|  |  |
| --- | --- |
|  | **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 огня, исходящего от свечи»***

Студент \_\_\_\_\_\_ИУ7-52Б\_\_\_\_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**А.А. Чалый**\_\_\_\_\_\_\_**

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

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_**А.В. Куров**\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2020 г.*

Оглавление

[Введение 3](#_Toc58097735)

[1. Аналитическая часть 5](#_Toc58097736)

[1.1. Постановка задачи 5](#_Toc58097737)

[1.2. Формализация объектов синтезируемой сцены 5](#_Toc58097738)

[1.3. Критерии выбора алгоритма 6](#_Toc58097739)

[1.4. Алгоритмы удаления невидимых линий и поверхностей 6](#_Toc58097740)

[1.5. Выбор модели освещения 10](#_Toc58097741)

[1.6. Выводы из аналитического раздела 13](#_Toc58097771)

[Вывод 13](#_Toc58097876)

[2. Конструкторская часть 14](#_Toc58097775)

[2.1. Схемы алгоритмов 14](#_Toc58097776)

[2.2 Пересечение луча с объектами сцены 17](#_Toc58097781)

[2.2.1 Пересечение луча со сферой 18](#_Toc58097786)

[2.2.2 Пересечение луча с плоскостью 19](#_Toc58097796)

[2.2.3 Пересечение луча с цилиндром 20](#_Toc58097808)

[2.2.4 Пересечение луча с конусом 21](#_Toc58097824)

[2.2.5 Пересечение луча с параллелепипедом. 22](#_Toc58097837)

[2.2.6 Пересечение луча с треугольником. 24](#_Toc58097846)

[2.3 Нахождение отраженного луча 26](#_Toc58097861)

[2.4 Диаграмма классов 27](#_Toc58097866)

[Вывод 28](#_Toc58097868)

[3. Технологическая часть 29](#_Toc58097869)

[3.1. Средства реализации 29](#_Toc58097870)

[3.2. Описание основных этапов реализации 30](#_Toc58097871)

[3.3. Описание интерфейса программы 32](#_Toc58097872)

[Вывод 34](#_Toc58097873)

[4. Исследовательская часть 35](#_Toc58097874)

[4.1. Постановка эксперимента по замеру времени 35](#_Toc58097875)

[Вывод 36](#_Toc58097876)

[Заключение 37](#_Toc58097877)

[Список литературы 38](#_Toc58097878)

# 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. Она используется повсеместно: для наглядного отображения данных, в компьютерных играх и даже в кино для создания эффектов.

Вследствие этого перед людьми, создающими трехмерные сцены, встает задача создания реалистичных изображений, которые будут учитывать такие оптические явления как преломление, отражение и рассеивание света, а также выбранную текстуру или цвет. Для создания еще более реалистичного изображения учитывается дифракция, интерференция, вторичные отражения света.

Существует множество алгоритмов компьютерной графики, которые решают эту задачу. Зачастую эти алгоритмы ресурсозатратны: чем более качественное изображение мы получаем в итоге, тем больше времени и памяти мы тратим на его синтез. Это становится проблемой при создании динамической сцены, где на каждом временном интервале необходимо производить расчеты заново.

Для решения поставленного задания необходимо проанализировать задачу, декомпозировать её, исследовать различные методы решения задач, выбрать подходящие методы, и выбрав язык программирования и определив нужные структуры, реализовать эти методы. Затем нужно спроектировать пользовательский удобный и красивый интерфейс.

Целью курсового проекта является разработка программы, для построения трехмерного изображения свечи и моделирование огня, исходящего от нее.

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 огонь.
* Проведение исследования на основе разработанной программы.

Итогом работы является программное обеспечение, демонстрирующее моделирование огня, исходящего от свечи. В программе предусмотрена возможность указать количество кадров в секунду, при которых будет проигрываться анимация, также допускается возможность генерации и сохранения заданного числа кадров для последующего моделирования огня.

# 1. Аналитическая часть

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

* 1. **Постановка задачи**

Необходимо смоделировать свечу с огнем, исходящим от ее фитиля, с учетом теней и зеркального отражения поверхности сфер, а также предоставить возможность генерации и сохранения заданного числа кадров для последующего моделирования огня. Данная программа является инструментом для создания реалистичной анимации огня, исходящего от свечи.

* 1. **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, плоскостей и цилиндров описываются аналитически.

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.
* Объекты, такие как сферы, цилиндры.
* Плоскости основания.
* Плоскости стены.
  1. **Критерии выбора алгоритма**

Критериями выбора алгоритма генерации огня служат получение реалистичного изображения, а также наличие тени, падающей от огня свечи.

* 1. **Алгоритм генерации огня**

Огонь - интенсивный процесс окисления, сопровождающийся излучением в видимом диапазоне и выделением тепловой энергии. В узком смысле - совокупность раскалённых газов (низкотемпературная плазма), выделяющихся в результате:

* Произвольного/непроизвольного нагревания горючего материала до определённой точки (здесь и далее под горючими материалами понимаются такие материалы, как древесина, а не вступившие в реакцию компоненты, допустим, сера) при наличии окислителя (кислорода);
* Химической реакции (в частности, взрыва);
* Протекания электрического тока в среде (электрическая дуга, электросварка).

**Моделирование с использованием смены текстур**

Самый простой способ смоделировать огонь - получить некоторое количество рисунков (или фотографий) пламени, раскрасить какой-либо объект сцены одной из них и каждый кадр сменять его текстуру на следующую.

Преимуществом данного моделирования является то, что способ прост в реализации, а также не требует особых моделей освещения.

Минус данного метода в том, что он не позволяет свободно настраивать параметры пламени, а также изменять пламя под действием внешних раздражителей, например, ветра.

**Моделирование с использованием системы частиц**

При горении выделяются видимые газы оттенков желтого и красного цвета, которые при отдалении от объекта горения тускнеют и совсем перестают быть различимы человеческим глазом. Если разбить эти газы на группы, можно заметить, что все они обладают одинаковыми свойствами. Следовательно, можно рассматривать огонь как систему частиц, на каждом кадре применяя к ним одинаковые операции. Этот метод используется в большинстве современных трехмерных приложениях и играх.

Очевидным преимуществом подобного моделирования является возможность изменения параметров пламени.

Минус данного метода в том, что изображение, полученное с помощью него, выглядит не так реалистично, как при моделировании с использованием смены текстур.

**Вывод**

Наиболее подходящим для моей программы методом является второй способ - моделирование с использованием частиц. Данный алгоритм предоставляет возможности изменения параметров генерируемого огня.

* 1. **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 работает в объектном пространстве и может быть применен для изображения множества выпуклых многогранников. Алгоритм состоит из трех больших этапов: удаления нелицевых граней для каждого тела, удаления видимых ребер данного тела, экранируемых другими телами сцены, удаление видимых ребер при взаимном “протыкании” тел.

Серьезным недостатком является вычислительная трудоемкость алгоритма. В теории она растет как квадрат количества объектов. Поэтому при большом количестве фигур находящихся в сцене, этот алгоритм будет показывать себя, как недостаточно быстрый. Можно использовать разные оптимизации для повышения эффективности, например сортировку по z. Минусом является и то, что алгоритм не позволяет визуализировать тени, зеркальные эффекты и преломление света. Использование растровых дисплеев, также снижает интерес к данному алгоритму, т.к. он работает в объектном пространстве.

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.

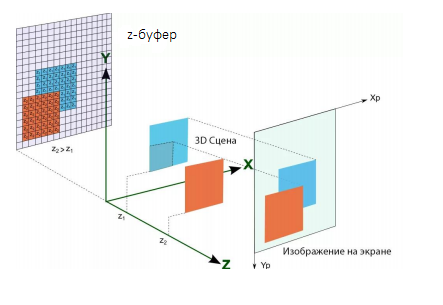


Рисунок 1.1. Работа алгоритма z-буфера.

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.

**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.

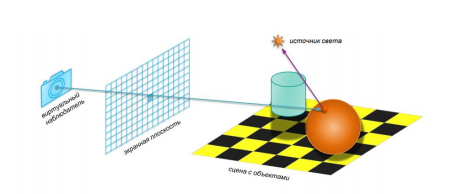


Рисунок 1.2. Работа алгоритма обратной трассировки лучей.

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.

**Вывод**

В ходе анализа приведенных алгоритмов, был выбран алгоритм обратной трассировки лучей, так как он позволяет получить изображение высокого качества, учитывая эффект отражения, а также предоставляет возможность работы с телами вращения.

* 1. **Выбор модели освещения**

Модели освещения делятся на локальные и глобальные. Локальные модели освещения не учитывают перенос света между поверхностями. Рассматривается свет только от явных точечных источников света, а взаимодействие ограничивается только однократным отражением света от непрозрачной поверхности. Глобальные модели освещения учитывают световое взаимодействие всех объектов сцены. В рамках этих моделей рассматриваются такие вопросы, как многократное отражение и преломление света, рассеянное освещение.

Так как огонь является источником света, необходимо выбрать модель освещения для придания реалистичного вида получаемой картине.

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 R на рисунке 1.4), и отраженного луча (вектор Е на рисунке 1.4).

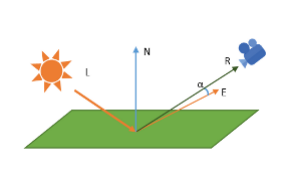


Рисунок 1.4. Получение бликов в модели освещения Фонга.

, где

– вектор нормали к поверхности в точке,

– падающий луч (направление на источник света),

– отраженный луч (направление идеально отраженного от поверхности луча),

,

- коэффициент фонового освещения

- коэффициент зеркального освещения

- коэффициент диффузного освещения

– коэффициент блеска материала.

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 ходе анализа приведенных моделей освещения, для реализации поставленной задачи была выбрана модель Фонга, т.к. она позволяет визуализировать объекты сцены, учитывая блики, которые делают изображение реалистичным.

**Выводы из аналитического раздела**

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

1. Моделирование с использованием системы частиц выбрана для изображения огня, исходящего от свечи. Он позволяет сгенерировать огонь наиболее правдоподобно моделирующий огонь свечи.
2. Алгоритм обратной трассировки лучей выбран для удаления невидимых линий и поверхностей. Он позволяет визуализировать эффект отражения, а также предоставляет возможность работы с телами вращения.
3. 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 модели освещения 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:
   1. Генерировать заданное количество кадров.
   2. Возможность сохранять сгенерированные кадры в указанной папке.
   3. Проигрывать анимацию огня.
   4. Загружать заранее сгенерированные кадры для последующей анимации.

**2.1. Схемы алгоритмов**

На рисунке 2.1 представлена схема алгоритма трассировки лучей.

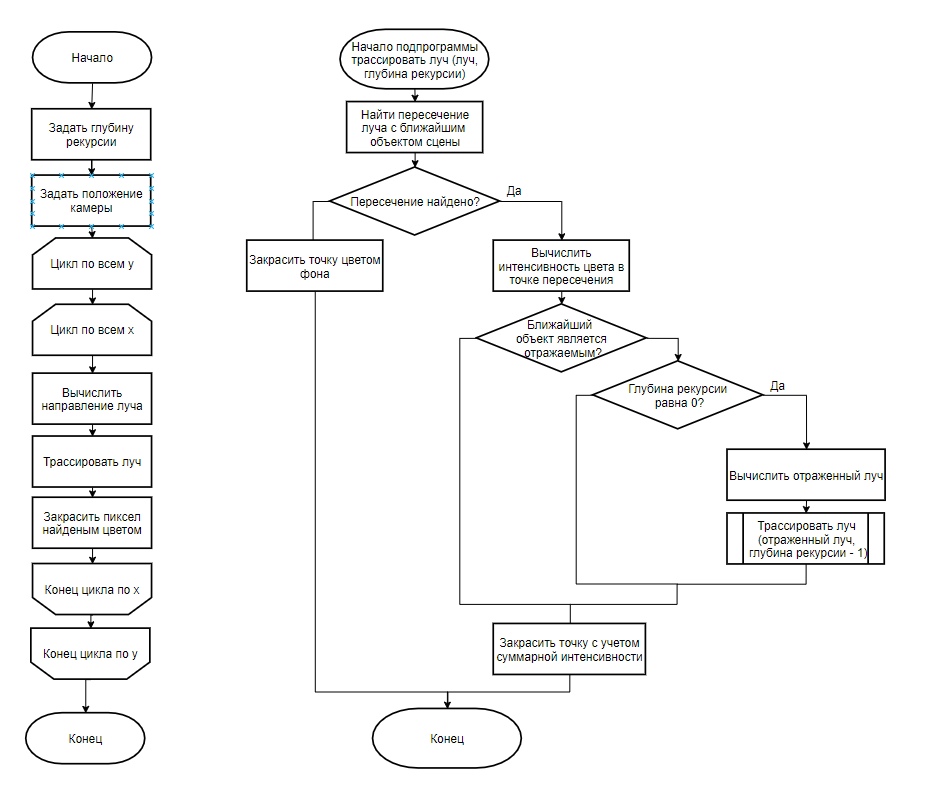


Рисунок 2.1. Схема алгоритма трассировки лучей.

На рисунке 2.2 представлена схема алгоритма освещения Фонга.

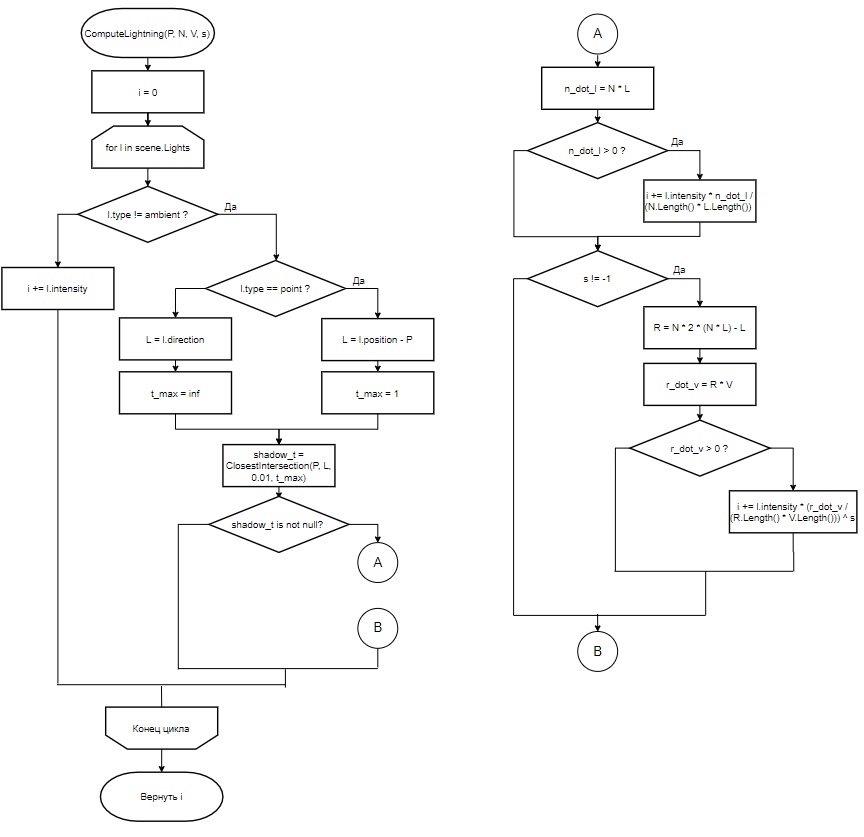


Рисунок 2.2. Схема алгоритма модели освещения Фонга.

**2.2 Пересечение луча с объектами сцены**

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

Необходимо рассмотреть объекты сцены, с которыми лучи сталкиваются. В сцене присутствуют сферы, цилиндры 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:

) – скалярное произведение векторов .

**2.2.1 Пересечение луча со сферой**

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

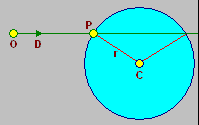


Рисунок 2.3. Пересечение луча со сферой

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

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.

**2.2.2 Пересечение луча с плоскостью**

Пусть – вектор нормали, P – точка пересечения луча с плоскостью, C – начальная точка (см. рис 2.4).

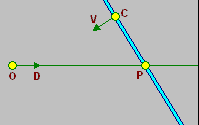


Рисунок 2.4. Пересечение луча с плоскостью

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:

**2.2.3 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 (см. рис 2.5).

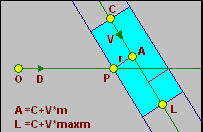


Рисунок 2.5. Пересечение луча с цилиндром

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.

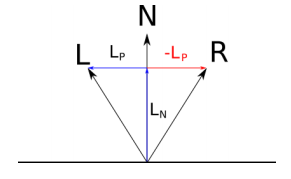


Рисунок 2.9. Разложение вектора падающего луча.

— 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:

**2.4 Диаграмма классов**

На рисунке 3 приведена диаграмма классов.

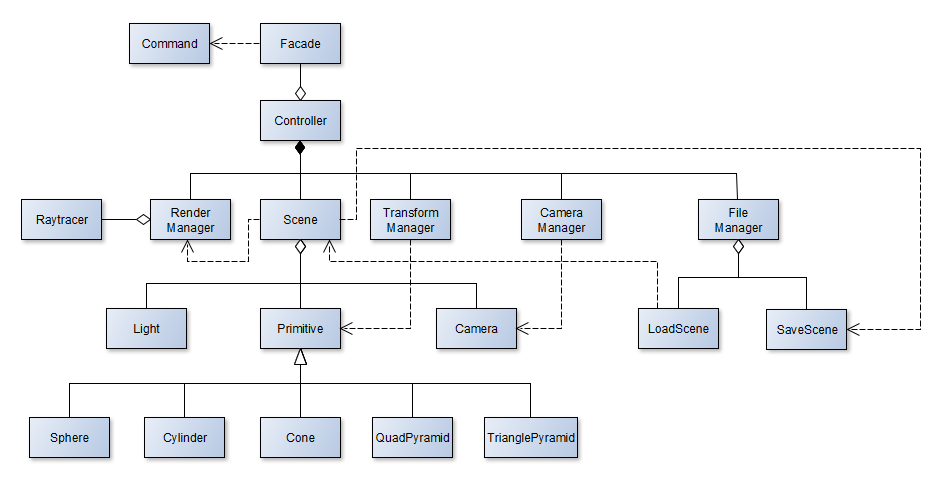


Рисунок 3. Схема классов программы.

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.
* 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
  + Faсade – 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 С# т.к:

* Данный язык использовался при программировании во время занятий по компьютерной графике, 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

В листинге 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.

**Листинг 3. 1 –** реализацияметода TraceRay.

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;

}

Листинг, как рисунок, надо подписывать снизу. Проверьте по ГОСТу По-моему, такого понятия нет в ГОСТе – листинг В листинге 3.2 представлен код метода ComputeLighting, который вычисляет интенсивность точки.

**Листинг 3.2 –** реализация метода ComputeLighting

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;

}

### 3.3. Описание интерфейса программы

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

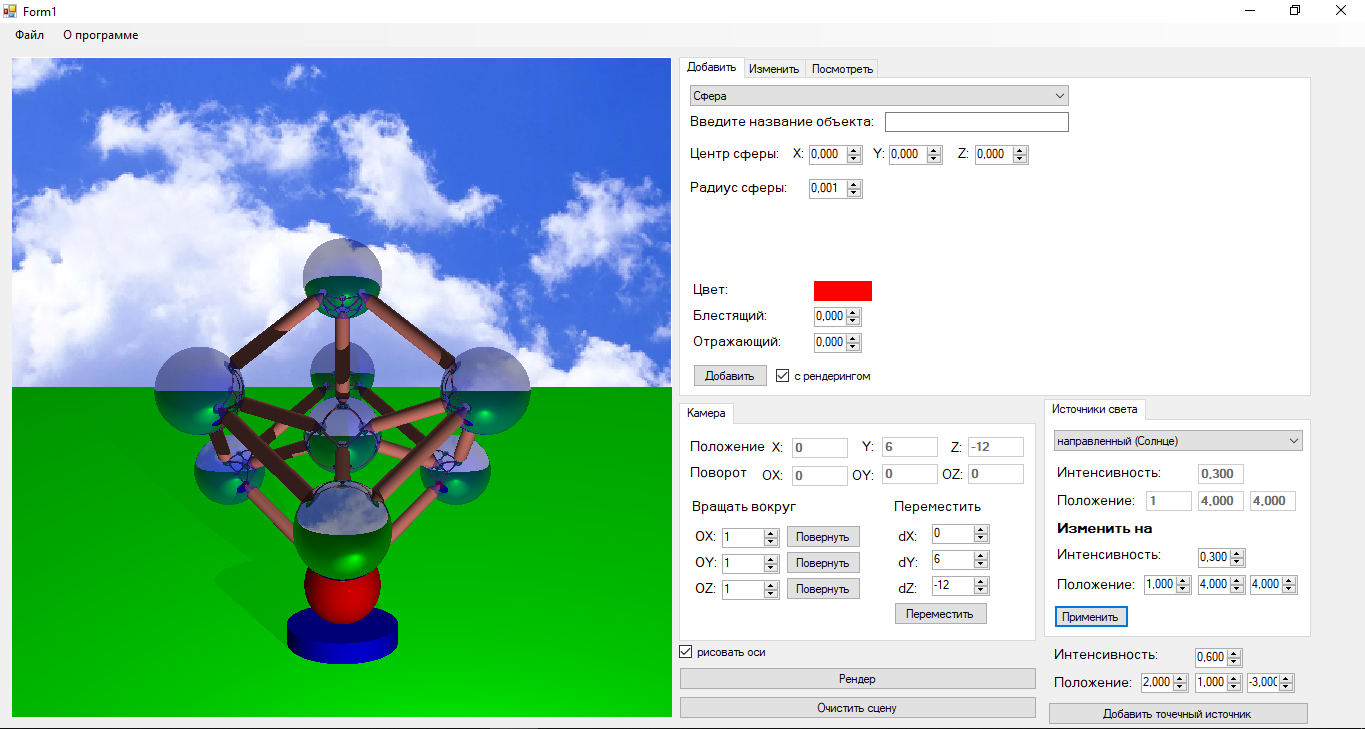


Рисунок 4.1. Общий интерфейс программы.

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

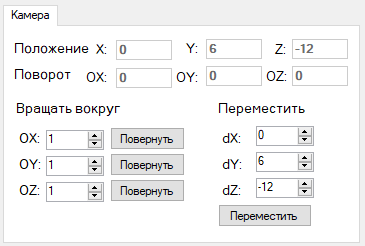


Рисунок 4.2. Блок «Камера».

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

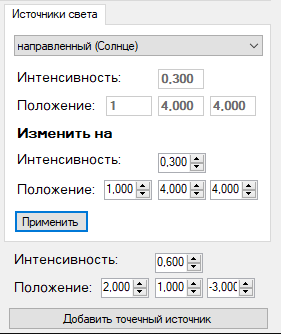


Рисунок 4.3. Блок «Источники света».

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

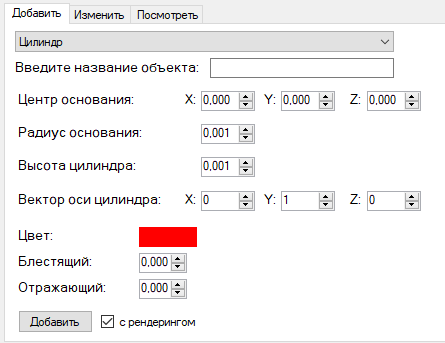


Рисунок 4.4. Блок «Работа с примитивами».

### Вывод

В данном разделе были выбраны средства реализации, рассмотрен интерфейс программы, а так же 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 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:

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:

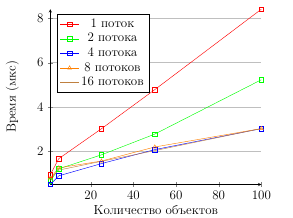


Рисунок 5.1. Сравнительный анализ времени рендеринга сцены от разных входных параметров.

Как видно из графика, время рендеринга сцены линейно зависит от количества объектов. Начиная с 4 потоков, увеличение их числа не дает прироста в скорости, это связано с тем, что число потоков, которые работают параллельно, равно числу логических процессоров, которых на экспериментальном процессоре. Последняя фраза требует исправления

### 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

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 памятник архитектуры «Атомиум» и создание собственных сооружений, состоящих из сфер, цилиндров, конусов, параллелепипедов, треугольных и четырехугольных пирамид.

Первым должен быть главный вывод, связанный с решением поставленной задачи (достижением цели).

# 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)