**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 памятника архитектуры “Спасская башня”»*

Студент ИУ7-52Б _____ К.М. Искакова
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ А.В. Куров
(Подпись, дата) (И.О.Фамилия)

*2021 г.*

# Оглавление

# Введение

Компьютерная графика – это совокупность методов и способов преобразования информации в графическую форму и из графической формы в ЭВМ.

В современном мире компьютерная графика является неотъемлемой частью человеческой жизни. Область применения компьютерной графики не ограничивается художественными эффектами. Она используются во всех отраслях науки, техники, медицины, в коммерческой деятельности, где используются построенные с помощью компьютера схемы, графики, диаграммы, предназначенные для наглядного отображения информации. Конструкторы, разрабатывая новые модели, используют трехмерные графические объекты, чтобы представить окончательный вид изделия. С помощью программных редакторов архитекторы проектируют здания, а затем просматривают их объемное изображение, что позволяет дать предварительные оценки будущего сооружения. Вследствие этого перед людьми, создающими трехмерные сцены, встает задача создания реалистичных изображений, которые будут учитывать оптические явления такие, как отражение, преломление и рассеивание света.

Существует множество алгоритмов компьютерной графики, которые решают данную задачу. Однако такие алгоритмы являются ресурсозатратными. Для получения более качественного изображения требуется большое количество времени и памяти.

Целью курсового проекта является разработка программы, моделирующей памятник архитектуры «Спасская башня» и создания собственных сооружений, состоящих из сфер, цилиндров, конусов, параллелепипедов, треугольных и четырехугольных пирамид.

В рамках реализации проекта должны быть решены следующие задачи:

- Изучение и анализ алгоритмов компьютерной графики, использующихся для создания реалистичной модели взаимно перекрывающихся объектов, и выбор наиболее подходящего для решения поставленной задачи.
- Проектирование архитектуры программного обеспечения.
- Реализация выбранных алгоритмов и структур данных.
- Разработка программного обеспечения, которое позволит отобразить трехмерную сцену и визуализировать памятник архитектуры.
- Проведение исследования на основе разработанной программы

Итогом работы является программное обеспечение, демонстрирующее визуализацию памятника архитектуры «Спасская башня». В программе предусмотрена возможность поворота камеры, приближения и отдаления объектов сцены, создания собственных сооружений, состоящих из сфер, цилиндров, конусов, параллелепипедов, треугольных и четырехугольных пирамид.

# 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 памятник архитектуры «Спасская башня» с учетом теней, а также предоставить возможность моделирования других сооружении, состоящих из сфер, цилиндров, конусов, параллелепипедов, треугольных и четырехугольных пирамид. Данная программа является инструментом для проектирования конструкции и может быть использована архитекторами, конструкторами и дизайнерами.

## 1.2. Формализация объектов синтезируемой сцены

Существует несколько видов геометрических моделей:

- каркасная модель;
- поверхностная модель;
- объемная модель.

Объекты сцены наилучшим образом описываются с помощью поверхностной модели, так как каркасные модели не обладают достаточной реалистичностью, а в объемной модели добавляется информация о том, где расположен материал, что в данной работе не нужно. Поверхности сфер, конусов и цилиндров описываются аналитически. Для представления треугольных и четырехугольных пирамид используется такой геометрический примитив как треугольник, т.е. полигональная аппроксимация. Параллелепипеды задаются координатами двух своих

вершин: той, у которой все три координаты минимальны, и той, у которой все три координаты максимальны.

Сцена состоит из следующих объектов.

- Точечных источников света – представляют собой фиксированную точку в пространстве, называемой его позицией, из которой свет испускается равномерно во всех направлениях. Точечный источник полностью характеризуется его позицией и яркостью.
- Направленный источник света – представляет собой вектор направления света, предполагается, что источник расположен в бесконечности. Направленный источник света является аппроксимацией солнца.
- Сооружения, состоящие из сфер, цилиндров, конусов, параллелепипедов, треугольных и четырехугольных пирамид.
- Плоскости основания.

## 1.3. Критерии выбора алгоритма

Критериями выбора алгоритма служат возможность работы с телами вращения и демонстрацией зеркального отражения.

## 1.4. Алгоритмы удаления невидимых линий и поверхностей

Для выбора подходящего алгоритма построения изображения, необходимо провести обзор известных алгоритмов и осуществить выбор наиболее подходящего для реализации поставленной задачи.

### Алгоритм Робертса

Алгоритм Робертса представляет собой первое известное решение задачи об удалении невидимых линий. Этот алгоритм работает в объектном

пространстве. Сначала удаляются из каждого тела те ребра или грани, которые перекрываются 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, перекрываются этими телами.

Алгоритм Робертса не может быть применим для решения поставленной задачи, т.к. работает с выпуклыми телами, т.е. с многогранниками, и не моделирует оптические свойства объекта.

**Алгоритм Варнока**

Алгоритм Варнока основывается на рекурсивном разбиении экрана. Алгоритм работает в пространстве изображения и анализирует область на экране дисплея (окно) на наличие в них видимых элементов. Если в окне нет изображения, то оно просто закрашивается фоном. Если же в окне имеется элемент, то проверяется, достаточно ли он прост для визуализации. Если объект сложный, то окно разбивается на более мелкие, для каждого из которых выполняется тест на отсутствие или простоту изображения. Рекурсивный процесс разбиения может продолжаться до тех пор, пока не будет достигнут предел разрешения экрана.

Алгоритм Варнока не может быть применим для решения поставленной задачи, т.к. не моделирует оптические свойства объекта.

**Алгоритм, использующий z-буфер**

Данный алгоритм удаления невидимых поверхностей является одним из самых простых и широко используемых. Этот алгоритм работает в пространстве изображения. Его идея заключается в использовании двух буферов: буфера кадра и буфера глубины, также называемого Z-буфером. Буфер кадра используется для хранения интенсивности каждого пикселя в пространстве изображения. В буфере глубины запоминается значение координаты Z (глубины) каждого видимого пикселя в пространстве

изображения. В ходе работы алгоритма значение глубины каждого нового пикселя, заносимого в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в Z-буфер. Если это сравнение показывает, что новый пиксель расположен ближе к наблюдателю, чем пиксель, уже находящийся в буфере кадра, то новый пиксель заносится в буфер кадра и производится корректировка Z-буфера: в него заносится глубина нового пикселя. Если же значение глубины нового пикселя меньше, чем хранящееся в буфере, то осуществляется переход к следующей точке. На рисунке 1.1 представлен пример работы z-буфера.

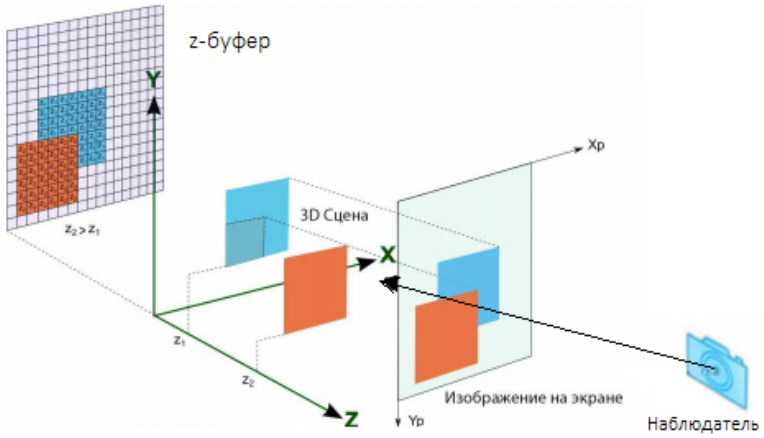


Рисунок 1.1. Работа алгоритма z-буфера.

Основными достоинствами данного алгоритма являются простота его реализации, корректная обработка случаев взаимных пересечений объектов, линейная зависимость трудоемкости от числа объектов на сцене, а также отсутствие необходимости предварительной сортировки объектов по глубине, то есть они могут обрабатываться в произвольном порядке.

К недостаткам данного алгоритма относят необходимость выделения памяти под два буфера, каждый из которых имеет размер, равный количеству пикселей на экране.

Благодаря модификации алгоритма z-буфера, можно учитывать тени и прозрачность. Однако алгоритм не может визуализировать эффект зеркального отражения, которое попадает на другие поверхности.

**Алгоритм обратной трассировки лучей**

Алгоритм является улучшенной модификацией алгоритма прямой трассировки. Из камеры испускаются лучи, проходящие через каждый пиксель вглубь сцены, затем идет поиск пересечений первичного луча с объектами сцены, как показано на рисунке 1.2, в случае обнаружения пересечения, рассчитывается интенсивность пикселя, в зависимости от положения источника света, при отсутствии пересечения, пиксель закрашивается цветом фона.

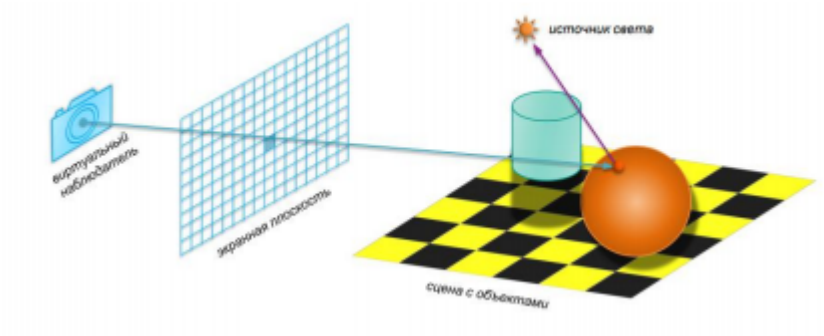


Рисунок 1.2. Работа алгоритма обратной трассировки лучей.

К достоинствам данного алгоритма можно отнести возможность получения изображения гладких объектов без аппроксимации их примитивами (например, треугольниками). Трассировка лучей позволяет визуализировать тени, эффекты прозрачности, преломления, отражения. Вычислительная сложность метода линейно зависит от сложности сцены. Полученное изображение получается очень реалистичным.

Серьёзным недостатком алгоритма трассирования является производительность. Для получения изображения необходимо создавать большое количество лучей, проходящих через сцену и отражаемых от объекта. Это приводит к существенному снижению скорости работы программы. Однако для ускорения работы программы лучи можно трассировать одновременно, поскольку каждый луч, исходящий из камеры, независим от всех остальных.

## 1.5. Выбор модели освещения

Физические модели, которые не учитывают перенос света между поверхностями (не используют вторичное освещение), называются локальными. В противном случае модели называются глобальными или моделями глобального освещения.

Для решения поставленной задачи нет необходимости использовать глобальные модели освещения.

В данном подразделе будут рассмотрены локальные модели освещения Ламберта и Фонга.

**Модель Ламберта**

Модель Ламберта является одной из самых простых моделей освещения. Модель Ламберта моделирует идеальное диффузное освещение. Считается, что свет, падающий в точку, одинакового рассеивается по всем направлениям полупространства. Сила освещения зависит исключительно от угла α между вектором падения света L и вектором нормали N, как показано на рисунке 1.3. Максимальная сила света будет при перпендикулярном падении света на поверхность и будет убывать с увеличением угла α.

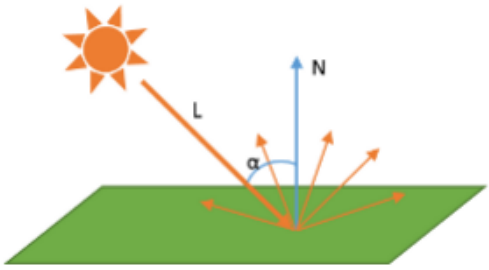


Рисунок 1.3. Модель освещения Ламберта.

Модель проста в реализации, но не позволяет передавать блики на телах сцены.

**Модель Фонга**

Основная идея модели Фонга заключается в предположении, что освещенность каждой точки тела разлагается на 3 компоненты:

1. фоновое освещение (ambient);

2. рассеянный свет (diffuse);

3. бликовая составляющая (specular).

Свойства источника определяют мощность излучения для каждой из этих компонент, а свойства материала поверхности определяют ее способность воспринимать каждый вид освещения.

Фоновое освещение присутствует в любом уголке сцены и никак не зависит от каких-либо источников света, поэтому для упрощения расчетов оно задается константой. Диффузное освещение рассчитывается аналогично модели Ламберта. Отраженная составляющая освещенности (блики) в точке зависит от того, насколько близки направления вектора, направленного на наблюдателя (вектор V на рисунке 1.4), и отраженного луча (вектор R на рисунке 1.4).

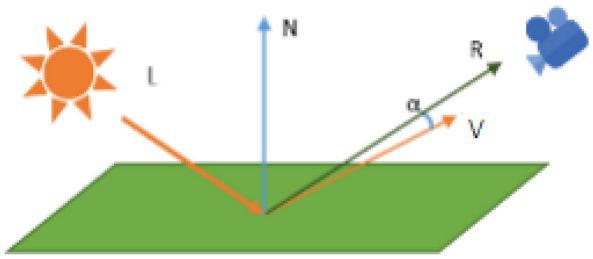


Рисунок 1.4. Получение бликов в модели освещения Фонга.

$$I = K_a I_a + K_d(\vec{N}, \vec{L}) + K_s(\vec{R}, \vec{V})^p$$, где

$\vec{N}$ – вектор нормали к поверхности в точке,

$\vec{L}$ – падающий луч (направление на источник света),

$\vec{R}$ – отраженный луч,

$\vec{V}$ – вектор, направленный к наблюдателю,

$K_a$ – коэффициент фонового освещения,

$K_d$ – коэффициент диффузного освещения,

$K_s$ – коэффициент зеркального освещения.

$p$ – степень, аппроксимирующая пространственное распределение зеркально отраженного света.

Все векторы являются единичными.

Модель Фонга улучшает визуальные качества сцены, по сравнению с моделью Ламберта, добавляя в нее блики (см. рис 1.5).

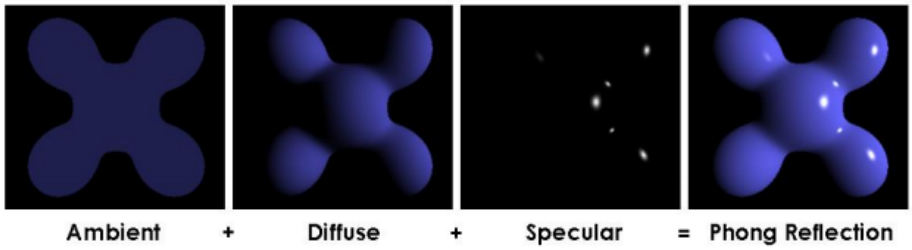


Рисунок 1.5. Модель освещения Фонга.

## Вывод

В результате анализа алгоритмов, в соответствии с поставленной задачей, были выбраны следующие алгоритмы.

1. Алгоритм обратной трассировки лучей выбран для удаления невидимых линий и поверхностей. Он позволяет получить изображение высокого качества, а также предоставляет возможность работы с телами вращения.

2. Модель Фонга выбрана для расчета интенсивности в точке. Она позволяет учитывать матовые и блестящие поверхности.

# 2. Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов для обратной трассировки лучей и расчета освещенности в соответствии с моделью Фонга, поиск пересечение луча с объектами сцены, диаграмма классов.

Программа должна обладать следующей функциональностью:

1. Визуализировать трехмерную сцену, состоящую из объектов, представленных в пункте 1.2, в режиме реального времени.
2. Предоставлять в интерфейсе возможность пользователю выполнять следующие действия:
   2.1. Добавлять и удалять объекты сцены.
   2.2. Изменять параметры объектов сцены.
   2.3. Изменять положение камеры и осуществлять её поворот.
   2.4. Добавлять точечные источники света.
   2.5. Изменять параметры источников света.

## 2.1. Схемы алгоритмов

На рисунке 2.1 представлена схема алгоритма трассировки лучей.

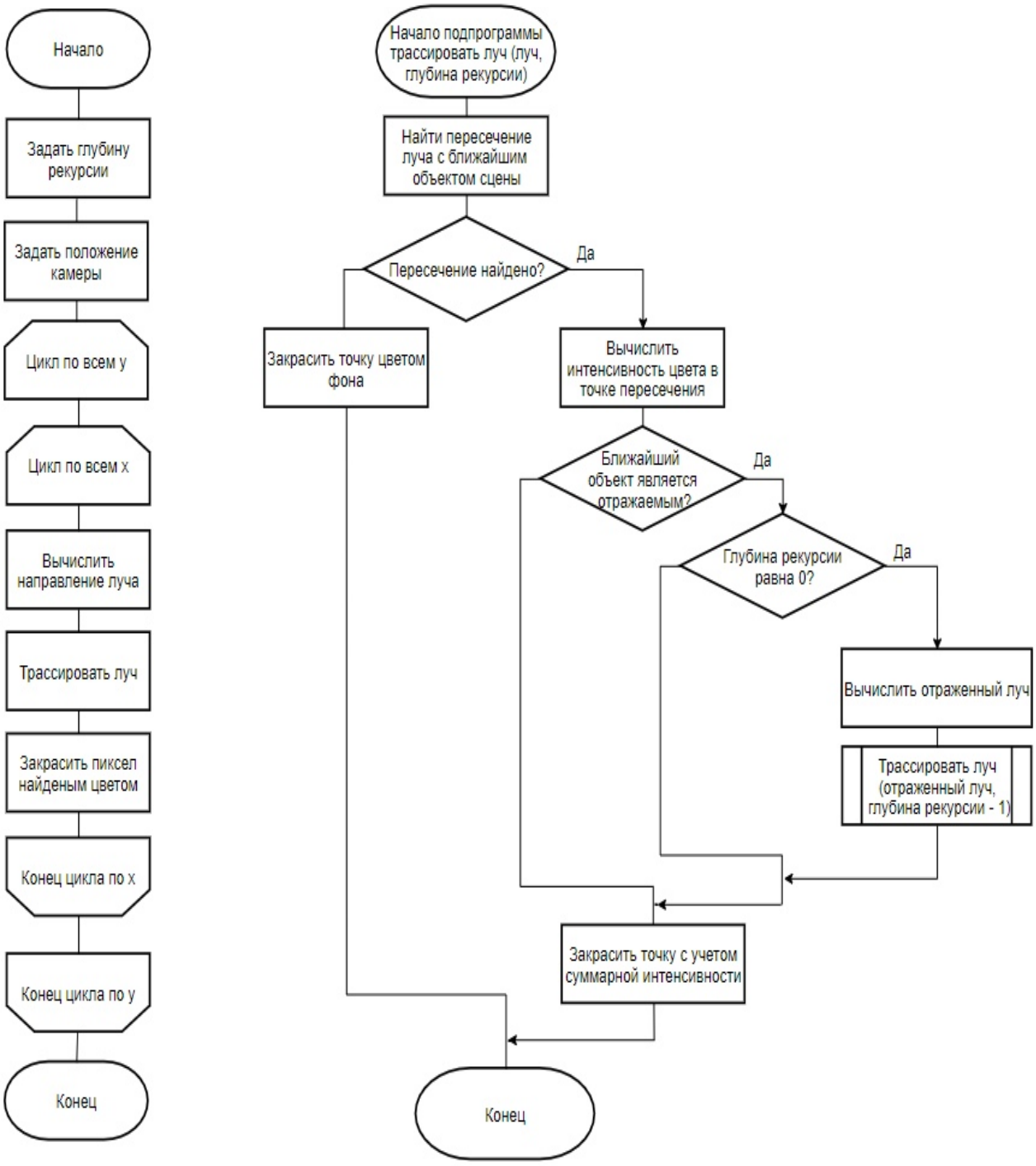


Рисунок 2.1. Схема алгоритма трассировки лучей.

На рисунке 2.2 представлена схема алгоритма расчета освещенности в соответствии с моделью Фонга.

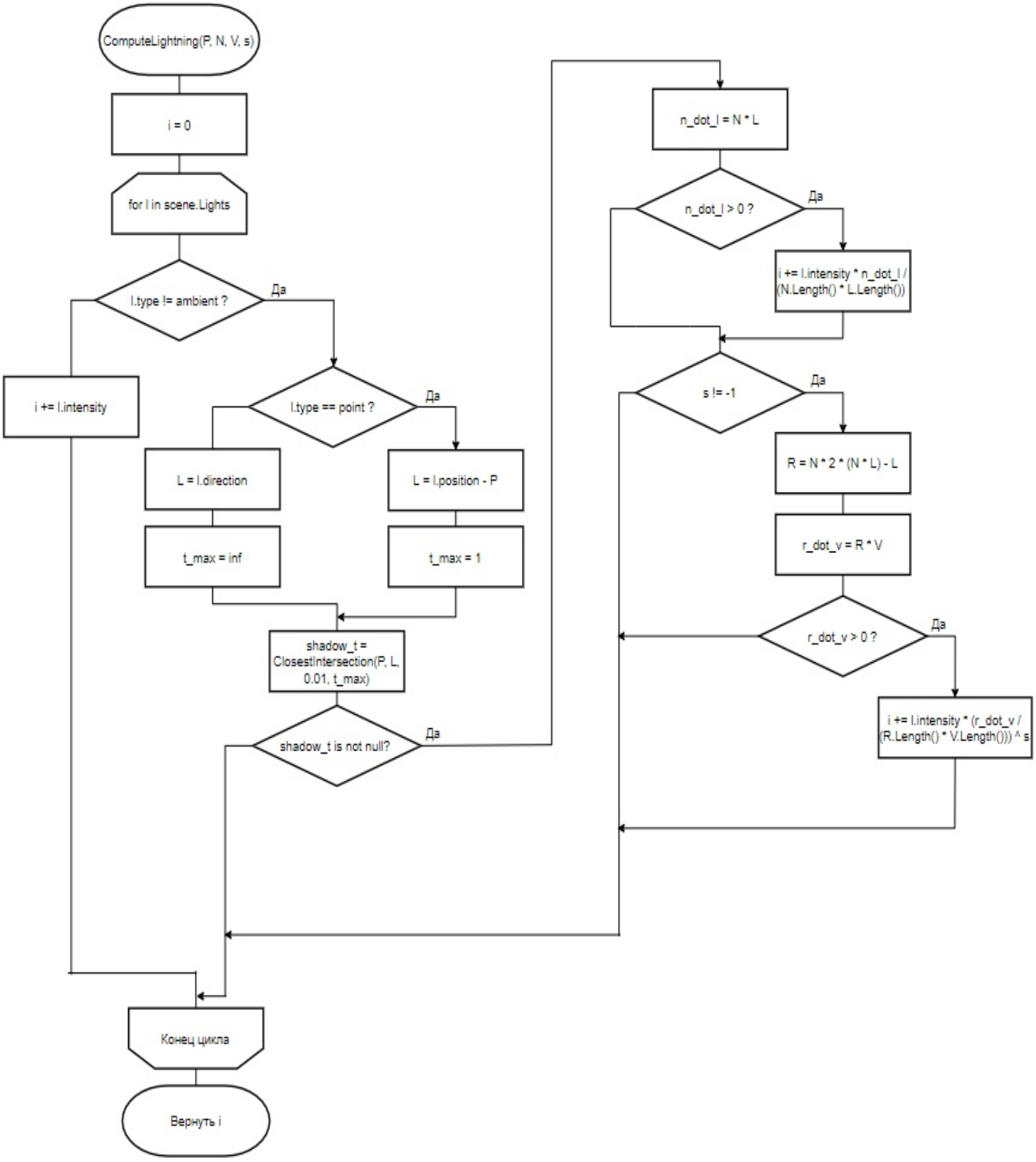


Рисунок 2.2. Схема алгоритма расчета освещенности в соответствии с моделью Фонга.

## 2.2 Пересечение луча с объектами сцены

Наилучшим способом представления лучей является использование параметрического уравнения. Пусть $O$ – начало луча, $\vec{D}$ – вектор, показывающий направление луча. Любую точку $P$ луча можно представить как $P = O + t\vec{D}$, где $t$ — произвольное действительное число.

Необходимо рассмотреть объекты сцены, с которыми лучи сталкиваются. В сцене присутствуют сферы, цилиндры, конусы, параллелепипеды, треугольные и четырехугольные пирамиды, для представления которых используется такой геометрический примитив как треугольник, и плоскости. Для нахождения точки пересечения луча с произвольной поверхностью необходимо знать аналитические уравнения, определяющие оба эти объекта 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:

$dot(\vec{a}, \vec{b})$ – скалярное произведение векторов $\vec{a}$ и $\vec{b}$.

### 2.2.1 Пересечение луча со сферой

Пусть C – центр сферы, r – ее радиус, P – точка пересечения луча со сферой (см. рис 2.3).

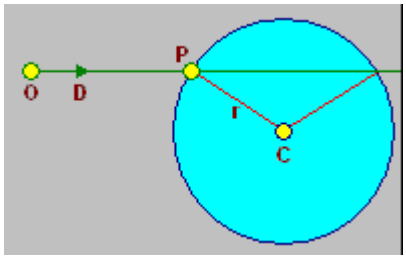


Рисунок 2.3. Пересечение луча со сферой

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

$$dot(P - C, P - C) = r^2 \quad (1.1)$$

$$P = O + t\vec{D} \quad (1.2)$$

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:

$$dot\left(\overrightarrow{CO} + t\vec{D}, \overrightarrow{CO} + t\vec{D}\right) = r^2 \quad (1.3)$$

$$t^2 dot\left(\vec{D}, \vec{D}\right) + 2 * t * dot\left(\overrightarrow{CO}, \vec{D}\right) + dot\left(\overrightarrow{CO}, \overrightarrow{CO}\right) - r^2 = 0 \quad (1.4)$$

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.

### 2.2.2 Пересечение луча с плоскостью

Пусть $\vec{V}$ – вектор нормали, P – точка пересечения луча с плоскостью, C – начальная точка (см. рис 2.4).

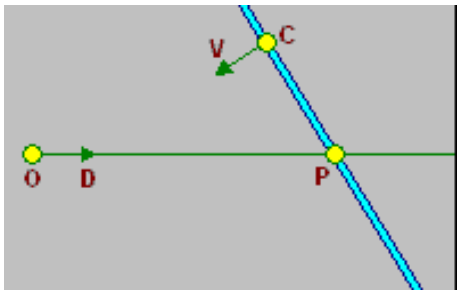


Рисунок 2.4. Пересечение луча с плоскостью

Уравнение плоскости:

$$Ax + By + Cz = 0 \quad (2.1)$$

Данное уравнение можно записать следующим образом:

$$dot\left(\vec{V}, P - C\right) = 0 \ (2.2)$$

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

$$dot\left(\vec{V}, O + t\vec{D}\right) = 0 \quad (2.3)$$

$$dot\left(\vec{V}, O\right) + t * dot\left(\vec{V}, \vec{D}\right) - dot\left(\vec{V}, \vec{C}\right) = 0 \quad (2.4)$$

Выразим отсюда t:

$$t = dot\left(\vec{V}, \vec{C}\right) - dot\left(\vec{V}, O\right)/dot\left(\vec{V}, \vec{D}\right) \ (2.5)$$

### 2.2.3 Пересечение луча с цилиндром

Пусть C – центр основания цилиндра, r – радиус его основания, $\vec{V}$ – вектор единичной длины, определяющий ось цилиндра, maxm – высота цилиндра, P – точка пересечения луча с цилиндром (см. рис 2.5).

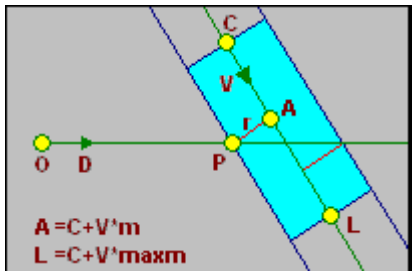


Рисунок 2.5. Пересечение луча с цилиндром

Из условия получим:

$$A = C + m\vec{V} \quad (3.1)$$

$$dot\left(P - A, \vec{V}\right) = 0 \quad (3.2)$$

$$len(P - A) = r \quad (3.3)$$

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

Тогда имеем:

$$dot\left(P - C - \vec{V} * m, \vec{V}\right) = 0 \quad (3.4)$$

$$dot\left(P - C, \vec{V}\right) = m * dot\left(\vec{V}, \vec{V}\right) = m, \text{т. к } \vec{V} - \text{единичный вектор} \quad (3.5)$$

$$m = dot\left(\vec{D} * t + \overrightarrow{CO}, \vec{V}\right) \quad (3.6)$$

$$m = dot\left(\vec{D}, \vec{V}t\right) + dot\left(\overrightarrow{CO}, \vec{V}\right) \quad (3.7)$$

$$len\left(P - C - \vec{V} * m\right) = r \quad (3.8)$$

Получим:

$$t^2\left(dot(\vec{D},\vec{D}) - dot(\vec{D},\vec{V})^2\right) + 2 * t * \left(dot(\vec{D},\overrightarrow{CO}) - dot(\vec{D},\vec{V}) * dot(\overrightarrow{CO},\vec{V})\right)$$
$$+\ dot(\overrightarrow{CO},\overrightarrow{CO}) - dot(\overrightarrow{CO},\vec{V})^2 - r^2 = 0 \quad (3.9)$$

Следовательно, для нахождения точки пересечения луча с боковой поверхностью цилиндра, 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.

Для того чтобы найти точки пересечения с основаниями цилиндра необходимо рассмотреть пересечения луча и пары плоскостей $\left(C, -\vec{V}\right)$ и $\left(C + \vec{V} * maxm, \vec{V}\right)$, ограниченные радиусом цилиндра.

### 2.2.4 Пересечение луча с конусом

Пусть C – вершина конуса, $\vec{V}$ – вектор единичной длины, определяющий ось конуса, k – тангенс половины угла конуса, minm и maxm определяют границы конуса, P – точка пересечения луча с цилиндром (см. рис 2.6).

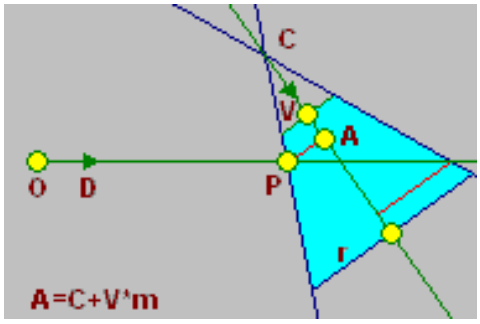


Рисунок 2.6. Пересечение луча с конуса

Из условия получим:

$$A = C + m\vec{V} \quad (4.1)$$

$$dot\left(P - A, \vec{V}\right) = 0 \quad (4.2)$$

$$len(P - A)/m = k \quad (4.3)$$

где m – скаляр, определяющий ближайшую точку на оси к точке пересечения луча и боковой поверхности конуса. Вектор $P - A$ перпендикулярен V, что гарантирует самое близкое расстояние до оси.

Тогда имеем:

$$m = dot\left(\vec{D}, \vec{V}t\right) + dot\left(\overrightarrow{CO}, \vec{V}\right) \quad (4.4)$$

$$len\left(P - C - \vec{V} * m\right) = m * k \quad (4.5)$$

Получим:

$$t^2 \left(dot\left(\vec{D}, \vec{D}\right) - (1 + k + k) * dot\left(\vec{D}, \vec{V}\right)^2\right) + 2 * t$$
$$* \left(dot\left(\vec{D}, \overrightarrow{CO}\right) - (1 + k + k) * dot\left(\vec{D}, \vec{V}\right) * dot\left(\overrightarrow{CO}, \vec{V}\right)\right)$$
$$+ \ dot\left(\overrightarrow{CO}, \overrightarrow{CO}\right) - (1 + k + k) * dot\left(\overrightarrow{CO}, \vec{V}\right)^2 - r^2 = 0 \quad (4.6)$$

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

Для того чтобы найти точку пересечения с основанием конуса необходимо рассмотреть пересечения луча и плоскости $\left(C + \vec{V} * maxm, \vec{V}\right)$.

### 2.2.5 Пересечение луча с параллелепипедом.

Параллелепипед задаётся координатами двух своих вершин: той, у которой все три координаты минимальны, и той, у которой все три координаты максимальны. Таким образом, получается шесть плоскостей, ограничивающих параллелепипед (см. рис 2.7).

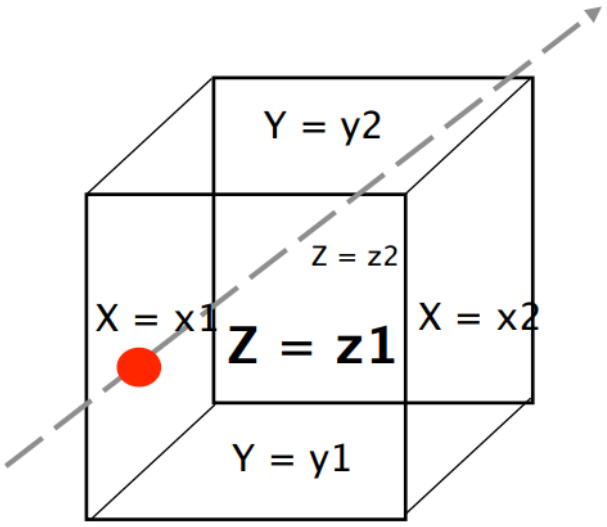


Рисунок 2.7. Пересечение луча с параллелепипедом.

Возьмем пару плоскостей, параллельных плоскости yz: x = x1 и x = x2. Пусть $\vec{D}$ – вектор направления луча.

Если координата x вектора $\vec{D}$ равна 0, то заданный луч параллелен этим плоскостям и, если x0 < x1 или x0 > x1, то он не пересекает рассматриваемый прямоугольный параллелепипед. Если же D.x не равно 0, то вычисляем отношения:

$$t1x = (x1 - x0)/D.x \quad (5.1)$$

$$t2x = (x2 - x0)/D.x \quad (5.2)$$

Можно считать, что найденные величины связаны неравенством t1x < t2x.

Пусть tn = t1x , tf = t2x. Считая, что D.y не равно 0, и рассматривая вторую пару плоскостей, несущих грани заданного параллелепипеда, y = y1 и y = y2, находим величины:

$$t1y = (y1 - y0)/D.y \quad (5.3)$$

$$t2y = (y2 - y0)/D.y \quad (5.4)$$

Если t1y > tn, тогда пусть tn = t1y. Если t2y < tf, тогда пусть tf = t2y.

При tn > tf или при tf < 0 заданный луч проходит мимо прямоугольного параллелепипеда.

Считая, что D.z не равно 0, и рассматривая вторую пару плоскостей, несущих грани заданного параллелепипеда, z = z1 и z = z2, находим величины:

$$t1z = (z1 - z0)/D.z \quad (5.5)$$

$$t2z = (z2 - z0)/D.z \quad (5.6)$$

и повторяем предыдущие сравнения.

Если в итоге всех проведенных операций мы получим, что 0 < tn < tf или 0 < tf, то заданный луч пересечет исходный параллелепипед со сторонами, параллельными координатным осям.

Если начальная точка луча лежит вне параллелепипеда, то расстояние от начала луча до точки его входа в параллелепипед равно tn, а до точки выхода луча из параллелепипеда tf.

### 2.2.6 Пересечение луча с треугольником.

Для нахождения пересечения луча с треугольником использовался алгоритм Моллера — Трумбора, который использует только вершины треугольника и не требует предварительное вычисление уравнения плоскости содержащей треугольник. Пусть P — луч, $\vec{V}$ — его направление, v0,v1,v2 — вершины треугольника, z — точка пересечения, t —расстояние от

точки вылета луча до точки пересечения луча и треугольника, u,v,t1 — барицентрические координаты, $cross(\vec{a}, \vec{b})$ — векторное произведение.

Барицентрические координаты представляют собой отношения площадей маленьких треугольников к большому треугольнику (см. рис 2.8).

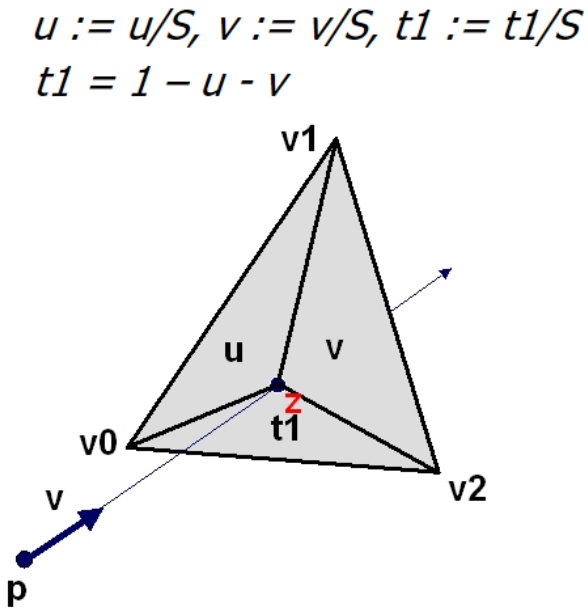


Рисунок 2.8. Пересечение луча с треугольником.

Имея 3 точки на плоскости, можно выразить любую другую точку через ее барицентричечкие координаты. Первое уравнение берется из определения барицентрических координат, выражая точку пересечения z. С другой стороны, эта же точка z лежит на прямой. Второе уравнение — параметрическое уравнение прямой. Прировняв правые части уравнений 1 и 2 получаем третье уравнение, которое, по сути, является системой 3-х уравнений (p,v,v1,v2,v3 - векторы) с 3-мя неизвестными (u,v,t).

$$z(u,v) = (1-u-v) * v1 + u * v2 + v * v0 \quad (6.1)$$

$$z(t) = p + t * d \quad (6.2)$$

$$p + t * d \; = (1-u-v) * v1 + u * v2 + v * v0 \quad (6.3)$$

Проведя алгебраические преобразования, получим ответ в следующем виде:

$$E1 = v1 - v0 \quad (6.4)$$

$$E2 = v2 - v0 \quad (6.5)$$

$$t = p - v0 \quad (6.7)$$

$$P = cross(D, E2) \quad (6.8)$$

$$Q = cross(T, E1) \quad (6.9)$$

$$D = v \quad (7.0)$$

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{dot(P, E1)} * \begin{bmatrix} dot(Q, E2) \\ dot(P, T) \\ dot(Q, D) \end{bmatrix} \quad (7.1)$$

### 2.3 Нахождение отраженного луча

Для нахождения направления отраженного луча достаточно знать направление падающего луча $\vec{L}$ и нормаль к поверхности $\vec{N}$ в точке падения луча.

Можно разложить $\vec{L}$ на два вектора $\vec{L}p$ и $\vec{L}n$, таких что $\vec{L} = \vec{L}p + \vec{L}n$, где $\vec{L}n$ параллелен $\vec{N}$, а $\vec{L}p$ перпендикулярен, как изображено на рисунке 2.9.

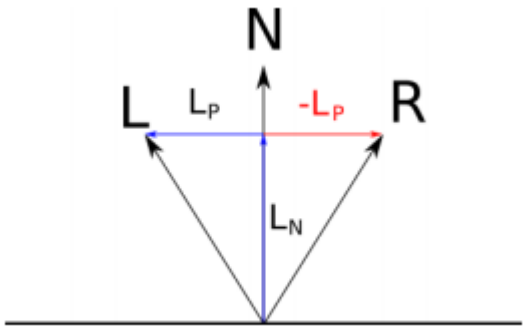


Рисунок 2.9. Разложение вектора падающего луча.

$\vec{L}n$ — проекция $\vec{L}$ на $\vec{N}$; по свойствам скалярного произведения и исходя из того, что $|\vec{N}|$ = 1, длина этой проекции равна $(\vec{N},\vec{L})$, поэтому

$$\vec{L}n = \vec{N}(\vec{N},\vec{L}) \quad (8.1)$$

Отсюда:

$$\vec{L}p = \vec{L} - \vec{L}n = \vec{L} - \vec{N}(\vec{N},\vec{L}) \quad (8.2)$$

Очевидно, что:

$$\vec{R} = \vec{L}n - \vec{L}p \quad (8.3)$$

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

$$\vec{R} = 2\vec{N}(\vec{N},\vec{L}) - \vec{L} \quad (8.4)$$

## 2.4 Диаграмма классов

На рисунке 3 приведена диаграмма классов.

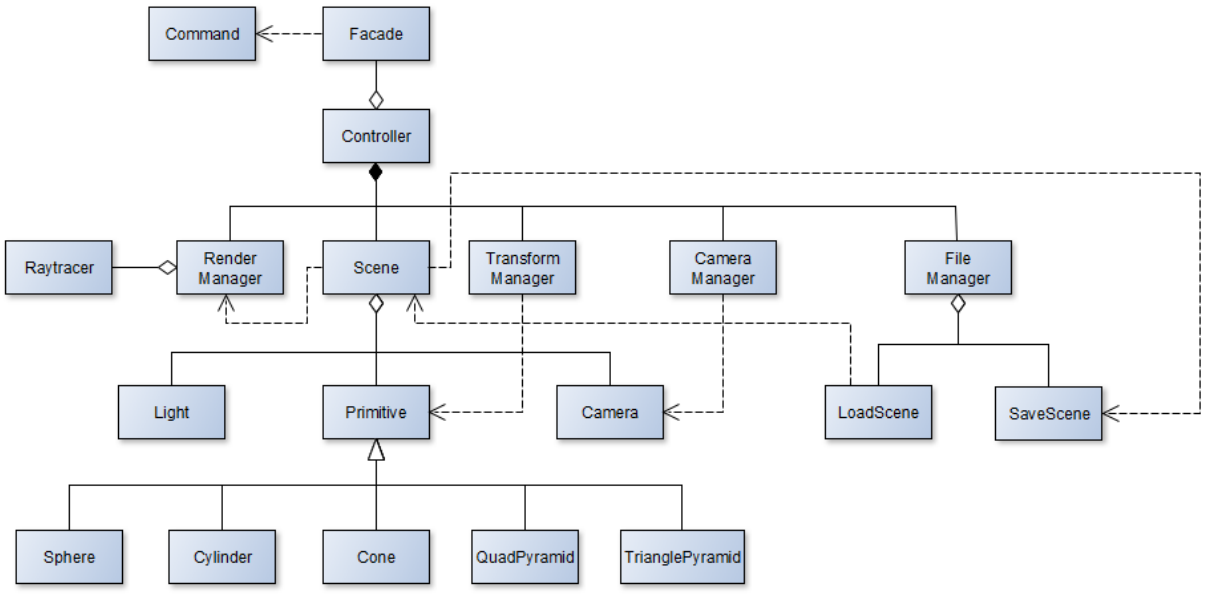


Рисунок 3. Схема классов программы.

Разработанная программа состоит из следующих классов:

- Классы объектов
    - Primitive – базовый класс примитивов;
    - Sphere – класс сферы с возможностью задания радиуса и центра сферы.
    - Cylinder – класс цилиндра с возможностью задания центра основания, радиуса, направление оси и высоты цилиндра.
    - Cone – класс конуса с возможностью задания вершины конуса, направление оси, угла вращения и высоты конуса.
    - Parallelepiped – класс параллелепипеда, который задается координатами двух своих вершин: той, у которой все три координаты минимальны, и той, у которой все три координаты максимальны.
    - TrianglePyramid – класс треугольной пирамиды с возможностью задания вершины пирамиды и точек основания.
    - QuadPyramid – класс четырехугольной пирамиды с возможностью задания вершины пирамиды и точек основания.
- Вспомогательные классы сцены
    - Camera – класс камеры с возможностью перемещения по сцене;
    - Light – класс источника освещения с возможностью перемещения по сцене и изменения интенсивности.
- Классы интерфейса
    - Facade – класс, который предоставляет интерфейс работы системы.
    - Controller – класс для взаимодействия управляющих классов с классами интерфейса.
- Класс визуализации сцены: Raytracer.

**Вывод**

В данном разделе были рассмотрены схемы алгоритмов, поиск пересечение луча с объектами сцены, диаграмма классов.

# 3. Технологическая часть

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

## 3.1. Средства реализации

Для написания курсового проекта в качестве языка программирования был выбран C# т.к:

- во время занятий по компьютерной графике было произведено ознакомление с данным языком программирования, что сократит время написания программы;
- этот язык поддерживает объектно-ориентированную модель разработки, что позволяет четко структурировать программу и легко модифицировать отдельные ее компоненты независимо от других;
- язык C# позволяет эффективно использовать ресурсы системы благодаря широкому набору функций и классов из стандартной библиотеки.

В качестве среды разработки была выбрана «Visual Studio 2019» по следующим причинам:

- предоставляет умную проверку кода, быстрое выявление ошибок и оперативное исправление, одновременно с автоматическим рефакторингом кода;
- обеспечивает работу с Windows Forms – интерфейсом, который упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обертки для существующего Win32 API в управляемом коде.

Для организации распараллеливания алгоритма трассировки лучей использовалось пространство имен «System.Threading», которое содержит в себе классы, поддерживающие многопоточное программирование.

### 3.2. Описание основных этапов реализации

На рисунке 3.1 представлен код метода TraceRay, который вычисляет пересечение луча с каждым объектом. Если оно есть, то возвращает цвет объекта в ближайшей точке пересечения, иначе цвет фона.

```
private Vec3d TraceRay(Vec3d O, Vec3d D, double t_min, double t_max, int depth)
{
    double closest_t = Double.PositiveInfinity;
    Primitive closest_object = null;

    ClosestIntersection(ref closest_object, ref closest_t, O, D, t_min, t_max);

    if (closest_object == null)
        return scene.background[x, y];

    Vec3d P = O + closest_t * D;
    Vec3d N;
    if (closest_object is Parallelepiped)
        N = Vec3dNormalParallelepiped(P, (Parallelepiped)closest_object);
    else if (closest_object is Cylinder)
        N = Vec3dNormalCylinder(P, closest_t, (Cylinder)closest_object, O, D);
    else if (closest_object is Cone)
        N = Vec3dNormalCone(P, closest_t, (Cone)closest_object, O, D);
    else if (closest_object is Triangle)
        N = Vec3dNormalTriangle((Triangle)closest_object);
    else if (closest_object is Plane)
    {
        Plane tmp = (Plane)closest_object;
        N = tmp.V;
    }
    else if (closest_object is DiskPlane)
    {
        DiskPlane tmp = (DiskPlane)closest_object;
        N = tmp.V;
    }
    else
        N = P - closest_object.C;

    N = N / Vec3d.Length(N);

    double intensity = ComputeLighting(P, N, -D, closest_object.specular);

    Vec3d localColor = intensity * closest_object.color;

    double r = closest_object.reflective;

    if (depth <= 0 || r <= 0)
        return localColor;

    Vec3d R = ReflectRay(-D, N);
    Vec3d reflectedColor = TraceRay(P, R, 0.001, Double.PositiveInfinity, depth - 1);

    Vec3d kLocalColor = (1 - r) * localColor;
    Vec3d rReflectedColor = r * reflectedColor;

    return kLocalColor+rReflectedColor;
}
```

Рисунок 3.1. Код метода TraceRay.

На рисунке 3.2 представлен код метода ComputeLighting, который вычисляет интенсивность точки.

```
private double ComputeLighting(Vec3d P, Vec3d N, Vec3d V, double specular)
{
    double intensity = 0;
    List<Light> sceneLight = scene.lights;

    for (int i = 0; i < sceneLight.Count; i++)
    {
        if (sceneLight[i].ltype == LightType.Ambient)
        {
            intensity += sceneLight[i].intensity;
        }
        else
        {
            Vec3d L;
            double t_max;
            if (sceneLight[i].ltype == LightType.Point)
            {
                L = sceneLight[i].position - P;
                t_max = 1;
            }
            else
            {
                L = sceneLight[i].position;
                t_max = Double.NegativeInfinity;
            }

            double shadow_t = Double.PositiveInfinity;

            Primitive shadow_object = null;
            ClosestIntersection(ref shadow_object, ref shadow_t, P, L, 0.001, t_max);
            if (shadow_object != null)
                continue;

            double n_dot_l = Vec3d.ScalarMultiplication(N, L);

            if (n_dot_l > 0)
            {
                intensity += sceneLight[i].intensity * n_dot_l / (Vec3d.Length(N) *
Vec3d.Length(L));
            }

            if (specular != -1)
            {

                Vec3d R = ReflectRay(L, N);
                double r_dot_v = Vec3d.ScalarMultiplication(R, V);

                if (r_dot_v > 0)
                {
                    intensity += sceneLight[i].intensity * Math.Pow(r_dot_v /
(Vec3d.Length(R) * Vec3d.Length(V)), specular);
                }
            }
        }
    }
    return intensity;
}
```

Рисунок 3.2. Код метода ComputeLighting.

### 3.3. Описание интерфейса программы

На рисунке 4.1 представлен общий интерфейс программы. Для управления сценой используется панель в правой части экрана.

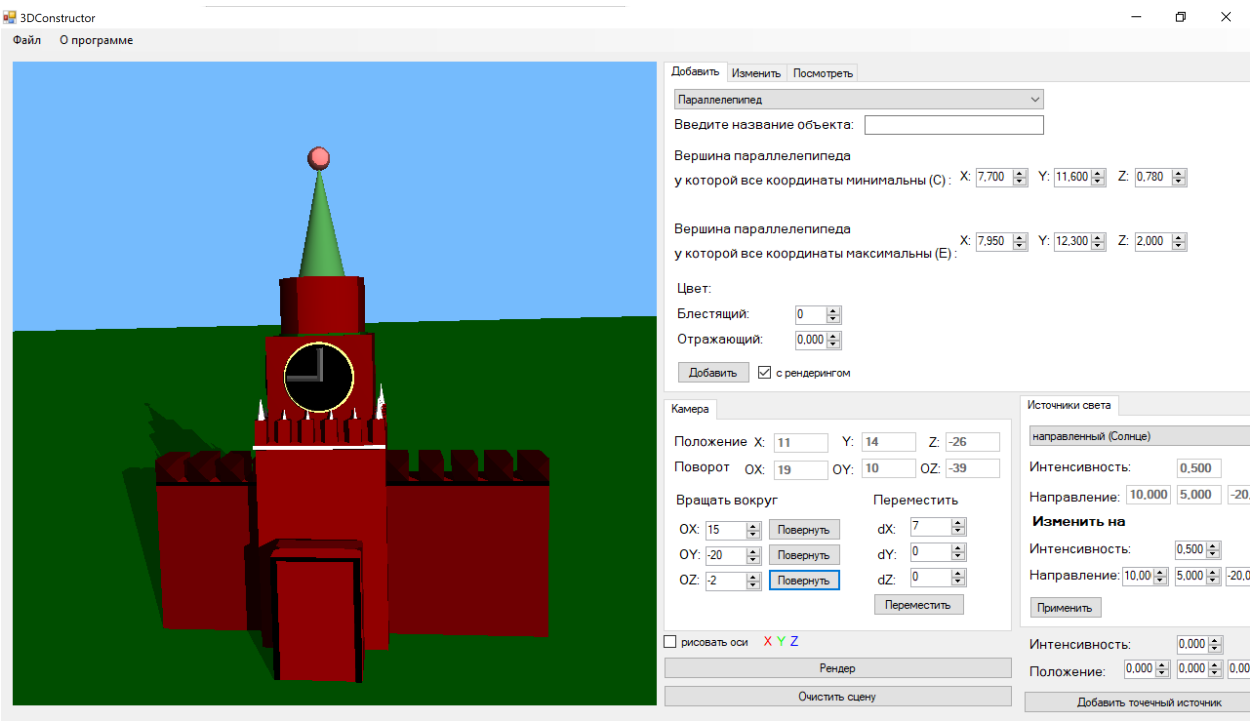


Рисунок 4.1. Общий интерфейс программы.

Блок «Камера» содержит текущее положение камеры и предоставляет возможность её передвинуть или повернуть (рисунок 4.2).

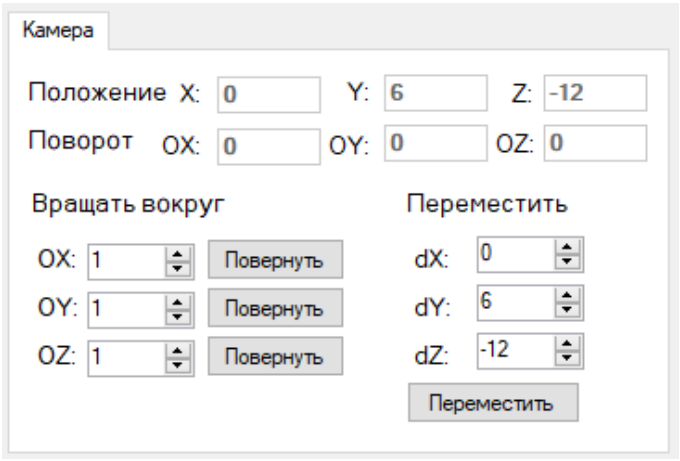


Рисунок 4.2. Блок «Камера».

Блок «Источники света» содержит тип источника,  его положение и интенсивность освещения (рисунок 4.3).

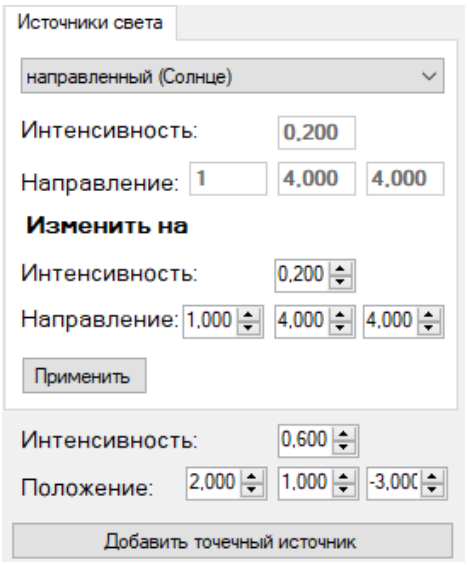


Рисунок 4.3. Блок «Источники света».

Блок «Работа с примитивами» предоставляет возможность добавить новый объект на сцену, изменить параметры примитивов и посмотреть, какие объекты присутствуют на сцене (рисунок 4.4).

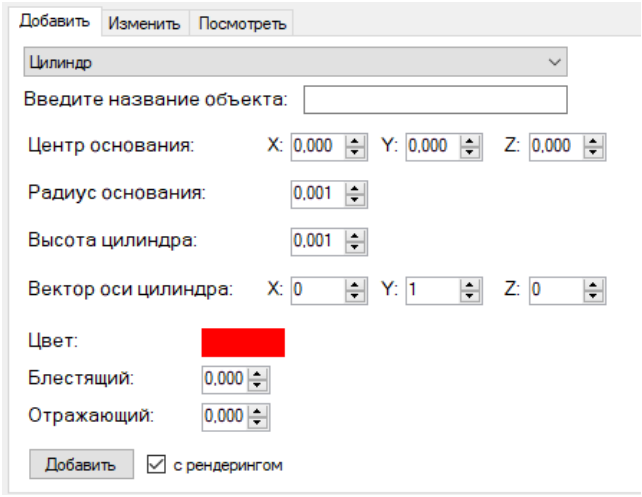


Рисунок 4.4. Блок «Работа с примитивами».

**Вывод**

В данном разделе были выбраны средства реализации, рассмотрен интерфейс программы, а так же описаны основные этапы 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 i3-8100. Для замеров времени использовалось пространство имен «System.Diagnostics».

## 4.1. Постановка эксперимента по замеру времени

Для проведения замеров времени выполнения реализации алгоритмов будет использована формула:

$$t = \frac{\mathrm{T}}{\mathrm{N}} \quad (9)$$

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, поверхности которых обладают зеркальным отражением. Время измеряется в микросекундах.

На рисунке 5.1 изображены графики зависимости времени рендеринга программы от количества объектов на сцене и числа потоков для параллельной версии алгоритма трассировки лучей:

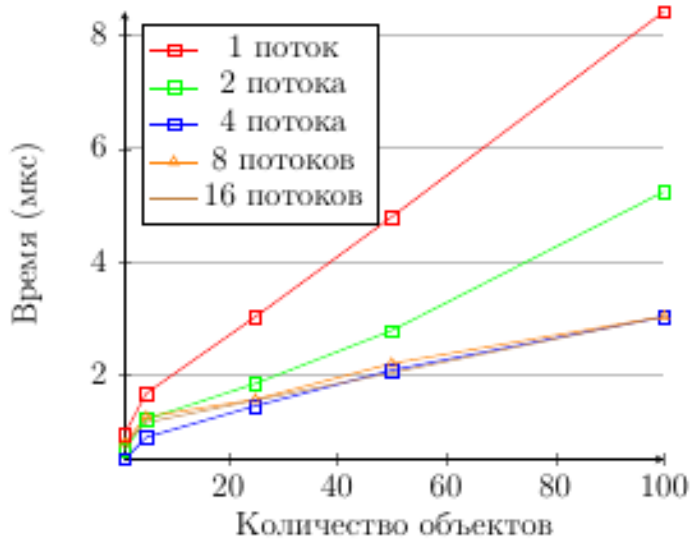


Рисунок 5.1. Сравнительный анализ времени рендеринга сцены от количества объектов.

Как видно из графика, время рендеринга сцены линейно зависит от количества объектов. Максимальная скорость работы достигается при количестве потоков равным 4. Дальнейшее увеличение их числа не дает прироста в скорости. Это связано с тем, что число потоков, которые работают параллельно, равно числу логических процессоров.

**Вывод**

В результате исследования временных затрат на рендеринг сцены с разными входными параметрами обнаружено, что от увеличения количества объектов линейно увеличивается и время рендеринга изображения.

# Заключение

Цель курсовой работы достигнута. Спроектировано и реализовано программное обеспечение, моделирующее памятник архитектуры «Спасская башня» и создание собственных сооружений, состоящих из сфер, цилиндров, конусов, параллелепипедов, треугольных и четырехугольных пирамид.

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.

# 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 28.06.19)
5. RayTracing – царь света и теней, Лев Дымченко [Электронный ресурс]. – Режим доступа: https://old.computerra.ru/206167/ (дата обращения 28.06.19)