|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

***«*Моделирование испускаемого RGB лампочкой света*»***

Студент \_\_\_\_\_\_ИУ7-52Б\_\_\_\_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**Д.В. Колосов**\_\_\_\_\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_**А.А. Павельев **\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2020 г.*

Оглавление

[Введение 2](#_Toc64418129)

[1. Аналитическая часть 4](#_Toc64418130)

[1.1 Постановка задачи 4](#_Toc64418131)

[1.2 Разбор моделей, моделируемых на сцене 4](#_Toc64418132)

[1.3 Алгоритмы, удаляющие невидимые поверхности и грани 5](#_Toc64418133)

[1.3.1 Алгоритм с использованием Z-буфера 5](#_Toc64418134)

[1.3.2 Алгоритм Варнока 7](#_Toc64418135)

[1.3.3 Алгоритм Робертса 7](#_Toc64418136)

[1.3.4 Алгоритм трассировки лучей (обратной трассировки лучей) 9](#_Toc64418137)

[Вывод 10](#_Toc64418138)

[1.4 Алгоритмы, моделирующие освещение 11](#_Toc64418139)

[1.4.1 Модель Ламберта 11](#_Toc64418140)

[1.4.2 Модель Фонга 12](#_Toc64418141)

[1.4.3 Модель Блинна-Фонга 14](#_Toc64418142)

[Вывод 14](#_Toc64418143)

[Вывод из аналитической части 14](#_Toc64418144)

[2. Конструкторская часть 15](#_Toc64418145)

[2.1 Схемы алгоритмов 16](#_Toc64418146)

[2.2 Пересечение луча с объектами сцены 20](#_Toc64418147)

[2.2.1 Пересечение луча с плоскостью 20](#_Toc64418148)

[2.2.2 Пересечение луча со сферой 21](#_Toc64418149)

[2.2.3 Пересечение луча с цилиндром 22](#_Toc64418150)

[2.3 Нахождение отраженного луча 23](#_Toc64418151)

[Вывод из конструкторской части 24](#_Toc64418152)

[3. Технологическая часть 25](#_Toc64418153)

[3.1. Средства реализации 25](#_Toc64418154)

[3.2. Описание основных этапов реализации 25](#_Toc64418155)

[3.3. Описание интерфейса программы 29](#_Toc64418156)

[Вывод из технологической части 31](#_Toc64418157)

[4. Исследовательская часть 32](#_Toc64418158)

[4.1. Постановка эксперимента по замеру времени 32](#_Toc64418159)

[Вывод из исследовательской части 33](#_Toc64418160)

[Заключение 34](#_Toc64418161)

[Список литературы 34](#_Toc64418162)

# Введение

**Компьютерная графика** — область деятельности, в которой компьютерные технологии используются **для создания изображений**, а также **обработки визуальной информации**.

В настоящее время в компьютерной графике большое внимание уделяется алгоритмам получения реалистичных изображений. Так как данные алгоритмы должны учитывать множество физических явления, таких как преломление, различные виды отражения и т.п. – они являются довольно требовательными к ресурсам системы. Степень затратности напрямую зависит от реалистичности изображения, степени его проработанности. Большая затратность становится критической проблемой при моделировании динамических сцен, где есть необходимо быстрого получения следующего кадра.

Для успешного решения поставленной задачи необходимо произвести её анализ, произвести декомпозицию на более мелкие задания, для каждого из задания исследовать возможные способы его решения и выбрать оптимальный. Решение задачи необходимо написать на выбранном языке программирования и спроектировать удобный для пользователя интерфейс.

Целью данной работы является разработка программы для моделирования трёхмерного изображения объектов и моделирования RGB-лампочки с исходящим от неё цветным светом.

В рамках выполнения курсового проекта должны быть решены следующие задачи:

* анализ алгоритмов компьютерной графики, использующихся для создания реалистичной модели взаимно перекрывающихся объектов, и выбор наиболее подходящего для решения поставленной задачи;
* проектирование архитектуры программы;
* реализация выбранных алгоритмов и структур данных;
* разработка программного обеспечения, которое позволит отобразить трехмерную сцену и визуализировать цветной свет;
* проведение исследования на основе разработанной программы.

Результатом выполнения работы является программное обеспечение, моделирующее цветной свет, исходящий от RGB лампочки. В программе должна иметься возможность изменять цвет света, испускаемого лампочкой.

# 1. Аналитическая часть

В данном разделе представлены постановка задачи, критерии выбора алгоритмов, ограничения, анализ алгоритмов и методов, средства для реализации поставленной задачи.

## 1.1 Постановка задачи

Необходимо смоделировать примитивные объекты на сцене и RGB-лампочку с исходящим от неё цветным светом. При моделировании необходимо учитывать различные свойства объектов на сцене, такие как блеск и отражения, а также свойство непрозрачных объектов отбрасывать тени.

## 1.2 Разбор моделей, моделируемых на сцене

Сцена состоит из следующих элементов:

* точечный источник света (RGB-лампочка), полностью характеризующийся местоположением в пространстве OXYZ и тремя значениями яркости: по одному значению на каждый канал (RGB);
* примитивы: сфера, цилиндр;
* плоскость основания.

В компьютерной графике существует несколько способов описания моделей из реальной жизни:

* объемная модель;
* каркасная модель;
* поверхностная модель.

Каркасная модель не обладает достаточной реалистичностью, имеется возможность неоднозначно интерпретировать ориентацию и видимость граней, чего нельзя сказать об объёмной и поверхностной моделях. В объёмной модели, в свою очередь, добавляется ненужная информация о расположении материала. Таким образом аналитически было выяснено, что поверхностная модель наилучшим образом подходит для реализации поставленной цели.

## 1.3 Алгоритмы, удаляющие невидимые поверхности и грани

Для выбора алгоритма, решающего поставленную задачу наилучшим образом, необходимо произвести анализ всех потенциальных алгоритмов.

### 1.3.1 Алгоритм с использованием Z-буфера

Алгоритм, использующий z-буфер это один из простейших алгоритмов удаления невидимых поверхностей. Работает этот алгоритм в пространстве изображения. Идея z-буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пикселя в пространстве изображения, z-буфер - это отдельный буфер глубины, используемый для запоминания координаты z или глубины каждого видимого пикселя в пространстве изображения. В процессе работы глубина или значение z каждого нового пиксела, который нужно занести в буфер кадра, сравнивается с глубиной того пиксела, который уже занесен в z-буфер. Если это сравнение показывает, что новый пиксел расположен впереди пиксела, находящегося в буфере кадра, то новый пиксел заносится в этот буфер и, кроме того, производится корректировка z-буфера новым значением z. Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по х и у наибольшего значения функции z (х, у).

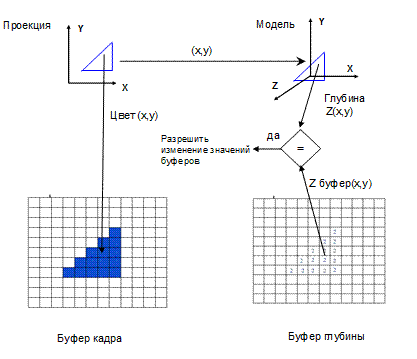


Рисунок 1.1 Работа алгоритма, использующего Z-буфер.

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. Поскольку габариты пространства изображения фиксированы, оценка вычислительной трудоемкости алгоритма не более чем линейна. Поскольку элементы сцены или картинки можно заносить в буфер кадра или в *z*-буфер в произвольном порядке, их не нужно предварительно сортировать по приоритету глубины.

Основной недостаток алгоритма - большой объем требуемой памяти. Если сцена подвергается видовому преобразованию и отсекается до фиксированного диапазона значений координат *z*, то можно использовать *z*-буфер с фиксированной точностью.

Несмотря на то, что в данном алгоритме можно реализовать эффекты прозрачности и просвечивания, эффект отражения в данном алгоритме не представляется возможным, что существенно скажется на реалистичности изображения, в следствие чего данный алгоритм следует исключить из потенциально пригодных для реализации задачи.

1.3.2 Алгоритм Варнока

Алгоритм Варнока является одним из примеров алгоритма, основанного на разбиении картинной плоскости на части, для каждой из которых исходная задача может быть решена достаточно просто.

Поскольку алгоритм Варнока нацелен на обработку картинки, он работает в пространстве изображения. В пространстве изображения рассматривается окно и решается вопрос о том, пусто ли оно, или его содержимое достаточно просто для визуализации. Если это не так, то окно разбивается на фрагменты до тех пор, пока содержимое фрагмента не станет достаточно простым для визуализации или его размер не достигнет требуемого предела разрешения.

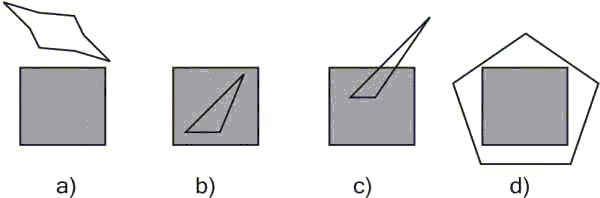


Рисунок 1.2 Примеры расположения окна.

Алгоритм Варнока не берёт во внимания свойства материала объекта (например, отражающие свойства), вследствие чего не может быть выбран для реализации задачи.

### **1.3.3 Алгоритм Робертса**

Алгоритм Робертса представляет собой первое известное решение задачи об удалении невидимых линий. Это математически элегантный метод, работающий в объектном пространстве. 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. Трудоёмкость алгоритма напрямую зависит от числа обрабатываемых объектов и теоретически имеет сложность, равную квадрату числа объектов на сцене. Несмотря на то, что математические методы, используемые в этом алгоритме, просты, мощны и точны, данный алгоритм не может использоваться для решения поставленной задачи в виду огромных вычислительных затрат, а также неспособности алгоритма к моделированию оптических свойств объекта.

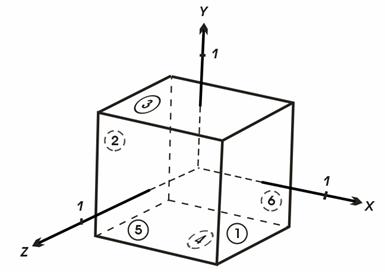


Рисунок 1.3 Пример работы алгоритма Робертса: Куб с центром в начале координат.

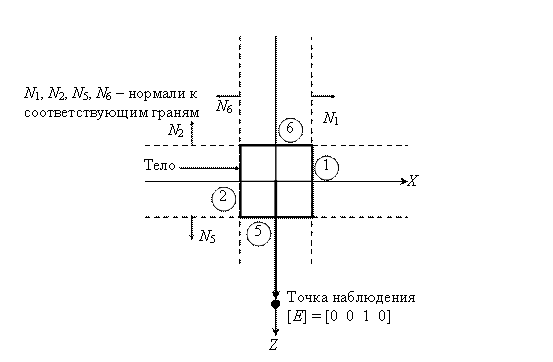


Рисунок 1.4 Пример работы алгоритма Робертса: Точка наблюдения на куб «вне тела».

**1.3.4 Алгоритм трассировки лучей (обратной трассировки лучей)**

Метод обратной трассировки лучей был разработан в начале 80-х годов и применялся для создания высококачественных реалистичных изображений не в реальном масштабе времени. Несмотря на медленную скорость сцены, качество получаемого изображения было максимально приближено к реальности. Метод обратной трассировки лучей позволяет получать такие эффекты, как отражение, преломление, затенение и т.д. Достоинство данного метода заключается также в том, что можно работать со сценами, заданными не набором треугольников для аппроксимации гладких поверхностей, а, собственно, самими этими поверхностями в математической форме. Таким образом задача построения изображения такой поверхности решается точно, а не приближённо, как если бы такая поверхность была аппроксимирована набором треугольников.

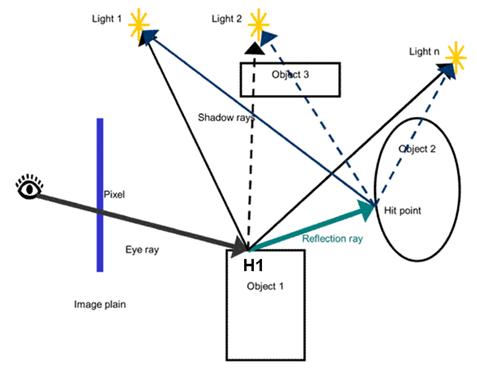


Рисунок 1.5 Пример обработки сцены с использованием алгоритма обратной трассировки лучей

Очевидным недостатком алгоритма обратной трассировки лучей является производительность, т.к. луч, например, может падать на объект, обладающий отражающими свойствами, что в свою очередь вызовет рекурсивный вызов функции. Но существуют множество методов оптимизации обратной трассировки лучей, например, кеширование данных или использование многопоточности.

### Вывод

В ходе анализа потенциальных алгоритмов был выбран алгоритм обратной трассировки лучей, т.к. он позволяет учитывать различные свойства материалов, а также работать с поверхностной моделью, что позволяет построить крайне реалистичные изображения.

## 1.4 Алгоритмы, моделирующие освещение

Реалистичность моделируемого изображения во многом зависит от правильного выбора модели освещения. В общем случае модели освещения делятся на локальные и глобальные. Локальные модели освещения учитывают только первичные источники света, не учитывая перенос света между поверхностями. В свою очередь глобальные модели освещения затрагивают такие аспекты, как преломление света, многократное отражение света, и т.п.

### 1.4.1 Модель Ламберта

Модель Ламберта моделирует идеальное диффузное освещение. Считается, что свет при попадании на поверхность рассеивается равномерно во все стороны. При расчете такого освещения учитывается только ориентация поверхности (нормаль N) и направление на источник света (вектор L).

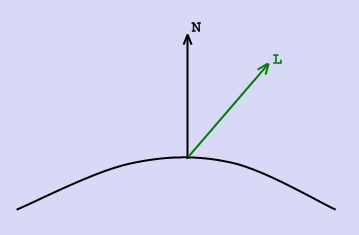


Рисунок 1.6 Модель освещения Ламберта: нормаль N и направление источника света L.

Рассеянная составляющая рассчитывается по закону косинусов (закон Ламберта):

где  
– рассеянная составляющая освещенности в точке,

- свойство материала воспринимать рассеянное освещение,

– мощность рассеянного освещения,  
L – направление из точки на источник,  
N - вектор нормали в точке.

Модель Ламберта является одной из самых простых моделей освещения. Данная модель очень часто используется в комбинации других моделей, практически в любой другой модели освещения можно выделить диффузную составляющую. Более-менее равномерная часть освещения (без присутствия какого-либо всплеска) как правило будет представляться моделью Ламберта с определенными характеристиками. Данная модель может быть очень удобна для анализа свойств других моделей (за счет того, что ее легко выделить из любой модели и анализировать оставшиеся составляющие).

### 1.4.2 Модель Фонга

Модель Фонга – классическая модель освещения. Модель представляет собой комбинацию диффузной составляющей (модели Ламберта) и зеркальной составляющей и работает таким образом, что кроме равномерного освещения на материале может еще появляться блик. Местонахождение блика на объекте, освещенном по модели Фонга, определяется из закона равенства углов падения и отражения. Если наблюдатель находится вблизи углов отражения, яркость соответствующей точки повышается.

Падающий и отраженный лучи лежат в одной плоскости с нормалью к отражающей поверхности в точке падения, и эта нормаль делит угол между лучами на две равные части. Т.о. отраженная составляющая освещенности в точке зависит от того, насколько близки направления на наблюдателя и отраженного луча.

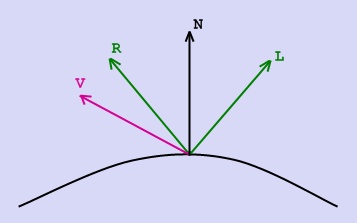


Рисунок 1.7. Модель освещения Фонга: N - нормаль, L - направление источника света, R – направление отражённого луча, V – вектор, направленный «на наблюдателя».

Это можно выразить следующей формулой:

где  
Is – зеркальная составляющая освещенности в точке,  
ks – коэффициент зеркального отражения,  
is – мощность зеркального освещения,  
R – направление отраженного луча,  
V - направление на наблюдателя,  
α - коэффициент блеска, свойство материала.

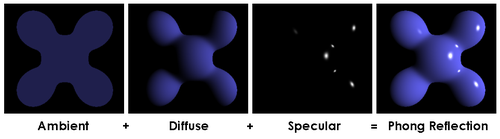


Рисунок 1.8 Модель освещения Фонга. Пример

### 1.4.3 Модель Блинна-Фонга

Модель Блинна-Фонга – это унификация модели Фонга, исключающая расчет отраженного луча, что упрощает вычисления. Принципиальной разницы между двумя этими моделями нет.

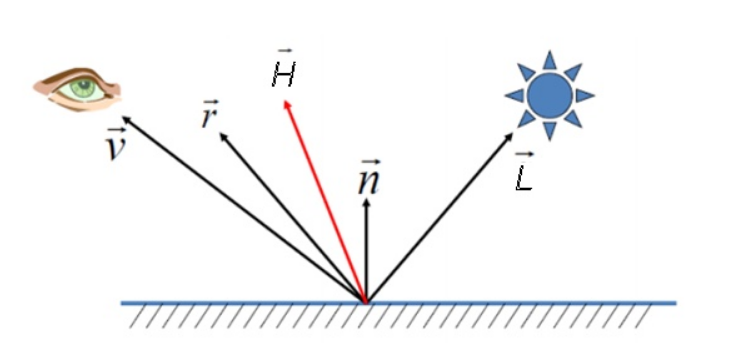


Рисунок 1.9 Модель освещения Блинна-Фонга: в отличие от простой модели Фонга добавляется дополнительный вектор H – средний вектора между отражённым вектором V и нормалью n.

### Вывод

Для моделирования освещения будет выбрана модель Фонга, так как она позволяет наиболее реалистичным образом смоделировать свет на сцене.

## Вывод из аналитической части

На основе сравнительного анализа для реализации поставленных задач были выбраны следующие алгоритмы:

* для представления всех объектов сцены будет использоваться поверхностная модель;
* для моделирования освещения будет использовать модель освещения Фонга;
* для удаления невидимых поверхностей и граней будет использоваться алгоритм обратной трассировки лучей.

# 2. Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов для обратной трассировки лучей и модели освещения Фонга, поиск пересечение луча с объектами сцены.

Программа должна обладать следующей функциональностью:

1. Визуализировать трехмерную сцену, состоящую из примитивных объектов в режиме реального времени.
2. Предоставлять в интерфейсе возможность пользователю выполнять следующие действия:
   1. менять цвет RGB-лампочки;
   2. позволить пользователю изменять координаты камеры, «перемещаться» по сцене;
   3. позволить пользователю поворачивать камеру относительно осей;
   4. позволить пользователю сохранять сгенерированное изображение.

## 2.1 Схемы алгоритмов

Ниже, на рисунке 2.1 представлена общая схема алгоритма обратной трассировки лучей.

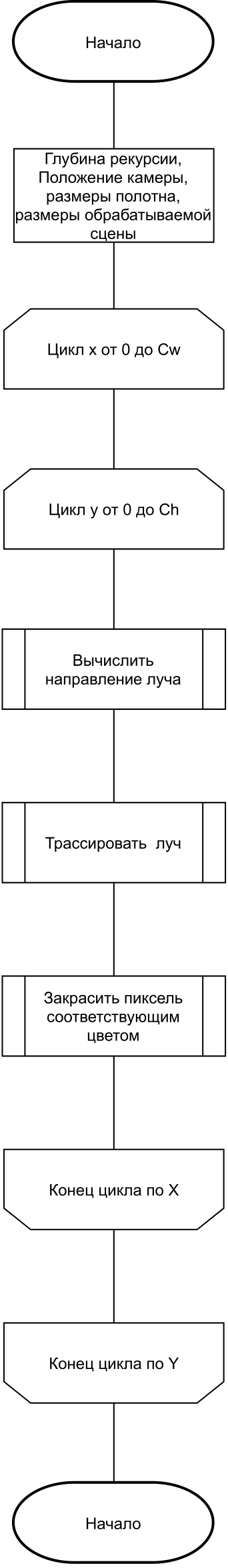


Рисунок 2.1 Схема алгоритма обратной трассировки лучей.

Ниже, на рисунке 2.2 представлена схема алгоритма функции «трассировать луч».

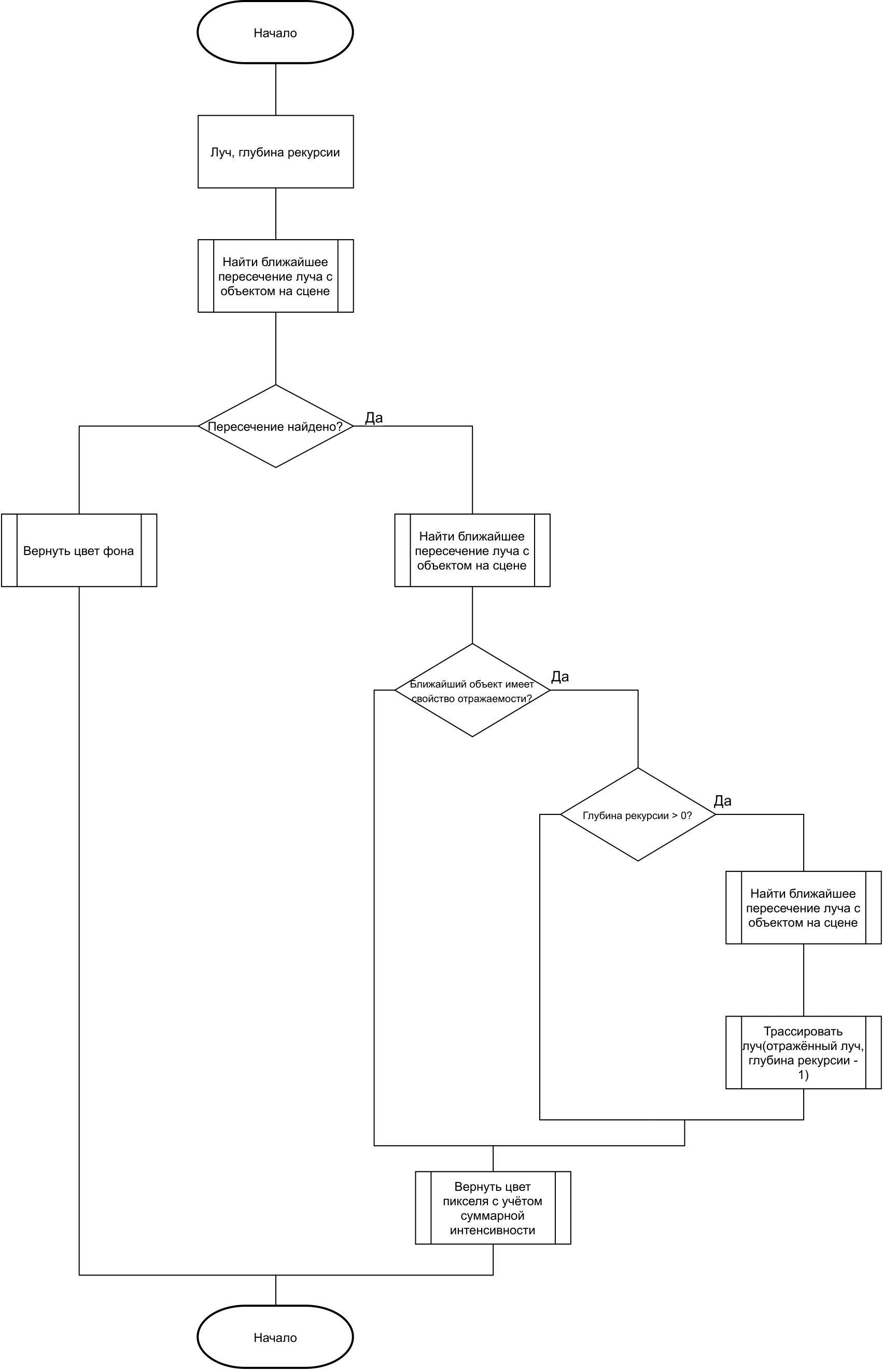


Рисунок 2.2 схема алгоритма функции «трассировать луч».

Ниже, на рисунке 2.2.1 представлена первая часть схемы алгоритма функции закраски по Фонгу.

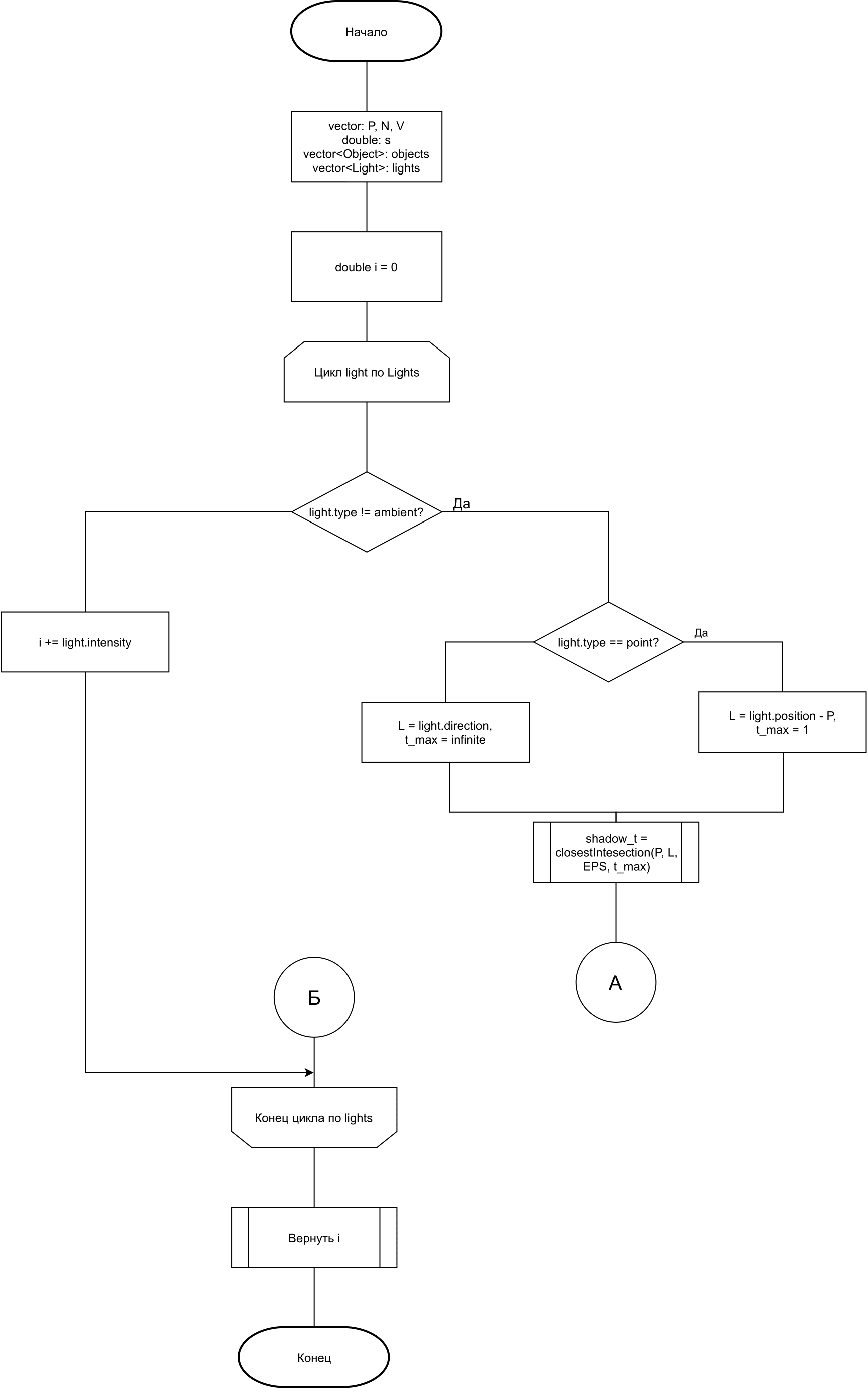


Рисунок 2.2.1 первая часть схемы алгоритма функции закраски по Фонгу.

Ниже, на рисунке 2.2.2 представлена вторая часть схемы алгоритма функции закраски по Фонгу.

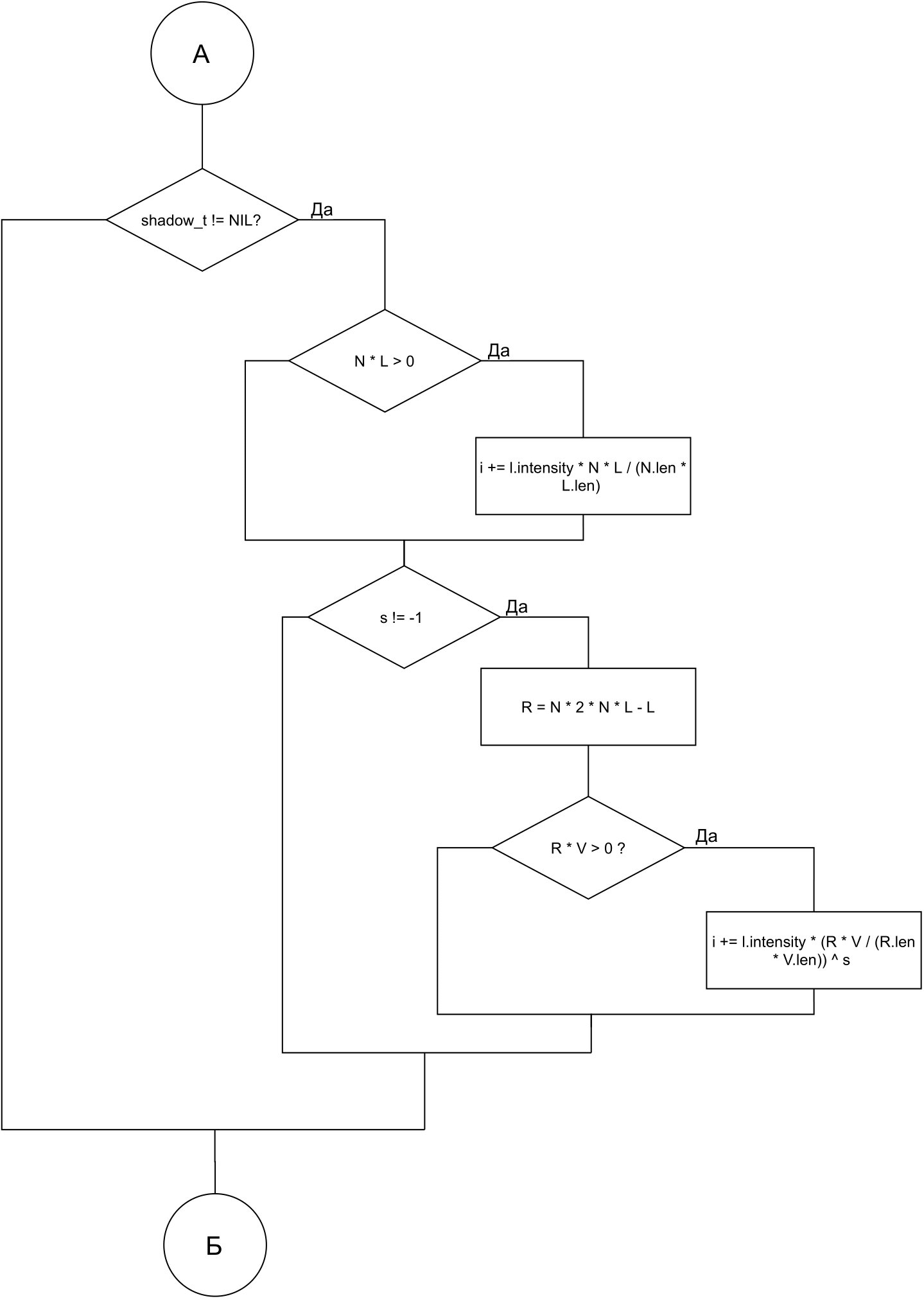


Рисунок 2.2.2 вторая часть схемы алгоритма функции закраски по Фонгу.

## 2.2 Пересечение луча с объектами сцены

Параметрическое уравнение наилучшим образом моделирует луч. Пусть точка O – начало луча, – вектор, показывающий направление луча. Любую точку 𝑃 луча можно представить как P = O + t \* , где 𝑡 — параметр, некое действительное число.

Необходимо рассмотреть объекты сцены, с которыми лучи сталкиваются. В сцене присутствуют плоскости, сферы и цилиндры. Для нахождения точки пересечения луча с произвольной поверхностью необходимо знать аналитические уравнения, определяющие эти объекты в трехмерном пространстве. Точка пересечения удовлетворяет всем уравнениям, так как принадлежит и лучу, и поверхности. Поэтому, сводя уравнения в систему и находя ее решения, можно получить координаты этой точки.

Для нахождения пересечения с объектами сцены используется скалярное произведение векторов, для которого принято следующее обозначение:

) – скалярное произведение векторов .

### 2.2.1 Пересечение луча с плоскостью

Пусть – вектор нормали, P – точка пересечения луча с плоскостью, C – начальная точка

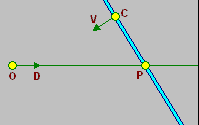


Рисунок 2.3. Пересечение луча с плоскостью

Уравнение плоскости:

Данное уравнение можно записать следующим образом:

Тогда пересечение уравнения, описывающего точки плоскости с уравнением описывающим точки луча, находится следующим образом:

Выразим отсюда t:

### 2.2.2 Пересечение луча со сферой

Пусть C – центр сферы, r – ее радиус, P – точка пересечения луча со сферой.

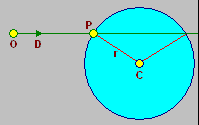


Рисунок 2.4. Пересечение луча со сферой

Тогда есть два уравнения, одно из которых описывает точки сферы, а другое – точки луча:

Точка P, в которой луч падает на сферу, является одновременно и точкой луча, и точкой на поверхности сферы, поэтому она должна удовлетворять обоим уравнениям одновременно. Получим:

Следовательно, для нахождения точки пересечения луча со сферой, необходимо решить квадратное уравнение, в котором отсутствие корней означает отсутствие пересечения, два корня – луч проходит через сферу, один корень – луч касается сферы. При получении двух корней выбирается меньший из них, он и будет расстоянием от начала луча до первого пересечения.

### 2.2.3 Пересечение луча с цилиндром

Пусть C – центр основания цилиндра, r – радиус его основания, – вектор единичной длины, определяющий ось цилиндра, P – точка пересечения луча с цилиндром.

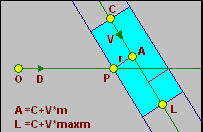


Рисунок 2.5. Пересечение луча с цилиндром

Из условия получим:

где m – скаляр, определяющий ближайшую точку на оси к точке пересечения луча и цилиндра. Вектор перпендикулярен V, что гарантирует самое близкое расстояние до оси. – это радиус цилиндра.

Тогда имеем:

Получим:

Следовательно, для нахождения точки пересечения луча с цилиндром, необходимо решить квадратное уравнение, в котором отсутствие корней означает отсутствие пересечения, два корня – луч проходит через цилиндр, один корень – луч касается цилиндра. При получении двух корней выбирается меньший из них, он и будет расстоянием от начала луча до первого пересечения.

## 2.3 Нахождение отраженного луча

Для нахождения направления отраженного луча достаточно знать направление падающего луча и нормаль к поверхности в точке падения луча.

Можно разложить на два вектора и , таких что +, где параллелен , а перпендикулярен, как изображено на рисунке 2.9.

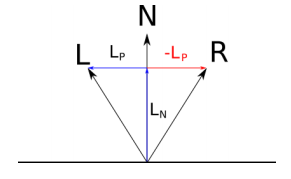


Рисунок 2.9. Разложение вектора падающего луча.

— проекция на ; по свойствам скалярного произведения и исходя из того, что = 1, длина этой проекции равна , поэтому

Отсюда:

Очевидно, что:

Подставим полученные ранее выражения и упростим, получим формулу отраженного луча:

## Вывод из конструкторской части

В данном разделе были рассмотрены схемы алгоритмов, а также алгоритмы поиска пересечений луча с объектами сцены.

# 3. Технологическая часть

В данном разделе рассмотрен выбор средств реализации и интерфейс программы, описаны основные этапы программной реализации.

## **3.1. Средства реализации**

Для решения поставленных в курсовом проекте задач был выбран язык программирования C++, поскольку:

* данный язык использовался мною при написании лабораторных работ по курсу «Компьютерная графика»;
* основной парадигмой в данном языке является ООП, что позволит разбить компоненты сцены на соответствующие классы;
* язык C++ крайне оптимизирован и очень быстр, что очень поможет при решении такой трудозатратной задачи, как трассировка лучей.

В качестве IDE был выбран “QT Creator” по следующим причинам:

* знаком с данной IDE, т.к. делал в ней лабораторные работы по курсу «Компьютерная графика»;
* позволяет очень быстро создать интерфейс приложения посредством форм.

Для организации многопоточности была использована стандартная библиотека <thread>, для организации взаимоисключений в виде мьютекса была использована стандартная библиотека <mutex>

## 3.2. Описание основных этапов реализации

В листинге 3.1 представлен код функции TraceRay. В случае, если в ходе выполнения данной функции выяснится, что луч пересекает какой-то объект – то будет возвращён цвет, которым следует закрасить пиксель. В противном случае вернётся цвет фона.

Листинг 3.1 – реализация метода TraceRay

std::vector<int> **TraceRay**(const std::vector<double>& O,

const std::vector<double>& D,

double t\_min,

double t\_max,

int depth,

const std::vector<int>& BG\_COLOR,

const std::vector<Object\*>& objects,

const std::vector<Light>& lights) {

double closest\_t = std::numeric\_limits<double>::max();

Object\* closest\_object = nullptr;

ClosestIntersection(O, D, t\_min, t\_max, *closest\_t*, *closest\_object*, objects);

if (closest\_object == nullptr) {

return BG\_COLOR;

}

std::vector<double> P = VectorSum(O, VectorMult(D, closest\_t)); // Intersection point

std::vector<double> N; // Normal of the object in the P point

if (closest\_object->object\_type == Object::type::sphere ||

closest\_object->object\_type == Object::type::sphereSouceColor) {

N = VectorDiff(P, closest\_object->*getCenter*());

N = {N[0] / VectorLength(N),

N[1] / VectorLength(N),

N[2] / VectorLength(N)};

if (closest\_object->object\_type == Object::type::sphereSouceColor) {

N = VectorMult(N, -1);

}

} else if (closest\_object->object\_type == Object::type::plane) {

N = closest\_object->*getNormal*();

if (VectorScalarMult(D, N) < 0) {

N = VectorMult(N, -1);

}

} else if (closest\_object->object\_type == Object::type::cylinder) {

N = closest\_object->*getNormal*(O, D, closest\_t);

}

std::vector<int> local\_color = VectorMult(closest\_object->*getColor*(),

ComputeLightning(P,

N,

VectorMult(D, -1),

closest\_object->*getSpecular*(),

lights, objects));

if (local\_color[0] > 255) { local\_color[0] = 255; }

if (local\_color[1] > 255) { local\_color[1] = 255; }

if (local\_color[2] > 255) { local\_color[2] = 255; }

if (local\_color[0] < 0) { local\_color[0] = 0; }

if (local\_color[1] < 0) { local\_color[1] = 0; }

if (local\_color[2] < 0) { local\_color[2] = 0; }

double r = closest\_object->*getReflective*();

double s = closest\_object->getTransparency();

if (depth <= 0 || r <= 0) {

return local\_color;

}

if (r >= 0) {

std::vector<double> R = ReflectRay(VectorMult(D, -1), N);

std::vector<int> reflected\_color = TraceRay(P, R, 1E-3, std::numeric\_limits<double>::max(),

depth - 1, BG\_COLOR,

objects, lights);

return VectorSum(VectorMult(local\_color, 1 - r), VectorMult(reflected\_color, r));

}

}

В листинге 3.2 представлен код метода ComputeLighting, который вычисляет интенсивность точки.

Листинг 3.2 – реализация метода ComputeLighting

std::vector<double> **ComputeLightning**(const std::vector<double>& P,

const std::vector<double>& N,

const std::vector<double>& V,

const double& s,

const std::vector<Light>& lights,

const std::vector<Object\*>& objects) {

std::vector<double> i = {0, 0, 0};

std::vector<double> L;

for (const auto& light : lights) {

if (light.t == Light::type::ambient) {

i = VectorSum(i, light.color\_intensity);

} else {

double t\_max;

if (light.t == Light::type::point) {

L = {light.position[0] - P[0],

light.position[1] - P[1],

light.position[2] - P[2]};

t\_max = 1;

} else {

L =light.direction;

t\_max = std::numeric\_limits<double>::max();

}

// Shadow check

double shadow\_t = std::numeric\_limits<double>::max();

Object\* closest\_shadow\_object = nullptr;

ClosestIntersection(P, L, 1E-3, t\_max, *shadow\_t*, *closest\_shadow\_object*, objects);

if (closest\_shadow\_object != nullptr) {

if (closest\_shadow\_object->object\_type != Object::type::sphereSouceColor)

continue;

}

// Diffusual

double n\_dot\_l = VectorScalarMult(N, L);

if (n\_dot\_l > 0) {

i[0] += light.color\_intensity[0] \* n\_dot\_l / (VectorLength(N) \* VectorLength(L));

i[1] += light.color\_intensity[1] \* n\_dot\_l / (VectorLength(N) \* VectorLength(L));

i[2] += light.color\_intensity[2] \* n\_dot\_l / (VectorLength(N) \* VectorLength(L));

}

// Mirroring

if (fabs(s + 1) > 1E-6) {

std::vector<double> R = VectorDiff(VectorMult(VectorMult(N, 2), VectorScalarMult(N, L)), L);

double r\_dot\_v = VectorScalarMult(R, V);

if (r\_dot\_v > 0) {

i[0] += light.color\_intensity[0] \* pow(r\_dot\_v / (VectorLength(R) \* VectorLength(V)), s);

i[1] += light.color\_intensity[1] \* pow(r\_dot\_v / (VectorLength(R) \* VectorLength(V)), s);

i[2] += light.color\_intensity[2] \* pow(r\_dot\_v / (VectorLength(R) \* VectorLength(V)), s);

}

}

}

}

if (i[0] > 255) { i[0] = 255; }

if (i[1] > 255) { i[1] = 255; }

if (i[2] > 255) { i[2] = 255; }

if (i[0] < 0) { i[0] = 0; }

if (i[1] < 0) { i[1] = 0; }

if (i[2] < 0) { i[2] = 0; }

return i;

}

## 3.3. Описание интерфейса программы

На рисунке 3.1 представлен общий интерфейс программы. Для управления сценой и сохранения изображения используется панель в правой части экрана.

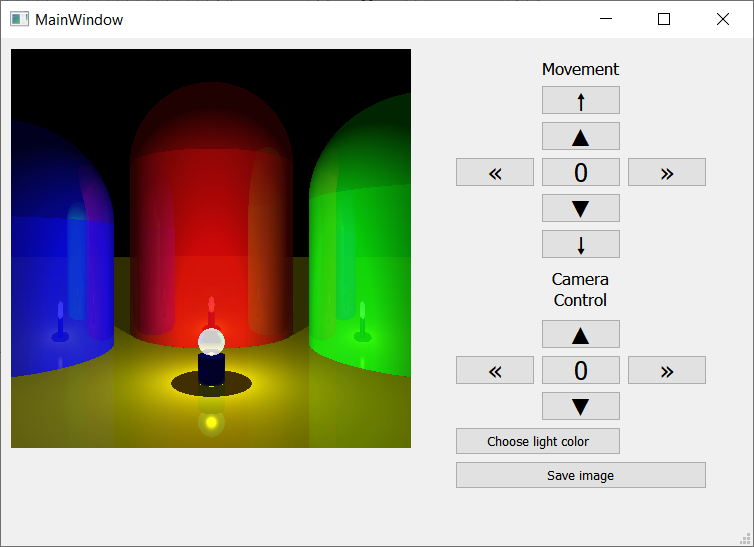


Рисунок 3.1. Общий интерфейс программы.

Блок «Movement» содержит кнопки управления и позволяет перемещать камеру по всем осям (рисунок 3.2).

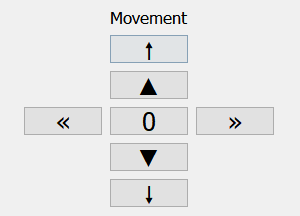


Рисунок 3.2. Блок управления «Movement».

Блок «Camera Control» содержит кнопки, позволяющие изменять угол наклона камеры

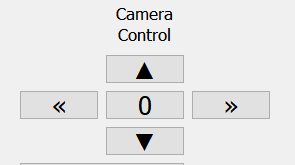


Рисунок 3.3. Блок управления «Camera Control»

Блок «Выбор цвета» позволяет изменять цвет света, излучаемого лампочкой (рисунки 3.4 и 3.5).



Рисунок 3.4. Блок «Выбор цвета».

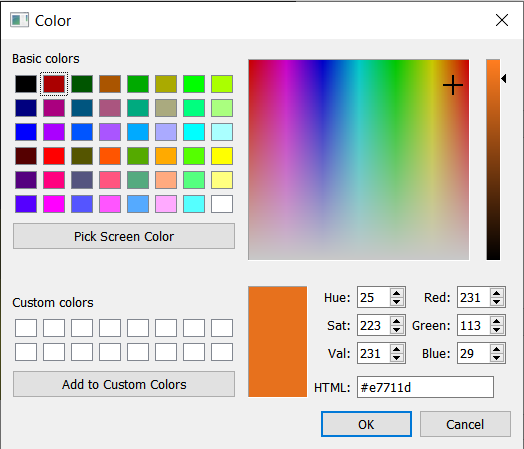


Рисунок 3.5. Меню выбора излучаемого цвета

Кнопка «Save Image» позволяет сохранять полученное изображение в любом месте на компьютере (рисунки 3.6 и 3.7).



Рисунок 3.6. Кнопка «Save Image».

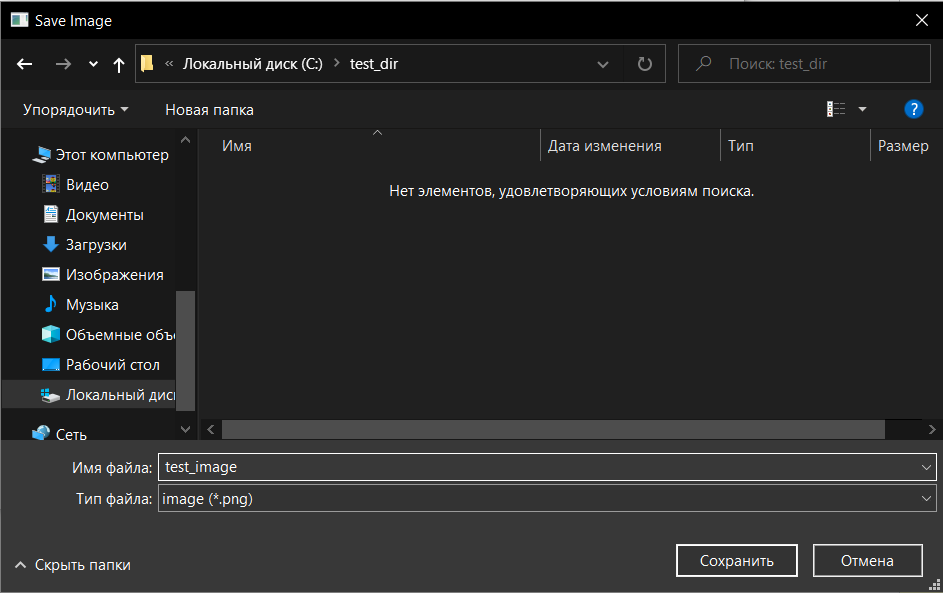


Рисунок 3.7. Меню «Save Image».

## Вывод из технологической части

В данном разделе были выбраны средства реализации, рассмотрен интерфейс программы, а также листинг основных этапов реализации.

# 4. Исследовательская часть

В данном разделе будут приведены постановка эксперимента и сравнительный анализ алгоритмов на основе экспериментальных данных. При исследовании временных характеристик разработанной программы использовался компьютер на базе 4-х ядерного процессора Intel Core i7-8550U. Для замеров времени использовалась библиотека «std::chrono».

## 4.1. Постановка эксперимента по замеру времени

Для проведения замеров времени выполнения реализации алгоритмов будет использована формула:

где t — среднее время выполнения алгоритма, N — количество замеров, T — время выполнения N замеров. Неоднократное измерение времени необходимо для получения более точного результата. Количество замеров взято равным 10. Для эксперимента используются сцена, состоящая из сфер и цилиндров, поверхности которых обладают зеркальным отражением. Время измеряется в секундах.

На гистограмме 4.1 изображена зависимость времени рендеринга сцены от количества задействованных для вычислений потоков:

Гистограмма 4.1. Сравнение зависимости скорости обработки сцены от количества используемых потоков

Как можно заметить, максимальной производительности удалость достичь при задействовании 16 потоков процессора благодаря технологии Hyper-Threading, использующейся в моём процессоре.

## Вывод из исследовательской части

В результате исследования временных затрат на рендеринг сцены было выяснено, что увеличение числа используемых потоков ведёт к значительному уменьшению времени обработки сцены.

# Заключение

Все поставленные задачи были выполнены. Целью курсовой работы являлось создание программного обеспечения, способного моделировать цвет, исходящий от RGB-лампочки, что было достигнуто.

В ходе работы были проанализированы существующие алгоритмы удаления невидимых линий и поверхностей, модели освещения, указаны их преимущества и недостатки.

Разработаны собственные и адаптированы существующие структуры данных и алгоритмы, необходимые для решения поставленной задачи.

Спроектировано и реализовано программное обеспечение, моделирующее свет, исходящий от RGB-лампочки.

# 

# Список литературы

1. Роджерс Д. Математические основы машинной графики. / Роджерс Д., Адамс Дж. – М.: Мир, 1989. – 512с.
2. Bruce Baumgart, Winged-Edge Polyhedron Representation for Computer Vision. National Computer Conference, May 1975
3. Е. А. Снижко. Компьютерная геометрия и графика [Текст], 2005. - 17 с.
4. Проблемы трассировки лучей – из будущего в реальное время. [Электронный ресурс]. – Режим доступа: <https://nvworld.ru/articles/ray-tracing/3/> (дата обращения 12.10.20)
5. RayTracing – царь света и теней, Лев Дымченко [Электронный ресурс]. – Режим доступа: <https://old.computerra.ru/206167/> (дата обращения 14.10.20)