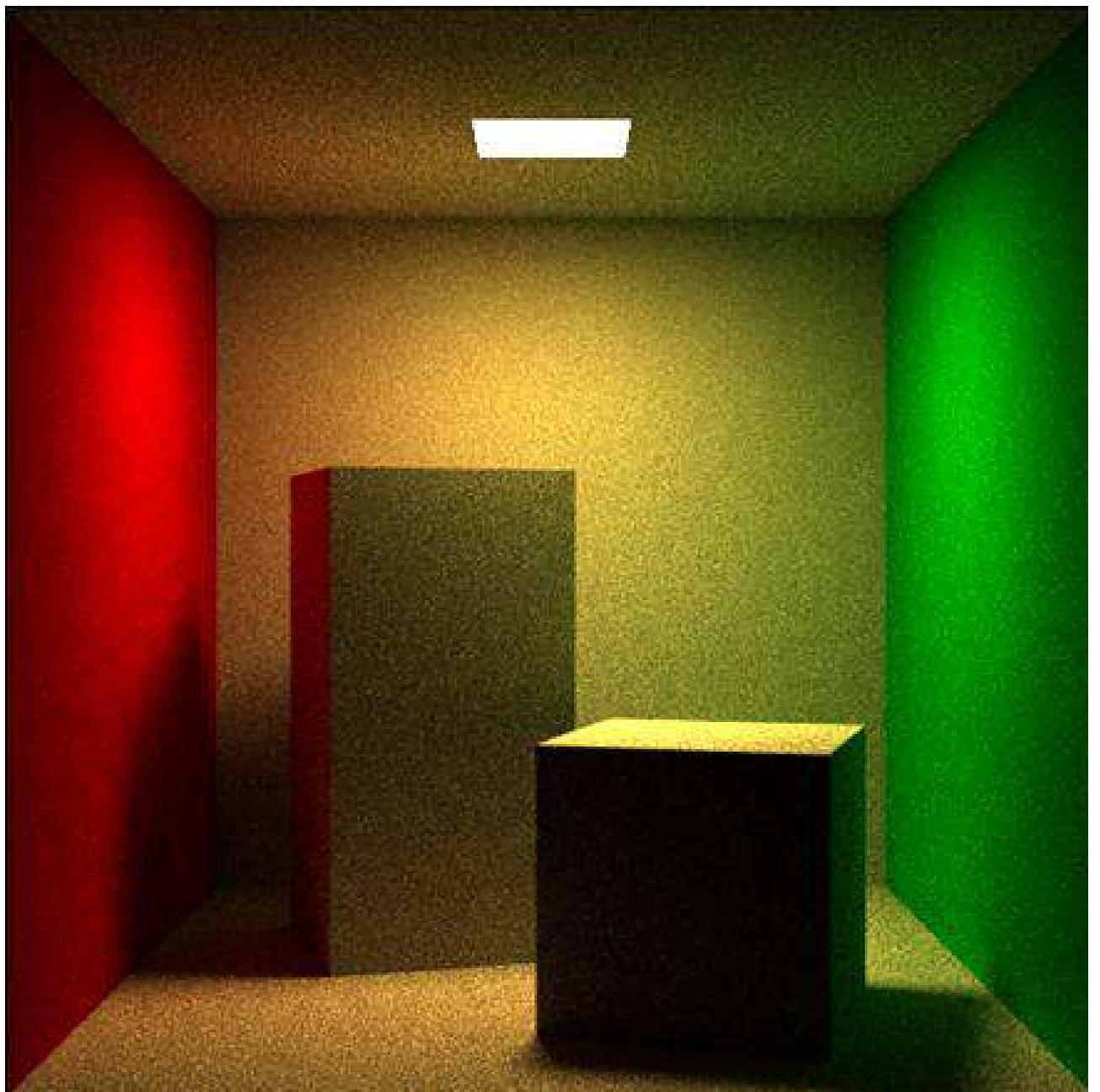


Рисунок 1.8 – Пример шума при использовании трассировки пути



Рисунок 1.9 – Пример кадра комнаты в игре Cyberpunk 2077 с помощью трассировки лучей



Рисунок 1.10 – Пример кадра комнаты в игре Cyberpunk 2077 с помощью трассировки пути

1.3.4 Сравнение алгоритмов

В таблице 1.1 приведено сравнение рассмотренных алгоритмов получения кадра. Цифры в строках таблицы показывают позицию соответствующего алгоритма по данному критерию, при сортировке по возрастанию.

Наиболее трудоемким алгоритмом является алгоритм трассировки пути, так как необходимо генерировать несколько лучей на каждый пиксель экрана и агрегировать результаты вычислений, трассировка лучей в пространстве изображения (SSR), анализирует только видимые с точки зрения наблюдателя объекты, что позволяет совершать меньше операций чем в обратной трассировке лучей [8; 9; 16].

Таблица 1.1 – Сравнение различных алгоритмов визуализации объектов

Критерии сравнения \ Алгоритм	Ray tracing	SSR	Path tracing
Время получения кадра	2	1	3
Реалистичность моделирования	2	1	3
Наличие шумов	нет	нет	да

Трассировка пути позволяет моделировать случайное отражение света, иные алгоритмы в общем случае рассматривают идеальное отражение луча, трассировка лучей в пространстве изображения анализирует отражения от объектов видимых только наблюдателю, из-за чего отражения не всегда получаются правдоподобными [9; 14].

Из-за случайности генерируемых лучей при трассировке пути возможно появление шумов, в иных алгоритмов рассматривается только 1 луч, так что они лишены данной проблемы [14].

При реализации отражений примитивов точность их представления играет важную роль, однако необходимо учитывать возможность генерации правдоподобных отражений в реальном времени. Трассировка пути позволяет получать наиболее реалистичные кадры, однако данный алгоритм требует больших затрат на удаление шума, что не позволяет его использовать в реально времени, трассировка лучей в пространстве изображения демонстрирует нереалистичные кадры при отсутствии объекта в области видимости наблюдателя. Таким образом наилучшим алгоритмом для поставленной задачи является алгоритм обратной трассировки лучей.

Вывод

В данном разделе были проанализированы модели отражения и алгоритмы создания отражений. Таким образом были выбраны:

1. **Алгоритм создания отражений** - Алгоритм обратной трассировки лучей.
2. **Модель отражения** - Модель отражения Фонга.

Входными данными для полученной модели будут являться:

1. интенсивность источника;
2. спектральные характеристики материала примитива;
3. положение источника;
4. положение примитива;
5. угол поворота примитива.

Таким образом возможно построение idef-0 диаграммы, представленной на рисунке 1.11.

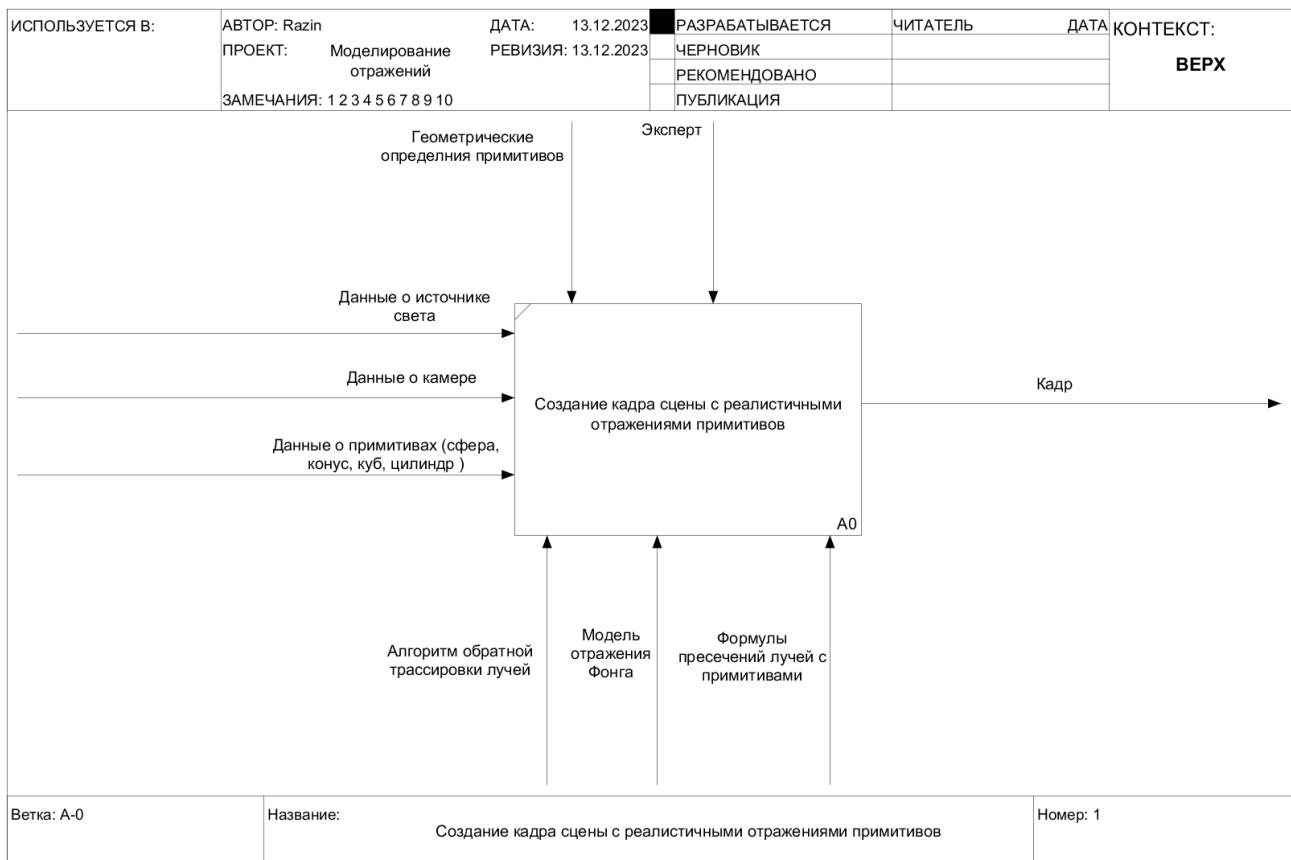


Рисунок 1.11 – Формализация поставленной задачи

2 Конструкторская часть

В данной части работы будут рассмотрены схемы описанных алгоритмов, типы и структуры данных, а также описаны требования к программному обеспечению.

2.1 Требования к программному обеспечению

Программа должна предоставлять следующие возможности:

1. задание положения источника света;
2. изменения положения камеры и направления ее взгляда;
3. визуализация трехмерных примитивов: шар, куб, цилиндр, конус;
4. визуализация отражений, света и теней в соответствии с параметрами примитивов;
5. поддержка перемещения и поворота заданных примитивов, а также изменения их цвета;
6. изменения интенсивности источника света.

2.2 Выбор типов и структур данных

Для формирования общего алгоритма синтеза изображения, необходимо ввести определения соответствующих структур данных.

1. Структура источника света описывается следующими полями:
 - position — вектор положения источника света в пространстве;
 - intensity — интенсивность источника света по каждой составляющей RGB.
2. Структура камеры описывается следующими полями:
 - position — вектор положения камеры в пространстве;
 - view — вектор направления взгляда камеры;
 - up — текущий «верхний» вектор камеры;

- right — текущий «правый» вектор каметры.
3. Структура материала объекта описывается следующими полями:
- Color — цвет данного материала;
 - kA — коэффициент рассеянного освещения;
 - kD — коэффициент диффузного освещения;
 - kS — коэффициент спектрального освещения.
4. Структура луча описывается следующими полями:
- position — точка, лежащая на данном луче;
 - direction — вектор задающий направление луча;
 - t — вещественное число, задающее точку на данном луче.
5. Также вводится структура пересечения луча с примитивом:
- t — вещественное число, задающее точку на трассируемом луче;
 - point — координаты точки пересечения луча с примитивом;
 - normal — нормаль данного примитива, проведенная из точки пересечения;
 - material — материал пересеченного примитива;
 - tracedRay — трассируемый луч.
6. Сцена представляет собой массив с заранее определенным максимальным числом примитивов.
7. Примитивы задаются аналогично их формальному описанию, приведенному в 1.1.1.

2.3 Общий алгоритм построения изображения

Рассмотрим алгоритм построения кадра, пока что не рассматривая конкретную реализацию алгоритма трассировки лучей на картинке 2.1.



Рисунок 2.1 – Общий алгоритм построения кадра

2.4 Алгоритм обратной трассировки лучей

Схема алгоритма трассировки лучей для 1 пикселя приведена на рисунке 2.2. Входными данными для него являются: описания примитивов, интенсивность источника света, положение источника света, максимальное

число поколений луча.

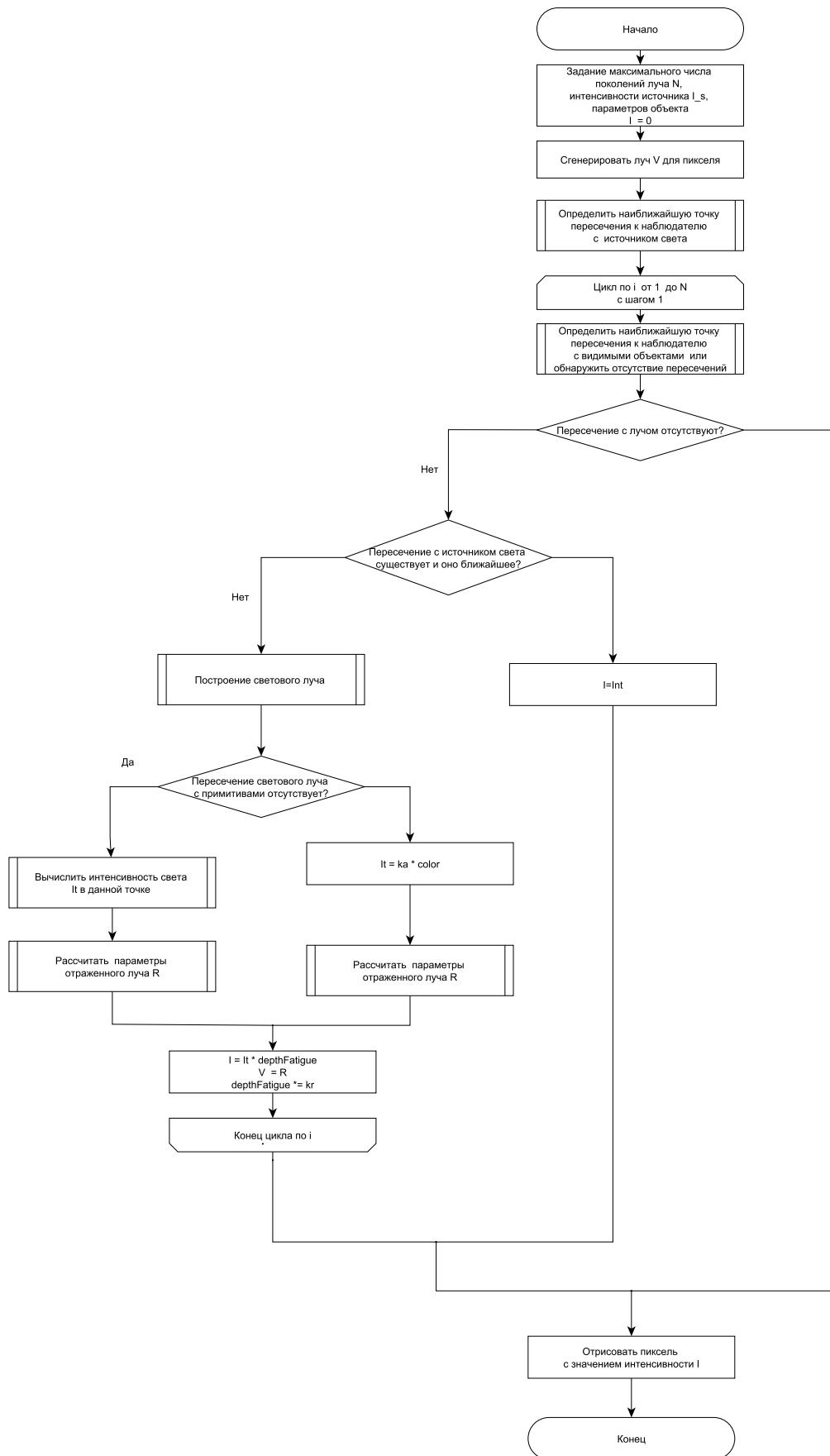


Рисунок 2.2 – Схема алгоритма трассировки лучей для 1 пикселя

Теоретически свет может отражаться бесконечно, введение ограничения на максимальное число поколений луча позволит ограничить время построения изображения, получая реалистичное поведение света. Введены отдельные условия для отображения источника света, данный объект описывается примитивом — сферой и выделяется отдельно, для предотвращений визуализации отражений от него.

Вывод

В данном разделе были рассмотрены типы и структуры данных, необходимые для реализации программы, также были описаны требования к программному обеспечению. Была построена схема алгоритма генерации кадра и схема алгоритма трассировки лучей.

3 Технологическая часть

В данной части рассматривается выбор средств реализации, описывается реализация алгоритмов и приводится интерфейс программного обеспечения.

3.1 Средства Реализации

Для реализации описанных алгоритмов был язык C++, в силу следующих причин:

1. в стандартной библиотеке языка присутствует поддержка всех структур данных, выбранных на этапе проектирования [17];
2. существуют фреймворк для выбранного языка, позволяющий реализовать графический интерфейс [18];
3. язык позволяет использовать библиотеку для упрощения использования графического процессора [19].

Ввиду высокой трудоемкости построения изображения, для ускорения получения кадра и парализации вычислений, был выбран OPENGL, поддерживаемый QT, позволяющий реализовать описанные алгоритмы на языке GLSL для реализации вычислений на графическом процессоре [20].

В качестве среды разработки была выбрана среда разработки CLion, так как она [21]:

1. позволяет использовать утилиту CMake, что позволит конфигурировать проект независимо от платформы;
2. поддерживает выделение glsl кода;
3. обладает необходимым функционалом для сборки, профилирования и отладки программ.

3.2 Реализация алгоритмов

В листинге A.2 приведен код расчета пересечений с рассматриваемыми примитивами на языке glsl, данные функции принимают параметры луча и примитива и рассчитывают параметр t точки пересечения луча, а также нормаль в точке пересечения. Аргументы функций, имеющие префикс

— `out` также являются результатом работы функции. В переменную `normal` сохраняется значение нормали в точке пересечения, в переменную `fraction` сохраняется значение параметра t точки пересечения. Функция возвращает `true` в случае если пересечение с примитивом было обнаружено и `false` иначе. В листинге А.1 приведена реализация расчета интенсивности света луча с использованием модели Фонга. Входными данными для данной функции являются параметры пересечения (нормаль, точка пересечения, трассируемый луч, материал пересеченного объекта), выходными данными являются параметры отраженного луча и интенсивность света в данной точке пересечения.

В листинге А.4 приведена реализация поиска ближайшего пересечения луча со всеми примитивами сцены.

В листинге А.3 приведена реализация алгоритма трассировки лучей. Стоит отметить умножение интенсивности результирующего луча на коэффициент зеркального отражения при каждом отражении, это необходимо для убывания интенсивности света при столкновении луча с примитивом, так как часть энергии света будет поглощена. Также в данной реализации строиться луч к источнику света для визуализации теней. Источник света представляет собой сферу. Для предотвращения отражений от источника, в случае попадания в него первичных (не отраженных) лучей цикл трассировки данного луча прекращается.

3.3 Интерфейс программного обеспечения

На картинке 3.1 представлен интерфейс ПО.

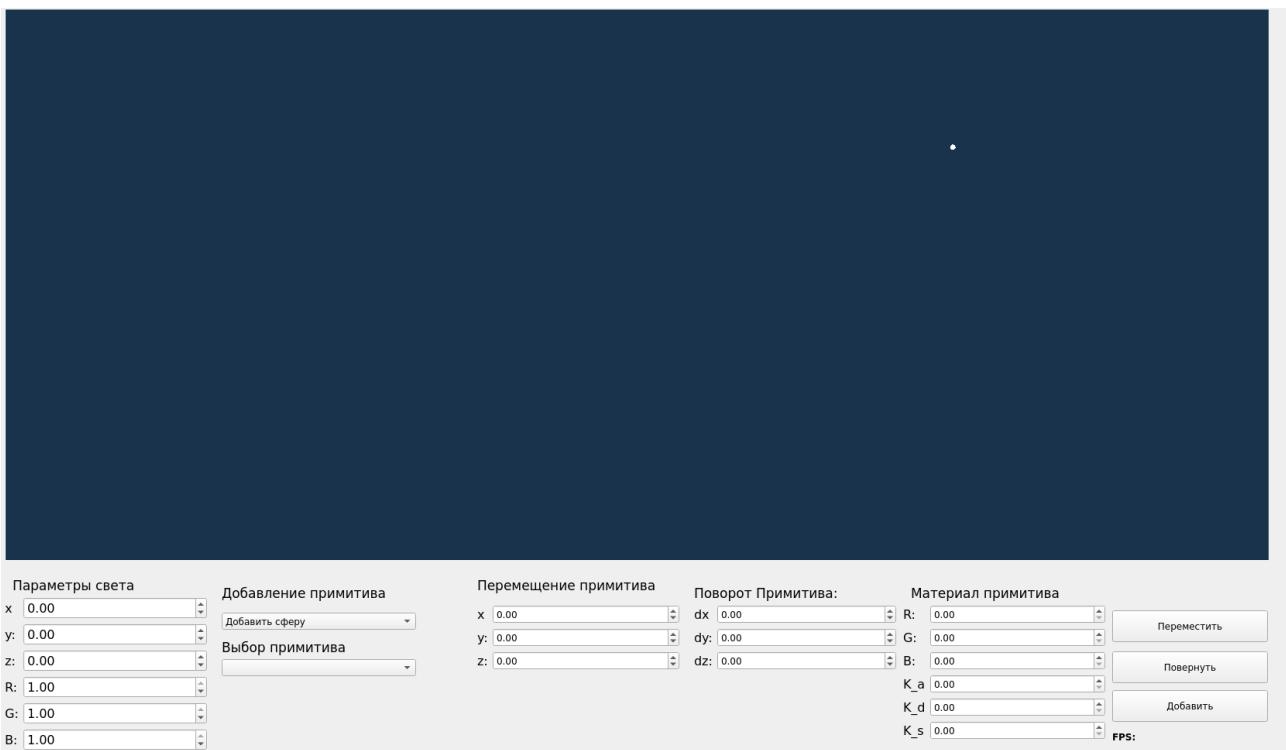


Рисунок 3.1 – Интерфейс разработанной программы

При запуске программы отображается источник освещения, пользователь удален на 2 от центра координат по оси z, источник света находится в центре координат.

При Нажатии на правую кнопку мыши с помощью перемещения мыши пользователь может поворачивать камеру.

Снизу экрана отображается панель управления На картинке 3.1 представлен интерфейс ПО.

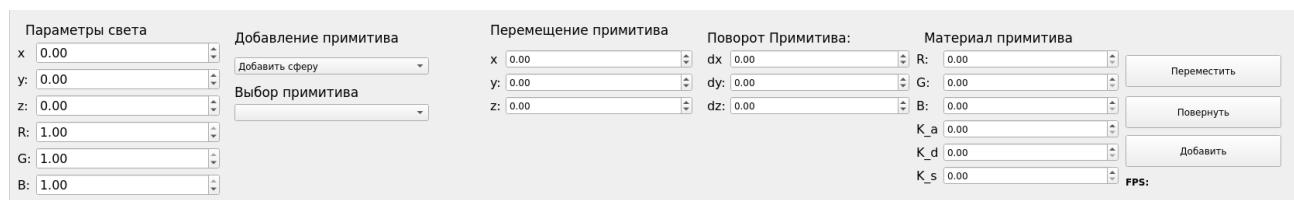


Рисунок 3.2 – Панель управления программы

В графе параметры света описывается текущее положение источника света и интенсивность каждого канала света, в случае изменения значений каждого из данных полей, будут соответственно изменены параметры источника света.

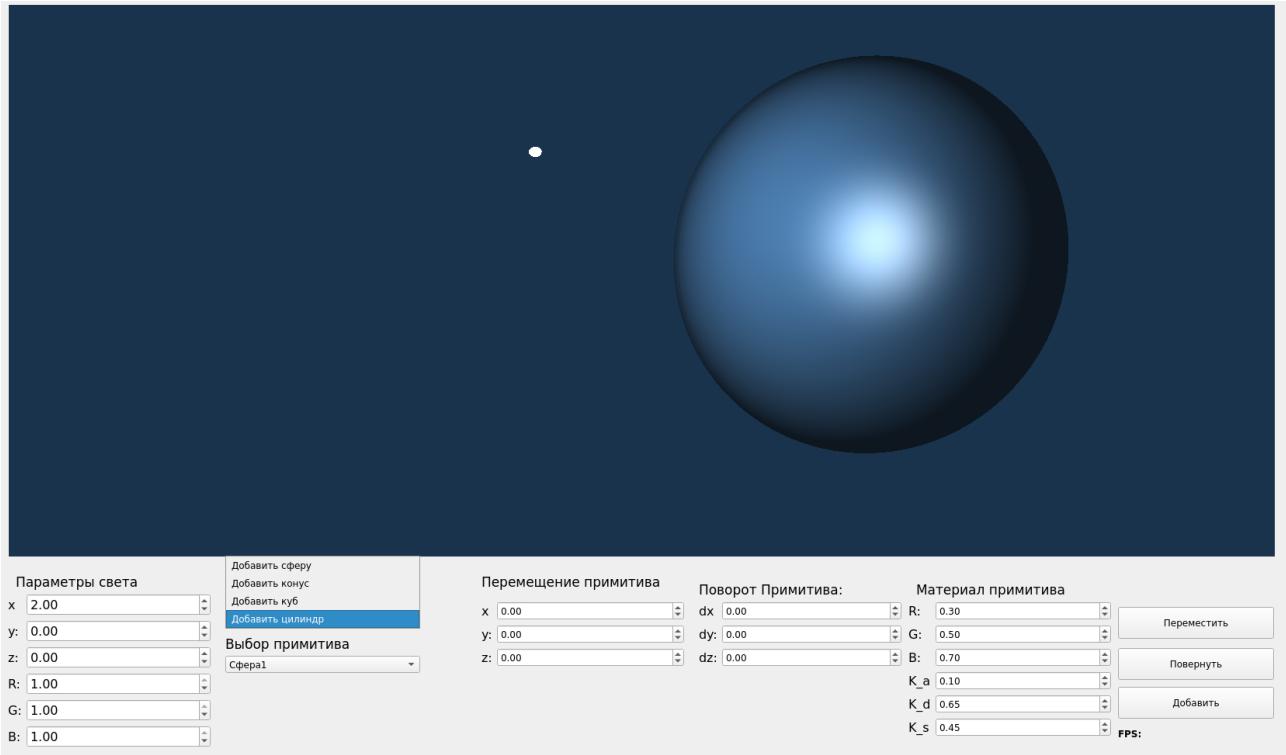


Рисунок 3.3 – Добавление примитивов

Поле «добавление примитива», позволяет добавить один из описанных примитивов, все примитивы при добавлении имеют координаты центра в начале координат и один и тот же материал. При смене примитива в поле «выбор примитива», его текущие характеристики отображаются в полях «перемещение примитива» и «материал примитива».

Поле «материал примитива» описывает, цвет примитива в RGB формате, а также описывает спектральные характеристики (коэффициенты рассеянного, диффузного и зеркального отражения соответственно). В случае изменения данных параметров изменяется материал выбранного примитива на изображении.

В случае нажатия на кнопки «поворнуть», «переместить», центр фигуры переместится в введенные в поле «перемещения примитива координаты». При нажатии на кнопку «поворнуть», выбранный примитив поворачивается на введенные в поля «поворот примитива» градусы вокруг своего центра масс, цилиндр вращается вокруг своей вершины.

3.4 Тестирование

В данной части работы будут рассмотрены тесты реалистичности получаемых изображений, а также функциональные тесты.

3.4.1 Реалистичность отображаемых примитивов

Ожидание: отображение примитивов в соответствии с их геометрическими определениями. Результат приведен на рисунке 3.4:



Рисунок 3.4 – Результат визуализации рассматриваемых примитивов

Тест был успешно пройден.

3.4.2 Смена блика

Ожидание: цвет бликов должен соответствовать цвету освещения. Результат приведен на рисунке 3.5:

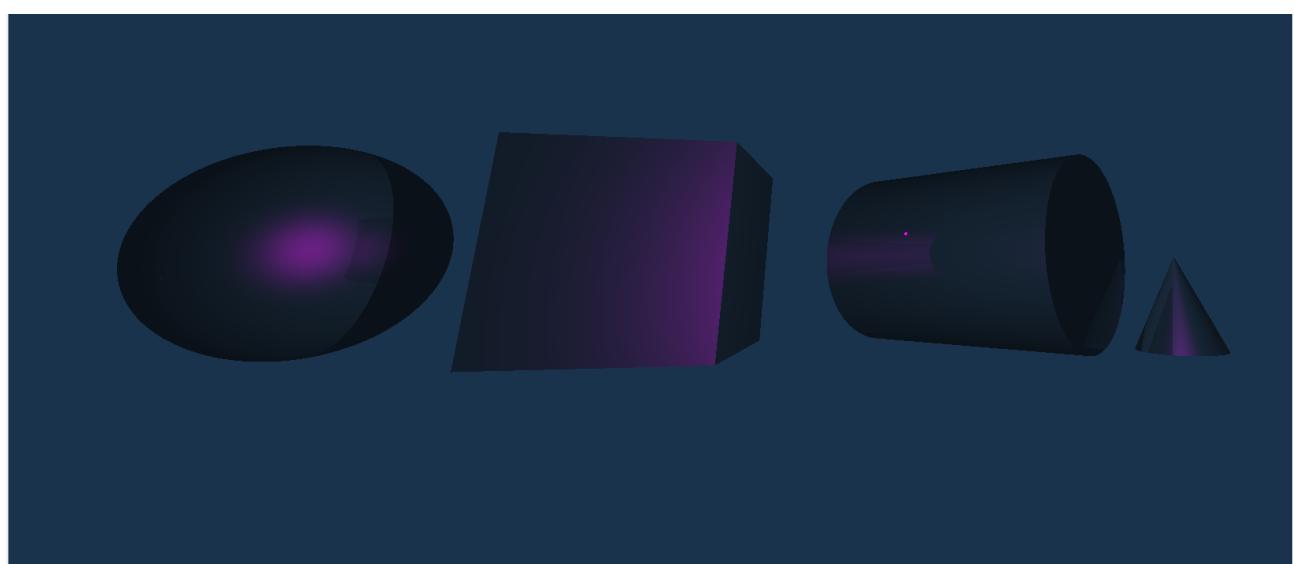


Рисунок 3.5 – Результат визуализации с измененным цветом источника

Тест был успешно пройден.

3.4.3 «Рекурсивное» отражение примитивов

Ожидание: один примитив должен отражаться во втором, затем второй в первом и т.д. Ввиду поглощения света последующие отражения примитивов должны иметь меньшую интенсивность. Результаты тестирования приведены на рисунке 3.6, тестирование было проведено при максимальном числе отражений равном пяти.

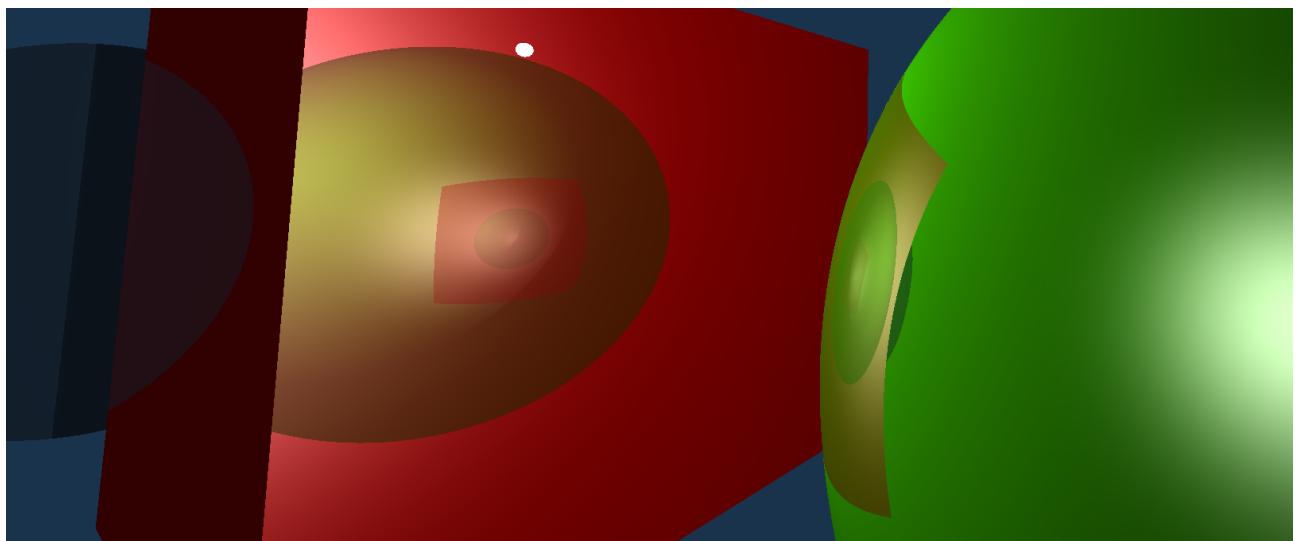


Рисунок 3.6 – Результат визуализации отражений

Тест был успешно пройден.

3.4.4 Отражение нескольких примитивов

Ожидание: на сцене присутствуют 3 примитива, на одном из примитивов присутствуют отражения от иных примитивов. Результаты представлены на рисунке 3.7.

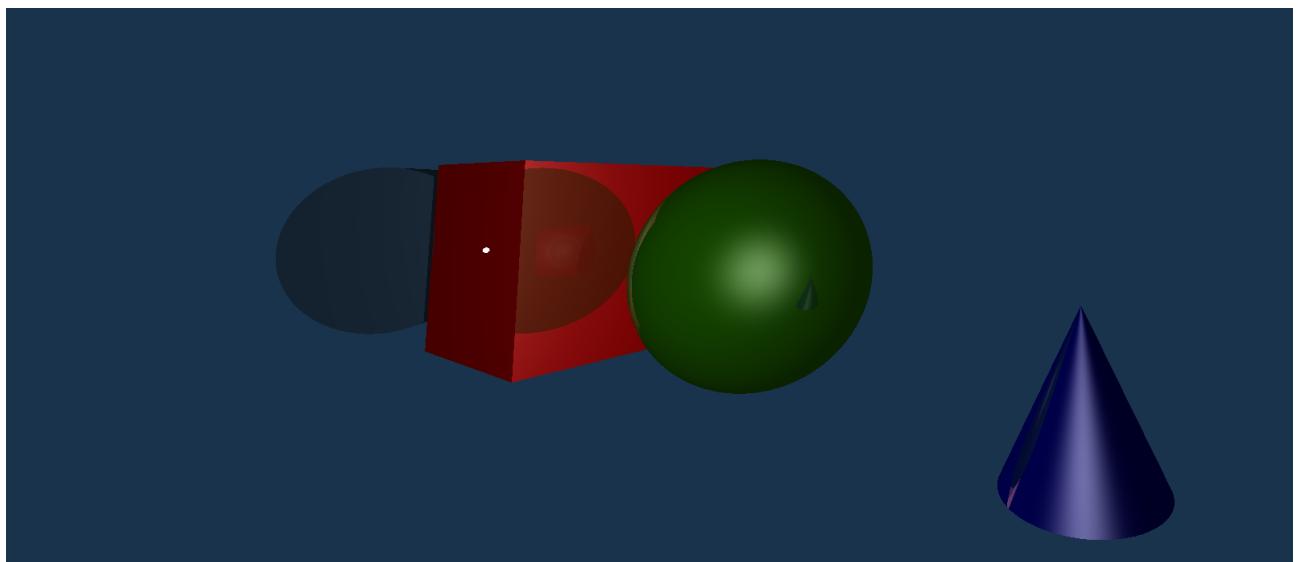


Рисунок 3.7 – Результат визуализации отражений

Тест был успешно пройден.

Вывод

В данном разделе был описан язык программирования, используемые библиотеки и описан разработанный интерфейс, также приведены листинги реализаций основных функций разработанной программы. Также были описаны тесты и приведены получаемые изображения.

4 Исследовательская часть

В данном разделе будет описано исследование зависимости среднего числа генерируемых кадров от числа и типа примитивов на сцене. Также будут описаны технические характеристики устройства, на котором проводились замеры и приведен анализ полученных результатов.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени, представлены далее.

1. Процессор Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 2592 МГц, ядер: 6, логических процессоров: 12.
2. Оперативная память: 16 ГБайт.
3. Операционная система: Microsoft Windows 10 Pro [22].
4. Использованная подсистема: WSL2 [23].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Временные характеристики

Результаты проведения временных замеров приведены в таблице 4.1.

В таблице 4.1 n обозначает число видимых примитивов, иные столбцы обозначают тип примитива, для которого совершался замер. Шаг изменения числа примитивов равен 10, наблюдатель не изменял своей точки наблюдения между замерами, замеры производились при максимальном поколении луча равном 20, для замера времени был использован класс `QEapsedTimer` [24]. Замеры числа кадров происходило 30 секунд, после чего результаты усреднялись и высчитывалось число кадров в секунду, генерируемые примитивы были расставлены в виде решетки (см. 4.1), со смещением 2 по осям X и Z. С помощью таблицы 4.1 был получен график на картинке 4.2.

Таблица 4.1 – Зависимость среднего числа получаемых кадров в секунду от типов примитивов на изображении

n	Сфера	Куб	Цилиндр	Конус
10	57.1419	54.9538	54.3946	55.4482
20	54.3279	50.9881	52.2229	54.4413
30	47.5699	41.1863	46.283	51.0728
40	42.3703	29.4745	37.9257	42.2258
50	33.1457	24.6811	31.8656	35.819
60	30.409	20.0419	27.4269	31.1626
70	26.0904	16.3896	24.1506	26.0774
80	23.2984	14.2909	21.2716	23.4664
90	22.4715	12.416	18.746	20.8784
100	18.6406	11.0113	16.4398	19.3409

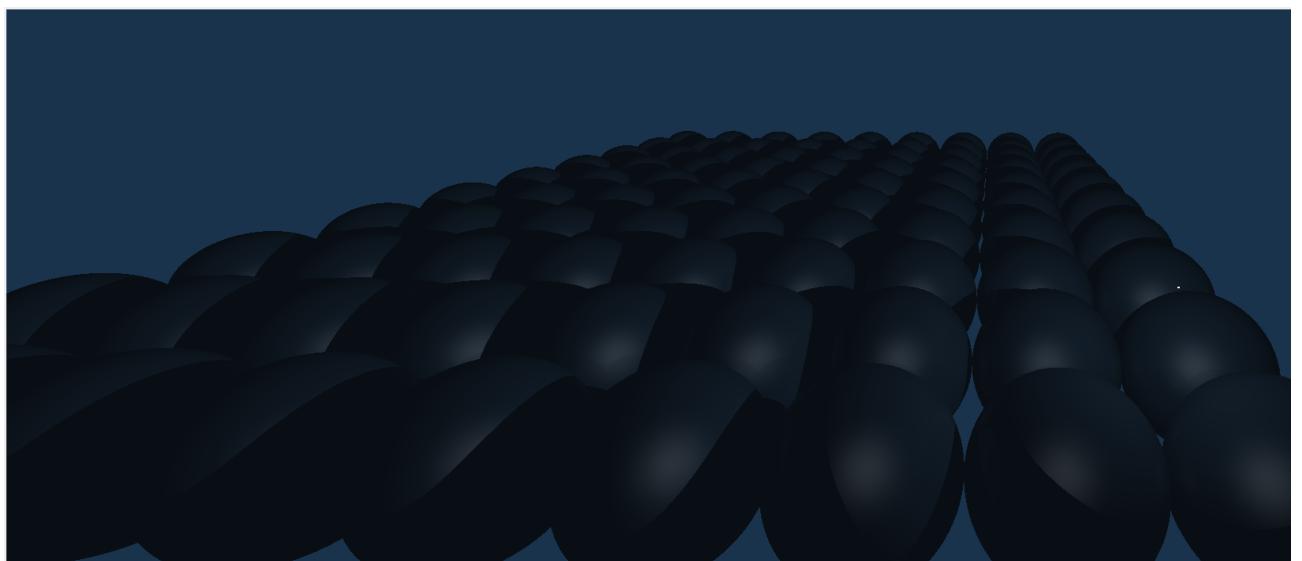


Рисунок 4.1 – Пример расстановки примитивов для проведения замеров

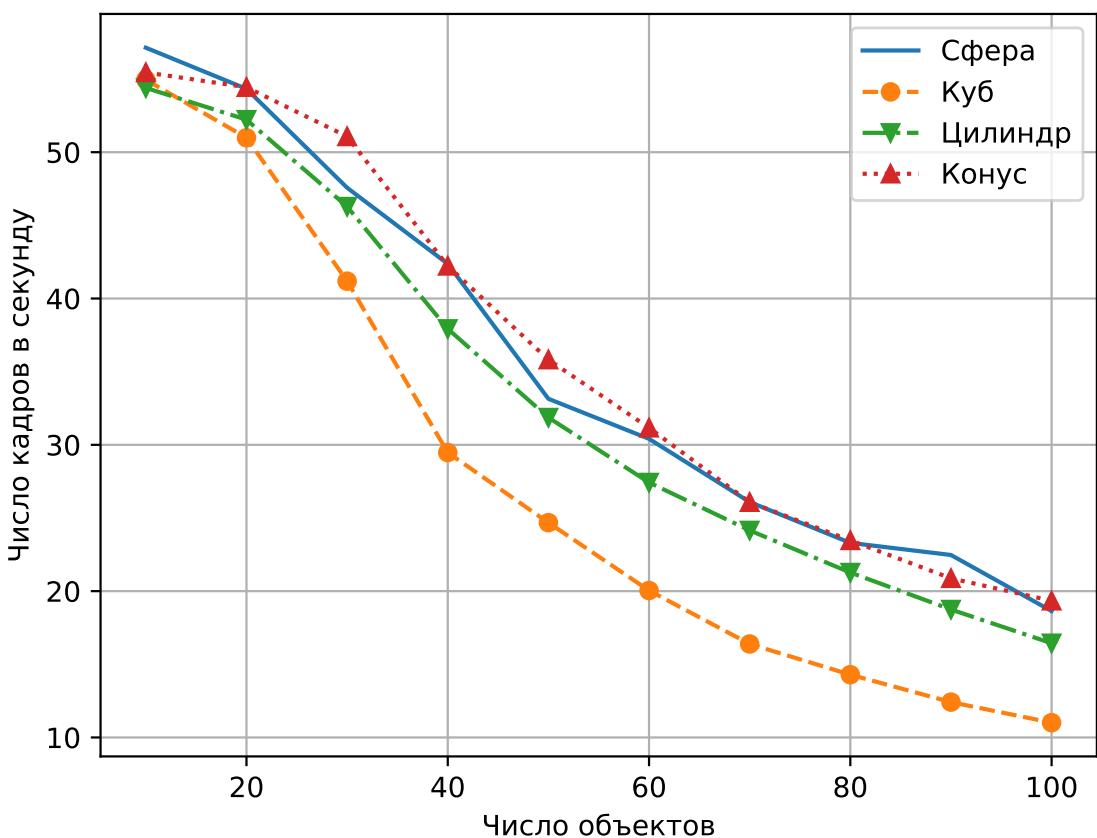


Рисунок 4.2 – Зависимость количества кадров в секунду от числа и типов примитивов на изображении

Из графика на рисунке 4.2, можно сделать вывод, что наибольшее число кадров в секунду было получено при генерации конусов, наименьшее число кадров было получено при генерации кадров с наличием кубов. При увеличении числа примитивов на изображении число кадров убывает с ускорением — это объясняется увеличением числа отражений, которые необходимо рассчитать при добавлении нового примитива.

Куб имеет больший объем по сравнению с другими примитивами, ввиду этого большее число лучей попадают в данный объект, генерируя вторичные лучи для расчета, учитывая расстановку объектов, а также рассмотрение идеальных отражений, все отраженные лучи попадут в соседний куб, что увеличит время генерации кадра. Расчет пересечения с цилиндром и поиск его нормали требует отдельной проверки пересечения луча с его основаниями, этим объясняется большее время получения кадра, чем время получения кадра у сферы и конуса. При 100 объектах на сцене при наблюдении сфер

в среднем было получено 18.64 кадра в секунду, при рассмотрении конусов было получено в 1.04 раза больше кадров, сцена с кубами была сгенерирована в 1.70 меньше раз, при генерации кадров с цилиндрами было получено в 1.13 меньше кадров, чем при использовании сфер.

ЗАКЛЮЧЕНИЕ

В результате исследования, было определено, что время генерации кадра зависит от типа объекта на сцене. Наибольшее число кадров при 100 объектов на сцене было получено при рассмотрении сцены с наличием конусов, что объясняется малыми размерами конуса, при рассмотрении сфер результат был близок к результату конуса, ввиду того, что для построения нормали к сфере необходима одна операция вычитания векторов, для получения точки пересечения необходимо решить уравнение, с введением только одной дополнительной переменной, без необходимости дополнительных математических преобразований. Наименьшее число кадров было получено при использовании кубов, что объясняется большим числом отраженных от них лучей и их наибольшим объемом по сравнению с иными примитивами. При 100 объектах на сцене при наблюдении сфер в среднем было получено 18.64 кадра в секунду, при рассмотрении конусов было получено в 1.04 раза больше кадров, сцена с кубами была сгенерирована в 1.70 меньше раз, при генерации кадров с цилиндрами было получено в 1.13 меньше кадров, чем при использовании сфер.

Поставленная цель: разработка и создание программного обеспечения, моделирующего отражения от геометрических тел был выполнена. Для поставленной цели были выполнены все задачи:

1. формализовать представление объектов сцены и описать их;
2. проанализировать алгоритмы построения реалистичных изображений и теней;
3. выбрать наилучшие алгоритмы для достижения цели из рассмотренных;
4. проанализировать полученную модель взаимодействия света с объектами;
5. выбрать программные средства для реализации модели;
6. реализовать полученную модель и создать интерфейс;
7. провести замеры времени построения кадра от количества и типа примитивов на сцене.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Классификация компьютерной графики по способу формирования изображения, размерности, назначению. — Режим доступа: <https://studfile.net/preview/5404067/page:7/> (дата обращения 01.12.23).
2. Роджерс Д. Алгоритмические основы машинной графики //. — Издательство «Мир». Редакция литературы по математическим наукам, 1989. — С. 512.
3. Компьютерная графика и сферы ее применения. — Режим доступа: <https://moluch.ru/archive/294/66793/?ysclid=1plohpfg4b711269405> (дата обращения 01.12.23).
4. Способы создания фотorealистических изображений [Электронный ресурс]. — Режим доступа: [https://studfile.net/preview/3620224/page:18//](https://studfile.net/preview/3620224/page:18/) (дата обращения 23.07.23).
5. Простые модели освещения [Электронный ресурс]. — Режим доступа: <https://www.cl.cam.ac.uk/teaching/1999/AGraphHCI/SMAG/node2.html> (дата обращения 20.07.23).
6. Raytracing shapes [Электронный ресурс]. — Режим доступа: <https://hugi.scene.org/online/hugi24/coding> (дата обращения 23.07.23).
7. Intersection of a ray and a cone [Электронный ресурс]. — Режим доступа: <https://lousodrome.net/blog/light/2017/01/03/intersection-of-a-ray-and-a-cone/> (дата обращения 23.07.23).
8. Современное состояние методов расчета глобальной освещенности в задачах реалистичной компьютерной графики [Электронный ресурс]. — Режим доступа: <https://cyberleninka.ru/article/n/sovremennoe-sostoyanie-metodov-raschyota-globalnoy-osveschyonnosti-v-zadachah-realisticnoy-kompyuternoy-grafiki-viewer> (дата обращения 15.07.23).
9. Screen Space Reflection Techniques [Электронный ресурс]. — Режим доступа: <https://ourspace.uregina.ca/handle/10294/9245> (дата обращения 15.07.23).

10. Отражение в играх. Как работают, различия и развитие технологий [Электронный ресурс]. — Режим доступа: <https://clck.ru/34zZCf> (дата обращения 15.07.23).
11. Ray Tracing vs. Path-Tracing: What's the Difference? — Режим доступа: <https://history-computer.com/ray-tracing-vs-path-tracing-whats-the-difference/> (дата обращения 01.12.23).
12. Метод Монте-Карло. — Режим доступа: <https://studfile.net/preview/4634775/page:63/> (дата обращения 01.12.23).
13. Трассировка путей (Path tracing). — Режим доступа: <http://www.ray-tracing.ru/articles216.html> (дата обращения 01.12.23).
14. Global Illumination in a Nutshell. — Режим доступа: <https://web.archive.org/web/20110428182101/http://www.thepolygoners.com/tutorials/GIIntro/GIIntro.html> (дата обращения 01.12.23).
15. Ray Tracing vs Path Tracing: Differences and Comparisons. — Режим доступа: <https://www.hardwaretimes.com/ray-tracing-vs-path-tracing-differences-and-comparisons/> (дата обращения 01.12.23).
16. Algorithmic Complexity path tracing. — Режим доступа: https://graphicscodex.courses.nvidia.com/_rn_path.xml.html (дата обращения 01.12.23).
17. C++ Standard Library. — Режим доступа: https://en.cppreference.com/w/cpp/standard_library (дата обращения 01.12.23).
18. Qt c++. — Режим доступа: https://wiki.qt.io/Qt_for_Beginners/ru (дата обращения 01.12.23).
19. Qt OpenGL. — Режим доступа: <https://doc.qt.io/qt-6/qtopengl-index.html> (дата обращения 01.12.23).
20. Realtime Raytracing Shader In Opengl Gsl. — Режим доступа: <https://jethrojeff.com/> (дата обращения 01.12.23).
21. Clion. — Режим доступа: <https://www.jetbrains.com/clion/?ysclid=1ppng5152s770422486> (дата обращения 01.12.23).

22. Windows 10 Pro 2h21 64-bit [Электронный ресурс]. — — Режим доступа: <https://www.microsoft.com/ru-ru/software-download/windows10> (дата обращения: 28.09.2023).
23. Что такое WSL [Электронный ресурс]. — — Режим доступа: <https://learn.microsoft.com/ru-ru/windows/wsl/about> (дата обращения: 28.09.2023).
24. QElapsedTimer Class [Электронный ресурс]. — — Режим доступа: <https://doc.qt.io/qt-6/qelapsedtimer.html> (дата обращения: 13.12.2023).

ПРИЛОЖЕНИЕ А

Листинг А.1 – Реализация расчета интенсивности света по модели Фонга

```
1 vec4 Phong(Intersection intersect, out Ray rayReflected) {  
2  
3     vec3 rayColor = vec3(0.0);  
4     vec3 finalColor = vec3(0.0);  
5  
6  
7     vec3 lightVector = normalize(lightSource.position -  
8         intersect.point);  
9  
10    vec3 shapeNormal = intersect.normal;  
11  
12    vec3 ambientIntensity = intersect.material.lightKoefs[0] *  
13        intersect.material.color;  
14  
15    rayColor += ambientIntensity;  
16  
17    float diffuseLight = dot(shapeNormal, lightVector);  
18    Material shapeMaterial = intersect.material;  
19  
20  
21    float lightIntersect = max(0.0f, diffuseLight);  
22    rayColor += lightIntersect * shapeMaterial.lightKoefs[1] *  
23        intersect.material.color;  
24  
25  
26  
27    Ray reflected = calculateReflected(intersect.tracedRay,  
28        intersect);  
29  
30    float specularDot = pow(max(dot(-reflected.direction,  
31        intersect.tracedRay.direction), 0.0f), 5);  
32  
33    rayColor += shapeMaterial.lightKoefs[2] *  
34        lightSource.intensivity * specularDot;
```

```

32     vec3 point = intersect.tracedRay.origin +
33         intersect.tracedRay.direction * intersect.t;
34
35
36
37     rayReflected = reflected;
38     return vec4(rayColor, 1);
39 }

```

Листинг А.2 – Реализация вычисления пересечений с примитивами

```

1  bool IntersectRayCyl(Ray ray, Cylinder cyl, out float fraction,
2                         out vec3 normal)
3 {
4     vec3 ba = cyl.extr_b - cyl.extr_a;
5     vec3 oc = ray.origin - cyl.extr_a;
6     float baba = dot(ba, ba);
7     float bard = dot(ba, ray.direction);
8     float baoc = dot(ba, oc);
9     float k2 = baba - bard * bard;
10    float k1 = baba * dot(oc, ray.direction) - baoc * bard;
11    float k0 = baba * dot(oc, oc) - baoc * baoc - cyl.ra *
12        cyl.ra * baba;
13    float h = k1 * k1 - k2 * k0;
14    if (h < 0.0)
15    {
16        return false; //no intersection
17    }
18    h = sqrt(h);
19    float t = (-k1 - h) / k2;
20    // body
21    float y = baoc + t * bard;
22    if (y > 0.0 && y < baba)
23    {
24        vec4 normal_n = vec4(t, (oc + t * ray.direction - ba * y
25                               / baba) / cyl.ra);
26        fraction = normal_n.x;
27        normal = normal_n.yzw;
28        return true;
29    }
30
31    t = (((y < 0.0) ? 0.0 : baba) - baoc) / bard;

```

```

29     if (abs(k1 + k2 * t) < h)
30     {
31         vec4 normal_n = vec4(t, ba * sign(y) / sqrt(baba));
32         fraction = normal_n.x;
33         normal = normal_n.yzw;
34         return true;
35     }
36     return false;
37 }
38
39
40 bool IntersectRayCone(Ray r, Cone s, out float fraction, out
41   vec3 normal)
42 {
43     vec3 co = r.origin - s.c;
44
45     float a = dot(r.direction, s.v) * dot(r.direction, s.v) -
46       s.cosa * s.cosa;
47     float b = 2. * (dot(r.direction, s.v) * dot(co, s.v) -
48       dot(r.direction, co) * s.cosa * s.cosa);
49     float c = dot(co, s.v) * dot(co, s.v) - dot(co, co) * s.cosa
50       * s.cosa;
51
52     float det = b * b - 4. * a * c;
53     if (det < 0.0f) return false;
54
55     det = sqrt(det);
56     float t1 = (-b - det) / (2. * a);
57     float t2 = (-b + det) / (2. * a);
58
59     float t = t1;
60     if (t < 0.0f || t2 > 0.0f && t2 < t) t = t2;
61     if (t < 0.) return false;
62
63     vec3 cp = r.origin + t * r.direction - s.c;
64     float h = dot(cp, s.v);
65     if (h < 0. || h > s.h) return false;
66
67     vec3 n = normalize(cp * dot(s.v, cp) / dot(cp, cp) - s.v);
68     fraction = t;

```

```

66     normal = n;
67     return true;
68 }
69
70
71 bool IntersectRaySphere(Ray ray, Sphere sphere, out float
72 fraction, out vec3 normal)
73 {
74     vec3 L = ray.origin - sphere.center;
75     float a = dot(ray.direction, ray.direction);
76     float b = 2.0 * dot(L, ray.direction);
77     float c = dot(L, L) - sphere.radius * sphere.radius;
78     float D = b * b - 4 * a * c;
79
80     if (D < 0.0) return false;
81
82     float r1 = (-b - sqrt(D)) / (2.0 * a);
83     float r2 = (-b + sqrt(D)) / (2.0 * a);
84     if (r1 < 0.0f && r2 < 0.0f)
85         return false;
86     fraction = INF;
87     if (r1 > 0.0)
88         fraction = r1;
89     else if (r2 > 0.0 && fraction > r2)
90         fraction = r2;
91     else
92         return false;
93
94     vec3 point = ray.direction * fraction + ray.origin;
95     normal = normalize(point - sphere.center);
96
97     return true;
98 }
99
100
101 bool IntersectRayBox(Ray ray, Box box, out float fraction, out
102 vec3 normal)
103 {
104     vec3 rd = box.rotation * ray.direction;
105     vec3 ro = box.rotation * (ray.origin - box.position);

```

```

105     vec3 m = vec3(1.0) / rd;
106
107
108     vec3 s = vec3((rd.x < 0.0) ? 1.0 : -1.0,
109                     (rd.y < 0.0) ? 1.0 : -1.0,
110                     (rd.z < 0.0) ? 1.0 : -1.0);
111     vec3 t1 = m * (-ro + s * box.halfSize);
112     vec3 t2 = m * (-ro - s * box.halfSize);
113
114     float tN = max(max(t1.x, t1.y), t1.z);
115     float tF = min(min(t2.x, t2.y), t2.z);
116
117     if (tN > tF || tF < 0.0) return false;
118
119     mat3 txi = transpose(box.rotation);
120
121     if (t1.x > t1.y && t1.x > t1.z)
122         normal = txi[0] * s.x;
123     else if (t1.y > t1.z)
124         normal = txi[1] * s.y;
125     else
126         normal = txi[2] * s.z;
127
128     fraction = tN;
129
130     return true;
131 }
```

Листинг А.3 – Реализация трассировки лучей

```

1  vec4 RayTrace(Ray primary_ray, PrimitiveArrLens lens) {
2      vec4 resColor = vec4(0, 0, 0, 1.0);
3
4      Intersection inters;
5      inters.t = INF;
6      Intersection inters_light;
7      inters_light.t = INF;
8
9      vec4 addColor;
10     vec4 depthFatigue = vec4(1.0, 1.0, 1.0, 1.0);
11
12     Material lightSourcematerial = Material(vec3(1.0, 1.0, 1.0),
13         vec3(0.0f));
```

```

13     Sphere lightSourceSphere = Sphere(lightSource.position ,
14         R_LSOURCE , lightSourcemat);
15
16     float fr;
17     vec3 nr;
18     bool lighInter = IntersectRaySphere(primary_ray ,
19         lightSourceSphere , fr , nr);
20
21     for (int i = 0; i < MAX_DEPTH; i++)
22     {
23         inters = findIntersection(primary_ray , spheres , boxes ,
24             cylinders , lens);
25         if (i == 0)
26         {
27             if (abs(inters.t - INF) < EPS)
28             {
29                 if (lighInter)
30                 {
31                     resColor = vec4(lightSource.intensivity , 1);
32                 }
33                 else
34                 {
35                     resColor = vec4(0.1 , 0.2 , 0.3 , 1.0);
36                 }
37                 break;
38             }
39             else if (lighInter && fr > 0.0 && fr < inters.t)
40             {
41                 resColor = vec4(lightSource.intensivity , 1);
42                 break;
43             }
44
45
46
47
48     vec3 lightVector = normalize(lightSource.position -
49         inters.point);

```

```

50     vec3 shadow_orig = dot(lightVector, inters.normal) <= 0
51         ? inters.point - inters.normal * 0.02 : inters.point
52         + inters.normal * 0.02;
53
54
55     if (inters_light.t < INF)
56     {
57         addColor = vec4(inters.material.color *
58                         inters.material.lightKoefs[0], 1.0);
59         primary_ray = calculateReflected(primary_ray,
60                                         inters);
61     }
62     else
63     {
64         addColor = Phong(inters, primary_ray);
65     }
66
67     resColor += depthFatigue * addColor;
68     depthFatigue *= inters.material.lightKoefs[2];
69
70     return resColor;
}

```

Листинг А.4 – Реализация поиска наиближайшего видимого примитива сцены

```

1 Intersection findIntersection(Ray ray, Sphere
2     spheres[SPHERE_COUNT], Box boxes[BOX_COUNT], Cylinder
3     cylinders[CYLINDERS_COUNT], PrimitiveArrLens sizes)
4 {
5     float minDistance = INF;
6     float D = INF;
7     vec3 N;
8     Intersection inters;
9     inters.t = minDistance;
10    for (int i = 0; i < sizes.size_spheres; i++)
11    {
12        if (IntersectRaySphere(ray, spheres[i], D, N))
13        {
14            if (D < minDistance && D > 0.0f)
15            {
16                minDistance = D;
17                N = N;
18            }
19        }
20    }
21    inters.N = N;
22    inters.t = minDistance;
23    return inters;
}

```

```

13    {
14        inters.normal = N;
15        inters.tracedRay = ray;
16        inters.point = ray.origin + ray.direction * D;
17        inters.material = spheres[i].material;
18        inters.t = D;
19        minDistance = D;
20    }
21}
22
23}
24
25
26
27 for (int i = 0; i < sizes.size_boxes; i++)
28 {
29     if (IntersectRayBox(ray, boxes[i], D, N))
30     {
31         if (D < minDistance && D > 0.0f)
32         {
33             inters.normal = N;
34             inters.tracedRay = ray;
35             inters.point = ray.origin + ray.direction * D;
36             inters.material = boxes[i].material;
37             inters.t = D;
38             minDistance = D;
39         }
40     }
41 }
42
43
44 for (int i = 0; i < sizes.size_cylinders; i++)
45 {
46     if (IntersectRayCyl(ray, cylinders[i], D, N))
47     {
48
49         if (D < minDistance && D > 0.0f)
50         {
51             inters.normal = N;
52             inters.tracedRay = ray;
53             inters.point = ray.origin + ray.direction * D;

```

```

54         inters.material = cylinders[i].material;
55         inters.t = D;
56         minDistance = D;
57     }
58 }
59
60
61 for (int i = 0; i < sizes.size_cones; i++)
62 {
63     if (IntersectRayCone(ray, cones[i], D, N))
64     {
65
66         if (D < minDistance && D > 0.0f)
67     {
68         inters.normal = N;
69         inters.tracedRay = ray;
70         inters.point = ray.origin + ray.direction * D;
71         inters.material = cones[i].material;
72         inters.t = D;
73         minDistance = D;
74     }
75 }
76
77
78
79 return inters;
80
81 }

```