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:

«Трехмерная визуализация конструктора сцены»

Студент _____ИУ7-52Б_____ _____ _____Н.А.Гарасев _____

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

Руководитель курсового проекта _____ _____К.А.Кивва _____

(Подпись, дата) (И.О.Фамилия)

2020 г.

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)

УТВЕРЖДАЮ
Заведующий кафедрой ___ИУ7____
(Индекс)
_____ __И.В.Рудаков__
(И.О.Фамилия)
« _____ » _____ 2020 г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине _____Компьютерная графика_____

Студент группы ____ИУ7-52_____

_____Гарасев Никита Алексеевич_____
(Фамилия, имя, отчество)

Тема курсового проекта ___Трехмерная визуализация конструктора сцены_____
_____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____учебный_____
Источник тематики (кафедра, предприятие, НИР) _____кафедра_____

График выполнения проекта: 25% к _4_ нед., 50% к _7_ нед., 75% к 11 нед., 100% к 14 нед.

Задание разработать программу для построения трехмерной сцены, состоящей из некоторых примитивов таких, как сфера, правильная четырехугольная пирамида, цилиндр и параллелограмм, обладающих изменяемыми атрибутами такие, как положение в пространстве, размеры, зеркальность и отполированность. Необходимо предоставить возможность добавлять, удалять, а также изменять объекты сцены. Исследовать и выбрать алгоритм для построения трехмерного изображения и моделирования сцены, обеспечивающий работу программы в режиме реального времени.

Оформление курсового проекта:

Расчетно-пояснительная записка на 25-30 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
На защиту проекта должна быть представлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, диаграмма классов, интерфейс, характеристики разработанного ПО, результаты проведенных исследований.

Дата выдачи задания « ___ » _____ 20__ г.

Руководитель курсового проекта _____ ___К.А.Кивва._____
(Подпись, дата) (И.О.Фамилия)
Студент _____ ___Н.А.Гарасев,_____
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Оглавление

Введение

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.

Трёхмерная графика активно применяется для создания изображений на плоскости экрана или листа печатной продукции в науке и промышленности, например, в системах автоматизации проектных работ, архитектурной визуализации, в современных системах медицинской визуализации.

Итак, перед людьми встает задача создания реалистичных изображений. 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 «интуитивно понятный» интерфейс для удобства использования программы.

Таким образом, создание программного комплекса для выполнения вышеописанных операций является актуальным. Для получения адекватного результата необходимо разделить задачи моделирования и визуализации и рассмотреть плюсы и минусы возможных подходов для решения каждой из задач. Получившийся комплекс должен быть способен учитывать такие физические эффекты, как отражение, преломление и рассеивание света, чтобы создавать реалистичное изображение. Более того, возможность наложения на

жидкость текстуры и придания ей цвета способна значительно улучшить визуальные качества изображения.

Исходя из вышеописанного, целью практики является выбор или модификация существующих алгоритмов и их реализация для создания трёхмерной сцены.

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

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.

- Сооружения, состоящие из сфер, цилиндров, конусов, параллелепипедов, и правильных четырехугольных пирамид.

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.

1.5. Модель представления объектов сцены

Выбор модели представления объектов для сцены. Выбранная модель должна обеспечивать достаточную для задачи точность описания, предоставлять возможность визуализации всех объектов за реальное время.

Необходимо выбрать модель представления объектов. Большая часть трехмерных алгоритмов работает исключительно с многоугольниками. В таком случае выбор полигонального представления объектов обеспечивает независимость представления от конкретного алгоритма, который при необходимости можно изменить, не меняя остальную часть программы.

В качестве примитивов можно взять параллелепипед, сферу и цилиндр. Для параллелепипеда построение полигональной модели является тривиальным.

Для сферы и цилиндра разумно использовать общий алгоритм построения полигонального приближения поверхности вращения.

В качестве конкретного типа многоугольника, используемого в программе, был выбран треугольник. У треугольника есть ряд особенностей, которые делают его наиболее подходящим для применения в моделировании объектов в рамках компьютерной графики:

a) Любой многоугольник можно точно разделить на некоторое количество треугольников, треугольник же можно разделить только на меньшие треугольники. Можно утверждать, что треугольник представляет собой атомарный многоугольник.

b) Треугольник, за исключением вырожденных случаев, всегда задает одну плоскость и расположен в одной плоскости, в отличие от многоугольников с большим количеством вершин.

c) Треугольник не может быть невыпуклым, в отличие от других многоугольников. Это свойство упрощает его растеризацию.

d) В треугольнике относительно просто восстановить интерполяцией нормаль к исходной поверхности в любой точке, что актуально для алгоритмов затенения.

e) Треугольники позволяют эффективно использовать память, если хранить их в виде полоски треугольников.

1.6. Анализ алгоритмов удаления невидимых линий и поверхностей

При выборе алгоритма удаления невидимых линий нужно учитывать особенности формулировки задачи. Важным вопросом является необходимость реалистичного моделирования преломления и отражения света. Более того,

алгоритм должен работать достаточно быстро, чтобы смена кадров не опережала вычисление данных нового кадра.

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, затем идет поиск пересечений первичного луча с объектами сцены, в случае обнаружения пересечения, рассчитывается интенсивность пикселя, в зависимости от положения источника света, при отсутствии пересечения пиксель закрашивается цветом фона. Вычислительная сложность метода линейно зависит от сложности сцены. Качество изображения получается очень реалистичным, этот метод отлично подходит для создания фотореалистичных картин.

Серьезным недостатком этого алгоритма будет являться большое количество необходимых вычислений. Для получения изображения необходимо создавать огромное число лучей, проходящих через сцену и отражаемых от объекта. С точки зрения реализма изображения этот алгоритм превосходит остальные, позволяя визуализировать тени, эффекты прозрачности, преломления, отражения. Плюсом является и то, что данный метод позволяет изобразить гладкие объекты без аппроксимации их полигональными поверхностями.

Плюсы:

- Хорошо моделирует оптические эффекты
- Может быть ускорен параллельными вычислениями

Минусы:

- Очень ресурсозатратен

- Сложен в реализации

Алгоритм, использующий Z буфер

Несомненным плюсом данного алгоритма может являться его простота, которая не мешает решению задачи удаления поверхностей и визуализации их пересечения. В этом алгоритме не тратится время на сортировку элементов сцены, что дает преимущество в скорости работы. Особенно полезным это может стать при большом количестве объектов в сцене.

Так как размер синтезируемого изображения сравнительно мал, затраты по памяти, при хранении информации о каждом пикселе, в данном алгоритме незначительны для современных компьютеров.

К недостаткам данного алгоритма можно отнести то, что требуются большие затраты памяти для хранения информации о каждом пикселе, учитывая размер синтезируемого изображения. Также минусом является то, что непосредственно в сам алгоритм z-буфера нельзя внести тени и эффект прозрачности, необходима модернизация, добавляющая дополнительные буферы теней и прозрачности, что еще больше увеличивает расход памяти.

Плюсы:

- Прост в реализации
- Не требуется сортировка элементов сцены

Минусы:

- Требует хранение информации о каждом пикселе
- Недостаточно детально моделирует отражение и преломление

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.

Преимуществом данного алгоритма является точность вычислений. Она достигается за счет работы в объектном пространстве, в отличии от большинства других алгоритмов.

Алгоритм Робертса является очень точным в силу того, что вычисления производятся в объектном пространстве, однако его вычислительная сложность слишком велика. В силу того, что асимптотическая сложность алгоритма равна $\mathbf{O}(\boldsymbol{n^2})$, где n – количество объектов, при большом количестве квантов жидкости алгоритм будет слишком сложен для использования.

Плюсы:

- Очень точен

Минусы:

- Сильно теряет в производительности при большом количестве объектов

- Сложен в реализации
- Недостаточно детально моделирует отражение и преломление

Алгоритм Варнока

Алгоритм Варнока основывается на рекурсивном разбиении экрана. В зависимости от расположения объектов это может стать, как положительной, так и отрицательной стороной алгоритма. Чем меньше пересечений объектов, тем быстрее алгоритм завершит свою работу. Этот алгоритм плох по причине акцента на совсем не нужных задачах построения.

Плюсы:

- Так как не пересечений, весьма эффективен

Минусы:

- Недостаточно детально моделирует отражение и преломление
- Нет конкретной реализации

Вывод: из всех рассмотренных выше алгоритмов наилучшим выбором является **алгоритм обратной трассировки**. Динамическая визуализация в случае, рассматриваемом в данной работе, не обязана быть наиболее эффективна по времени обработки, по причине возможности пререндеринга объектов сцены.

1.7. Анализ методов закрашивания

Существует не мало методов закрашивания, каждый из которых имеет как плюсы, так и минусы

Простая закраска

В случае использования простой закраски каждая грань закрашивается одним уровнем интенсивности согласно закону Ламберта.

Плюсы:

- Наименее ресурсоёмкий метод закраски

Минусы:

- Плохо учитывает оптические эффекты
- Резкий переход цветов на гранях
- Плохо учитывает отражение

Закраска по Гуро

Данный метод закраски использует билинейную интерполяцию интенсивностей (иными словами, мы вычисляем интенсивности для вершин и получаем значения для граней интерполируя нормали вершин).

Плюсы:

- Не очень ресурсоёмок
- Неплохо имитирует криволинейные поверхности

Минусы:

- Не учитывает кривизну поверхности
- Недостаточно хорошо обрабатывает световые эффекты

Закраска по Фонгу

Данный метод закраски использует билинейную интерполяцию нормалей, что позволяет улучшить результирующую картинку за счёт повышения алгоритмической сложности.

Расчёт освещения по Фонгу требует вычисления цветовой интенсивности трёх компонент освещения: фоновой, рассеянной и глянцевых бликов.

Плюсы:

- Создаёт наиболее реалистичные изображения
- Наилучшим образом имитирует зеркальные блики

Минусы:

- Ресурсоёмок

Вывод: в качестве метода закрашивания был выбран метод **закраски по Фонгу** из-за реалистичного изображения.

1.8. Анализ алгоритмов построения теней

В силу выбора метода **трассировки лучей** вопрос построения теней оказывается решённым: пиксел затенён, если луч попадает на объект, и позже не попадает ни в объект, ни в источник света.

1.9. Анализ моделей 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.

Фотонная модель освещения

В фотонной модели освещения источник представляет собой плоскость, излучающую фотоны. По физическим законам определяется интенсивность и направление движения этих фотонов, которые определяют освещенность тех или иных граней.

Преимуществом данной модели освещения является то, что получается реалистичное изображение, соответствующее физическим законам (передает свет и материал объекта)

Недостатком данной модели освещения являются довольно сложные вычисления, вследствие чего ухудшается скорость визуализации.

Плюсы:

- Реалистичное изображение
- Соответствие физическим законам

Минусы:

- Ресурсоёмок
- Сложные вычисления

Модель Ламберта

Модель Ламберта позволяет симулировать диффузное освещение. Метод основан на том, что свет при попадании на поверхность рассеивается равномерно во все стороны. Таким образом, освещенность в точке определяется только плотностью света в точке поверхности, а она линейно зависит от косинуса угла падения. Данная модель освещения является одной из самых простых. Также она очень часто используется в комбинации других моделей, так как практически в любой другой модели освещения можно выделить диффузную составляющую.

Преимуществом, несомненно, является простота данной модели, а также она удобна для анализа свойств других моделей (за счет того, что ее легко выделить из любой модели и анализировать оставшиеся составляющие)

Очевидным недостатком является то, что данная модель моделирует лишь диффузное освещение.

Плюсы:

- Анализ свойств других моделей

Минусы:

- Ограниченность

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 которого определяется из закона равенства углов падения и отражения.

Расчёт освещения по Фонгу требует вычисления цветовой интенсивности трёх компонент освещения: фоновой, рассеянной и глянцевых бликов.

$I = K_a I_a + K_d(\bar{n}, \bar{l}) + K_s(\bar{n}, \bar{h})^p$

$\bar{n}$ - вектор нормали к поверхности в точке

$\bar{l}$ - направление проецирования (направление на источник света)

$\bar{h}$ - направление на наблюдателя

K_a - коэффициент фонового освещения

K_s - коэффициент зеркального освещения

K_d - коэффициент диффузного освещения

Преимущество модели Фонга в том, что алгоритм достаточно прост в реализации по сравнению с фотонной моделью освещения, хотя и сложнее модели Ламберта, так как по сути является расширением данной модели. Также, модель Фонга улучшает визуальные качества сцены, по сравнению с моделью Ламберта, добавляя в нее блики.

Главным недостатком данной модели является то, что многие оптические эффекты либо не учитываются, либо рассчитываются с сильным приближением.

Плюсы:

- Достаточно прост
- Наилучшим образом имитирует зеркальные блики

Минусы:

- Ресурсоёмок

Вывод: так как объектами сцены являются различные фигуры, имеющие не только шероховатую поверхность, лучшим выбором будет **модель освещения Фонга**

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:
    a. Добавлять и удалять объекты сцены.
    b. Изменять параметры объектов сцены.
    c. Изменять положение камеры и осуществлять её поворот.
    d. Добавлять точечные источники света.
    e. Изменять параметры источников света.

2.1. Схемы алгоритмов

На рисунке 1 представлена схема алгоритма трассировки лучей.

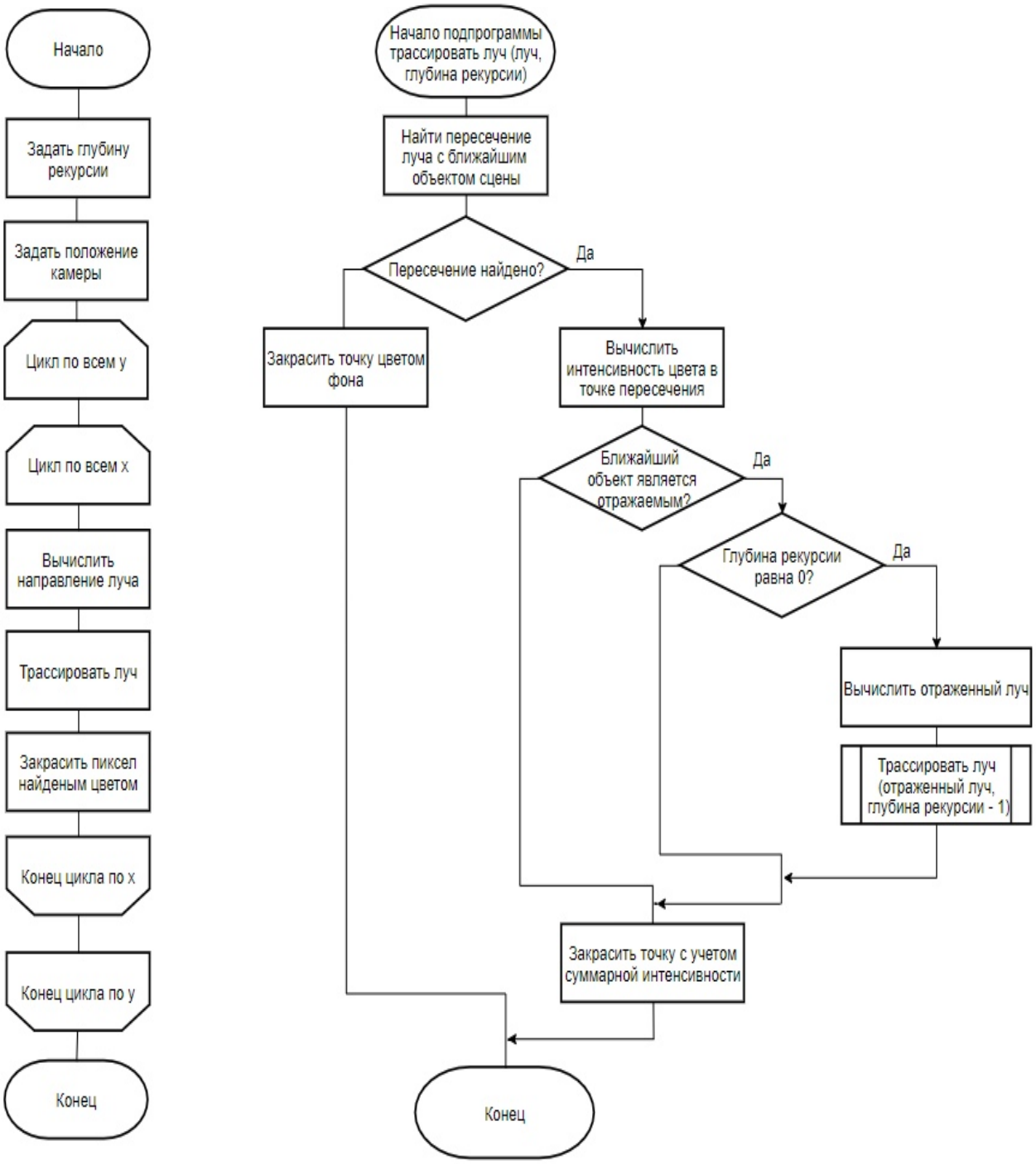

Рисунок 1. *Трассирование лучей*.

На рисунке 2 представлена схема алгоритма расчёта освещенности в соответствии с моделью Фонга.

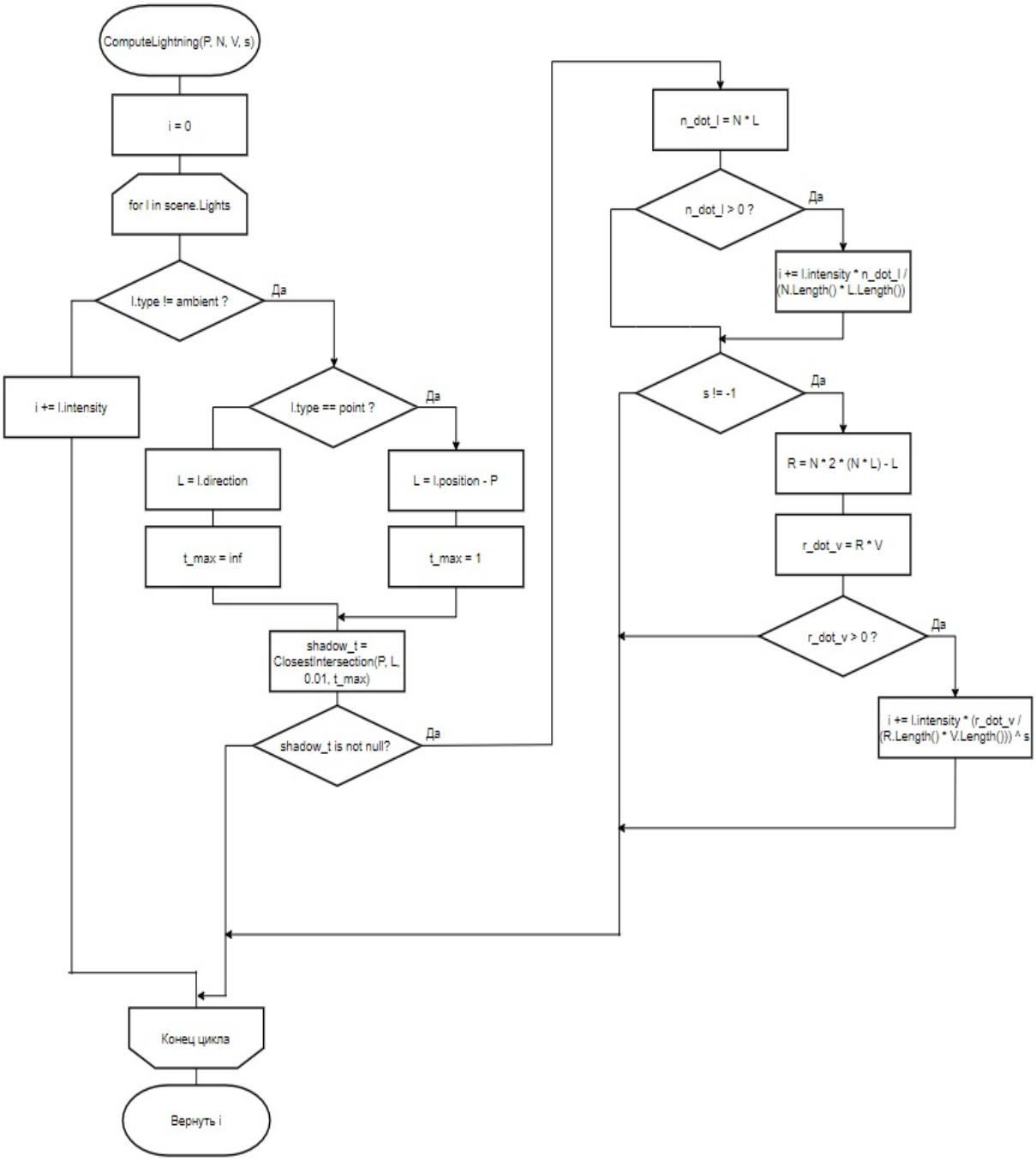

Рисунок 2. *Модель Фонга.*

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 3).

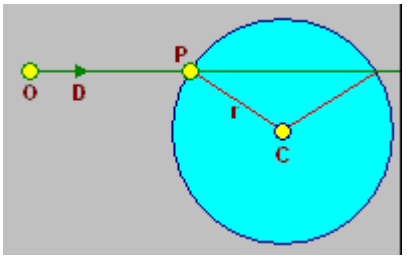

Рисунок 3. *Пересечение луча со сферой*

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 4).

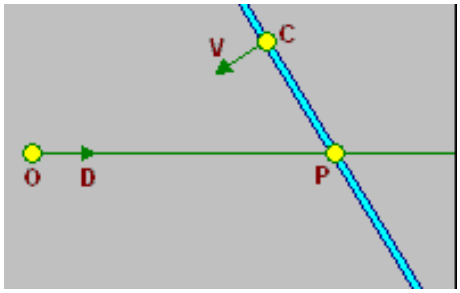

Рисунок 4. *Пересечение луча с плоскостью*

Уравнение плоскости:

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

Данное уравнение можно записать следующим образом:

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

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

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

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

Выразим отсюда t:

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

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 5).

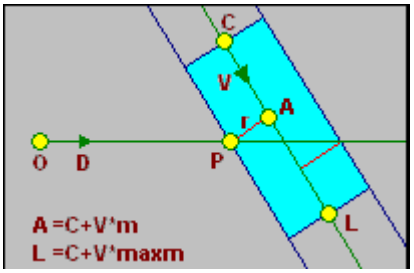

Рисунок 5. *Пересечение луча с цилиндром*

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:

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

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 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 6).

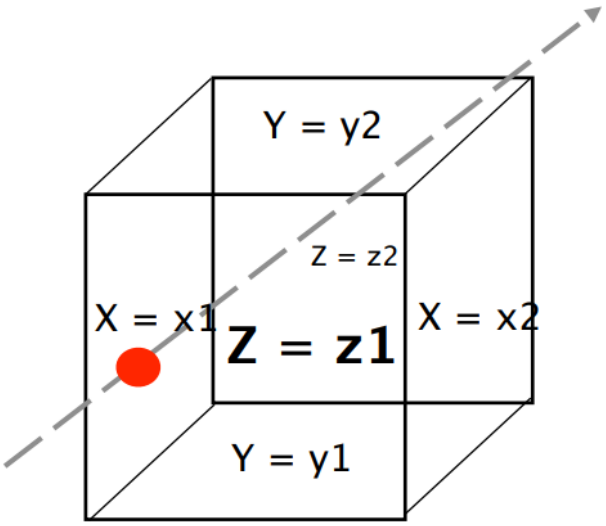

Рисунок 6. *Пересечение луча с параллелепипедом.*

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:

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.

2.2.5. 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 7).

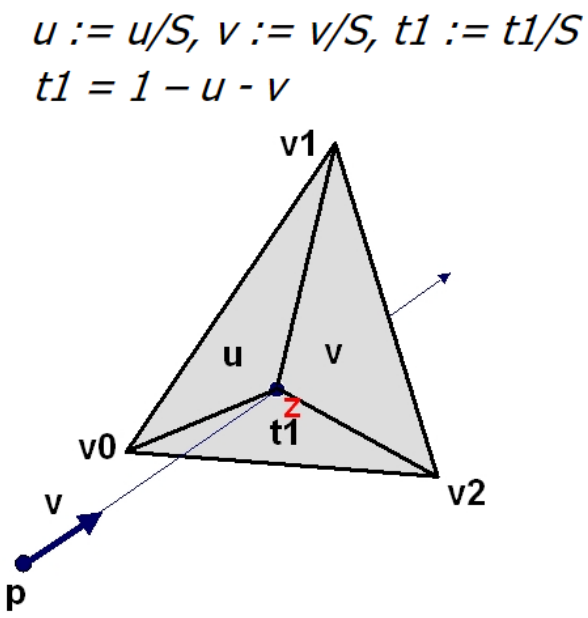

Рисунок 7. *Пересечение луча с треугольником.*

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 8.

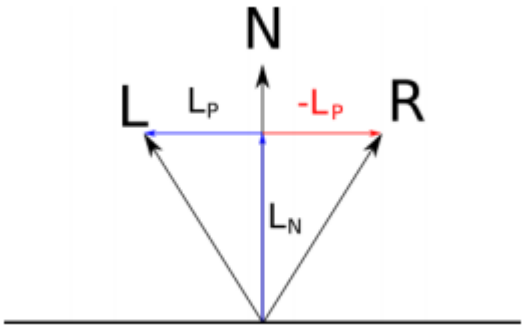

Рисунок 8. *Разложение вектора падающего луча.*

$\vec{L}n$ — 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 9 изображена диаграмма классов

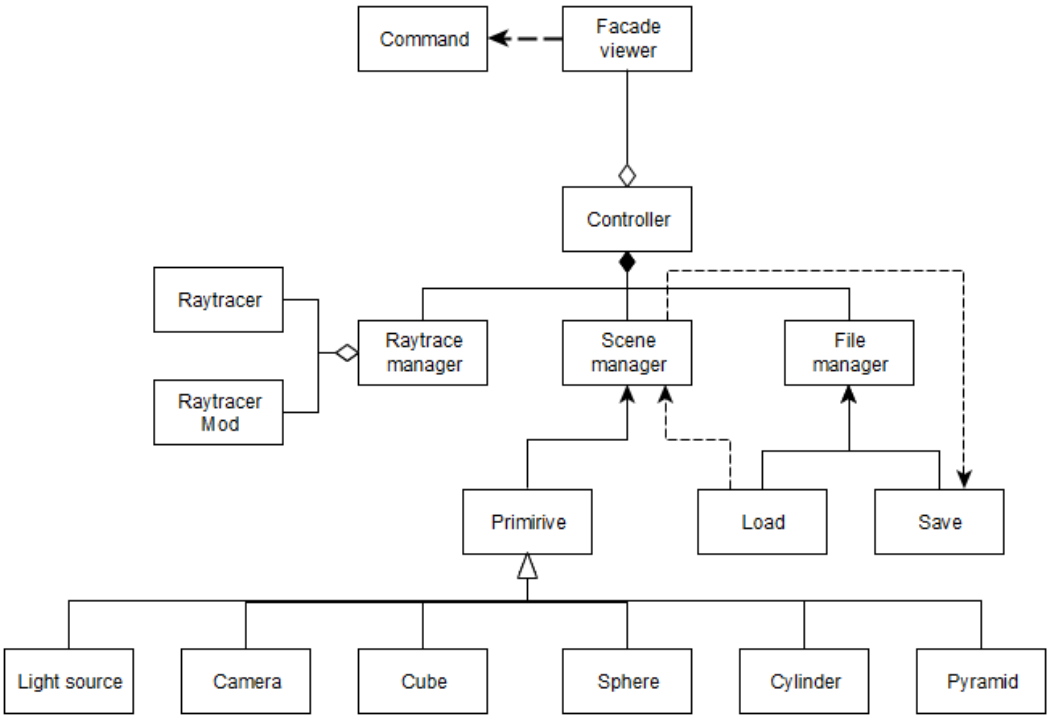

Рисунок 9. *Диаграмма классов*

Разработанная программа состоит из следующих классов.

Классы объектов:

- Primitive – базовый класс объектов;
- 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.
- Cube – класс параллелепипеда, который задается координатами двух своих вершин: той, у которой все три координаты минимальны, и той, у которой все три координаты максимальны.
- Pyramid – класс четырехугольной пирамиды с возможностью задания вершины пирамиды и точек основания.

Вспомогательные классы сцены:

- Camera – класс камеры с возможностью перемещения по сцене;
- Light source – 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.

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. Листинг программы

В листингах 1 – 2 представлены основные функции реализованной программы.

Листинг 1. Код метода TraceRay

```
private Vector3 TraceRay(Vector3 camera, Vector3 dir, double t_min, double t_max, int depth,
int x, int y)
{
    double closest_t = Double.PositiveInfinity;
    Primitive closest_object = null;

    ClosestIntersection(ref closest_object, ref closest_t, camera, dir, t_min, t_max);

    if (closest_object == null)
        return new Vector3();

    Vector3 P = camera + closest_t * dir;
    Vector3 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, camera, dir);
    else if (closest_object is DiskPlane)
    {
        DiskPlane tmp = (DiskPlane)closest_object;
        N = tmp.dir;
    }
    else if (closest_object is Triangle)
        N = Vec3dNormalTriangle((Triangle)closest_object);
    else
        N = P - closest_object.position;
    N = N / Vector3.Length(N);

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

    Vector3 localColor = intensity * closest_object.color;

    double r = closest_object.reflective;

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

    Vector3 R = ReflectRay(-dir, N);
    Vector3 reflectedColor = TraceRay(P, R, 0.001, Double.PositiveInfinity, depth - 1, x,
y);

    Vector3 kLocalColor = (1 - r) * localColor;
    Vector3 rReflectedColor = r * reflectedColor;

    return kLocalColor + rReflectedColor;
}
```

Листинг 2. Код метода ComputeLighting

```
private double ComputeLighting(Vector3 P, Vector3 N, Vector3 V, double specular)
{
```

```
    double intensity = scene.ambient_light.intensity;
    List<LightSource> sceneLight = scene.light_sources;

    for (int i = 0; i < sceneLight.Count; i++)
    {
        Vector3 L;
        double t_max;
        L = sceneLight[i].position - P;
        t_max = Double.PositiveInfinity;

        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 = Vector3.ScalarMultiplication(N, L);

        if (n_dot_l > 0)
        {
            intensity += sceneLight[i].intensity * n_dot_l / (Vector3.Length(N) *
Vector3.Length(L));
        }

        if (specular != -1)
        {

            Vector3 R = 2 * N * n_dot_l - L;
            double r_dot_v = Vector3.ScalarMultiplication(R, V);

            if (r_dot_v > 0)
            {
                intensity += sceneLight[i].intensity * Math.Pow(r_dot_v / (Vector3.Length(R)
* Vector3.Length(V)), specular);
            }
        }
    }
    return intensity;
}
```

3.3. Интерфейс программы

На рисунке 10 продемонстрирован интерфейс программы.

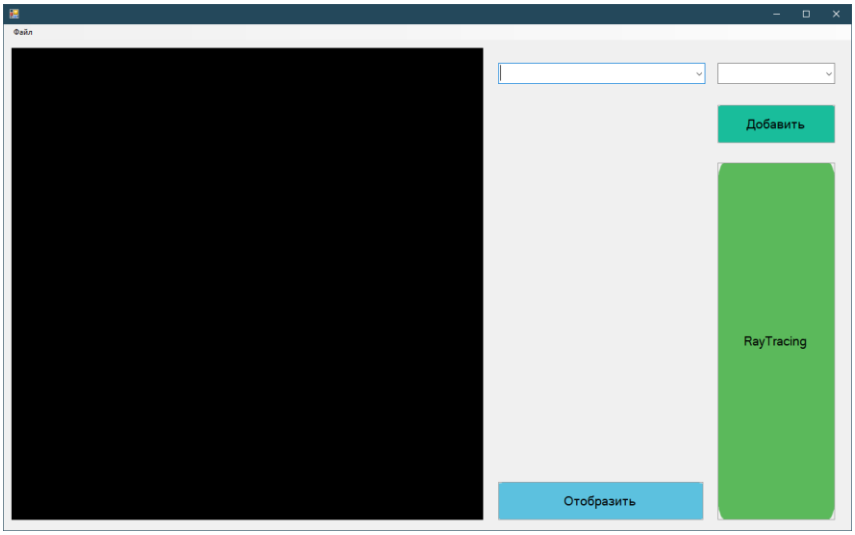

Рисунок 10. *Интерфейс.*

Интерфейс состоит из полотна для изображения, меню выбора объектов для добавления, меню выбора объектов, находящихся на сцене и кнопок отображения.

На рисунке 11 вы можете увидеть результат работы программы после загрузки сцены из файла.

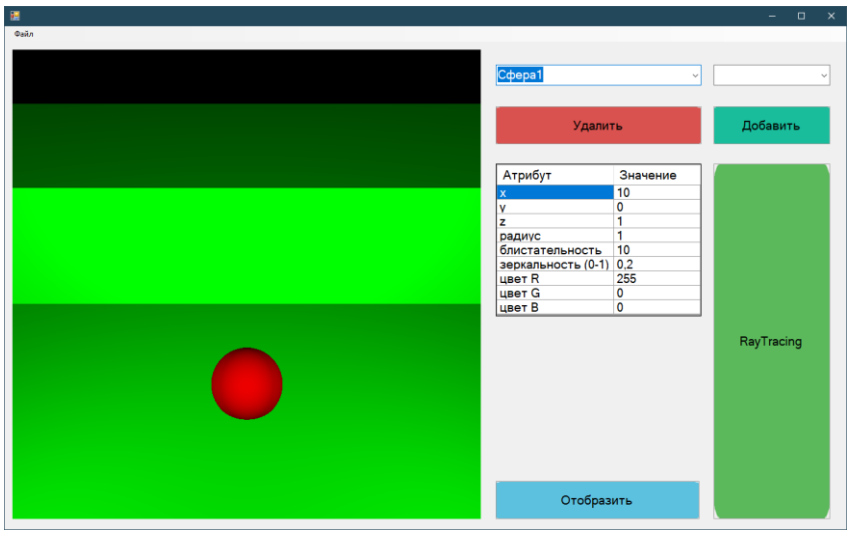

Рисунок 11. *Загрузка сцены*

Появилась таблица с атрибутами объекта, а также кнопка удаления.

На рисунке 12 продемонстрирован результат работы программы при нажатии на кнопку RayTracing.

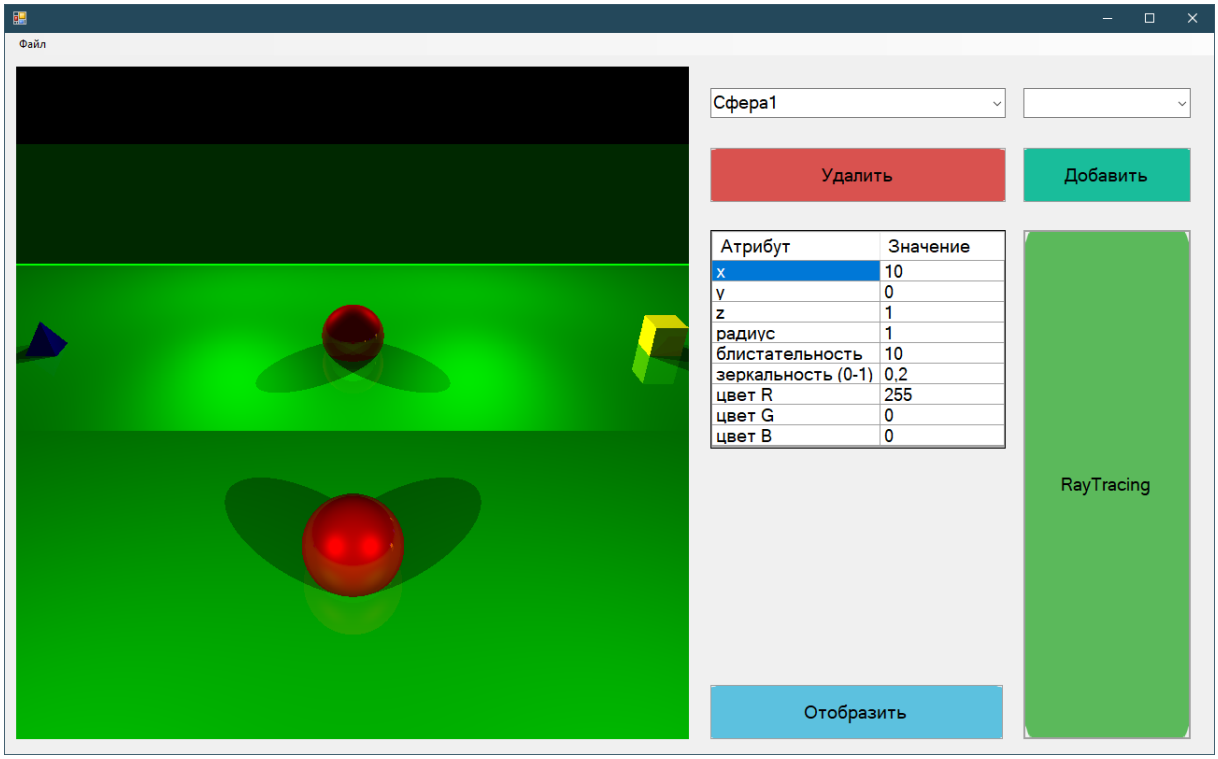

Рисунок 12. *RayTracing*

Задний параллелепипед оказался зеркалом, объект стал отбрасывать тени от источников света, а также стало возможно увидеть объекты находящиеся за пределом поля зрения камеры с помощью зеркала.

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. Сравнение реализаций трассирования лучей

В ходе выполнения курсового проекта было реализовано два алгоритма трассирования лучей. Алгоритмы отличаются сложностью реализаций. Для дальнейшего различения алгоритмов назовем их: простой и сложный. Простой отличает от сложного отсутствием теней и таких атрибутов, как зеркальностью объектов и их гладкость.

В таблице 1 записаны результаты замеров времени выполнения реализаций.

Таблица 1. *Время работы алгоритмов*

Реализация	Количество объектов	Время, сек
Простой	12	1.4
Сложный	12	2.8
Простой	23	1.8
Сложный	23	8.1

Вывод: наглядно видно, что время работы простой реализации гораздо меньше сложной, однако стоит учесть, что простой алгоритм не совсем справляется с задачей получения реалистичного изображения. Простой алгоритм крайне удобен для работы со сценой, для добавления объектов, изменения их

свойств и т. д. Когда все объекты расположены и изменены, то можно запускать сложный алгоритм для получения реалистичного изображения.

4.2. Зависимость времени работы алгоритма трассирования лучей от количества потоков программы

На рисунке 13 изображены графики зависимости времени рендеринга программы от количества объектов на сцене и числа потоков для параллельной версии алгоритма трассировки лучей:

5.

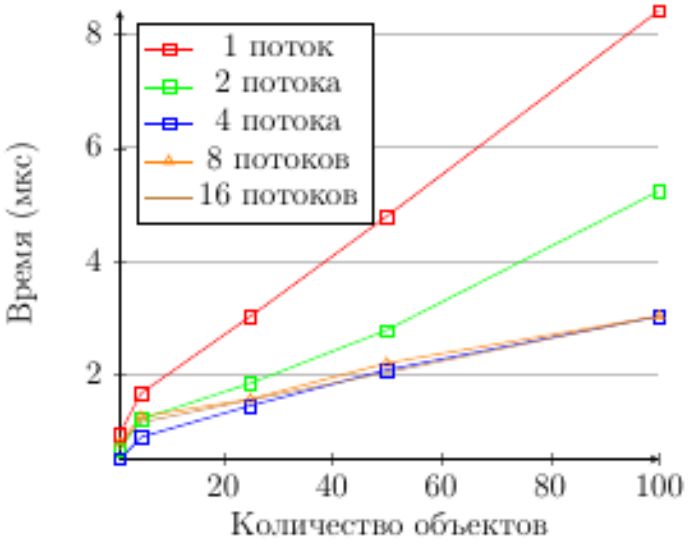

Рисунок 13. *Сравнительный анализ*

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.

Заключения

Цель курсовой работы достигнута. Спроектировано и реализовано программное обеспечение, позволяющее конструировать собственные сцены, состоящие из сфер, цилиндров, параллелепипедов 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.

Была проведена исследовательская работа на тему параллельных потоков и сравнения реализаций трассирования лучей.

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)