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 Информатики и систем управления

КАФЕДРА Программное обеспечение ЭВМ (ИУ7)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Визуализация отражения объектов в плоском зеркале»

Студент ИУ7-54Б _____ С. Д. Параскун
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ Ю. И. Терентьев
(Подпись, дата) (И.О.Фамилия)

2021г.

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)
« _____ » _____ 2021 г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине Компьютерная графика

Студент группы ИУ7-54Б

Параскун София Дмитриевна
(Фамилия, имя, отчество)

Тема курсовой работы Визуализация отражения объектов в плоском зеркале

Направленность КР (учебная, исследовательская, практическая, **производственная**, др.)
учебная
Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения работы: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Задание: разработать программу визуализации сцены, включающей объекты из заданного набора (куб, шар, призма, усеченная пирамида) и плоское зеркало. Предоставить пользователю возможность изменения диффузной составляющей коэффициента отражения зеркала. Продемонстрировать изменение отраженного изображения в зависимости от величины диффузной составляющей.

Оформление курсовой работы:

1) Расчетно-пояснительная записка на 35-40 листах формата А4.
Расчетно-пояснительная записка должна содержать постановку задачи, введение, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение, список литературы, приложения.
2) Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.д.)
На защиту проекта должна быть предоставлена презентация, состоящая из 10-15 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, интерфейс, результаты проведенных исследований.
Дата выдачи задания « ___ » _____ 2021г.

Руководитель курсовой работы _____ Ю. И. Терентьев
(Подпись, дата) (И.О.Фамилия)

Студент _____ С. Д. Параскун
(Подпись, дата) (И.О.Фамилия)

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

Содержание

Введение

Современная компьютерная графика широко используется в наши дни. Она помогает людям решить ряд задач, например, создание спецэффектов в кинематографе, разработка визуального контента в играх или анимации, моделирование различных установок. Для всего этого очень важно делать получаемое изображение максимально приближенным к реальному.

В повседневной жизни мы часто пользуемся таким предметом как зеркало. Для нас важно, чтобы то, что мы в нем видим, было четким и без искажения. Качество этого изображения напрямую зависит от покрытия зеркала. Его шероховатости влияют на отражение света от поверхности, а значит и на полученное мнимое изображение.

Целью курсовой работы является построение реалистичного изображения сцены, на которой располагаются трехмерные объекты и визуализируется их отражение в зеркале, свойства которого можно изменять.

Для достижения поставленной цели, необходимо решить следующие задачи:

- рассмотреть и выбрать алгоритмы трехмерной графики для визуализации объектов и построения их отражения в зеркале;
- изучить работу выбранных алгоритмов;
- спроектировать архитектуру программы и её интерфейс;
- реализовать выбранные алгоритмы и структуры данных.

Итогом работы станет программа, демонстрирующая отражение объектов в плоском зеркале и позволяющая менять диффузную составляющую его покрытия. Также должна быть предусмотрена возможность перемещения и поворота камеры, изменения положения и интенсивности источника освещения.

1. Аналитический раздел

1.1 Формализация задачи

Для исследования получаемого в зеркале изображения будет смоделирована сцена, состоящая из следующих объектов:

- композиция геометрических объектов, выбираемых пользователем (для выбора доступны сфера, куб, правильная усеченная четырехгранная пирамида, правильная трехгранная призма);
- платформа, на которой стоят геометрические объекты;
- плоское зеркало, свойства которого являются целью изучения;
- точечный источник освещения, задаваемый координатами положения и интенсивностью.

Геометрические объекты будут задаваться минимально достаточным набором информации:

- сфера – координаты центра и радиус;
- куб – координаты левой нижней ближайшей вершины, длина ребра и угол поворота вокруг оси Oy;
- усеченная пирамида – координаты левой нижней ближайшей вершины, длина основания, высота неусеченной версии пирамиды и высота усеченной (если исходная высота оказывается меньше усеченной, строится обычная четырехгранная пирамида высотой, равной наименьшему из двух значений);
- призма – координаты левой нижней ближайшей вершины, длина основания, высота призмы.

Наиболее удобным представлением вышеперечисленных объектов, за исключением сферы, будет полигональная сетка в виде списка граней. Также

для каждой фигуры задается цвет RGB-тройкой и характеристики материала (т.е. свойствами взаимодействия со светом): коэффициент отражения, зеркальность и шероховатость поверхности.

Также необходимо реализовать камеру, задаваемую координатами местоположения и вектором, определяющим направление взгляда.

Для того, чтобы поставленная задача реализовывалась быстро, независимо от меняющихся объектов, источников освещения и покрытия плоского зеркала, необходимо выбрать алгоритмы, которые будут максимально эффективны в данном случае.

1.2 Алгоритмы построения изображения

Чтобы правильно построить изображение, необходимо воспользоваться алгоритмом, который будет корректно и быстро удалять невидимые линии и поверхности. Некоторые алгоритмы работают в объектном пространстве, в таком случае мы имеем дело с мировой системой координат. Другие же оперируют в пространстве изображения – экранная система координат [1]. Рассмотрим часто используемые алгоритмы и выберем оптимальный.

1.2.1 Алгоритм Робертса

Данный алгоритм работает в объектном пространстве и только с выпуклыми телами. В случае невыпуклых тел необходимо сначала провести разбиение его на выпуклые.

Для каждого объекта формируется матрица тела — для каждой грани вычисляются коэффициенты уравнения плоскости, проходящей через нее, после необходимо проверить матрицу на корректность [2].

Первоочередно в алгоритме удаляются грани и ребра, экранируемые самим телом, в дальнейшем — другими телами. При этом возможны несколько случаев:

- ребро полностью невидимо;
- ребро полностью видимо;

- ребро экранируется частично, в таком случае ребро разбивается на несколько частей.

Преимуществом данного алгоритма является использование математических методов, которые просты и обладают высокой точностью.

Главным недостатком является рост сложности алгоритма. Ее зависимость представляет собой квадрат числа всех объектов сцены. Также если рассматривать уже оптимизированные варианты алгоритмов, их реализация оказывается достаточно трудоемкой [3].

1.2.2 Алгоритм Варнока

Данный алгоритм работает в пространстве изображения, разбивая картинную плоскость на части (области), для каждой из которых исходная задача может быть решена относительно просто.

Для каждой части определяются связанные с ней многоугольники, и полностью видимые из их числа отображаются на экране. В ином случае область разбивается на подобласти до тех пор, пока не достигнется простая видимость или же область не станет размером в 1 пиксель. Предполагается, что чем меньше область, тем меньше многоугольников ее перекрывают. Однако если полученная область в 1 пиксель все еще перекрыта несколькими многоугольниками, определяется глубина каждого из них и визуализируется находящийся ближе к наблюдателю.

Объективным преимуществом является простота понимания алгоритма, однако анализ картинной плоскости с последующим ее разбиением и отображением может занять большое количество времени [3].

1.2.3 Алгоритм, использующий Z-буфер

Данный алгоритм работает в пространстве изображения. В ходе его работы значение z-координаты (глубины) каждого нового пикселя, заносимого в буфер кадра, сравнивается с z-координатой пикселя уже занесенного в Z-буфер. Если сравнение показывает на то, что новый пиксель располагает-

ся ближе к наблюдателю, то Z-буфер обновляется, а также корректируется значение интенсивности в буфере кадра [2].

Алгоритм является одним из простейших в реализации. Его сложность линейно зависит от сложности сцены. Также алгоритм Z-буфера позволяет рассматривать объекты сцены в произвольном порядке, не затрачивая время на их сортировку.

Несмотря на подобную простоту и удобство алгоритма, он является затратным по памяти, так как нам приходится хранить буфер кадра и Z-буфер для попиксельного хранения изображений.

1.2.4 Алгоритм прямой трассировки лучей

Данный алгоритм работает в пространстве изображения. Для каждого пикселя картинной плоскости определяется ближайшая к нему грань, это производится выпусканием луча и нахождением всех его пересечений с гранями. Так как объекты могут отражать, поглощать или пропускать лучи света сквозь себя, необходимо учитывать все образующиеся лучи.

Для каждого первичного луча строится дерево трассировки, ветви которого составляют вторичные лучи. Ветвление заканчивается, когда луч вышел за пределы сцены, встречается с непрозрачным телом, попадает в источник света, когда интенсивность падает ниже порога чувствительности или когда расщепление первичного луча становится слишком велико.

Из преимуществ можно отметить то, что достигается высокая реалистичность получаемого изображения. Помимо этого каждый луч можно рассматривать независимо от остальных, что предоставляет возможность для распараллеливания выполнения построений.

Однако низкая производительность является серьезным недостатком. Для получения изображения приходится выпускать множество лучей, проходящих через сцену и отраженных от объектов [3].

1.2.5 Алгоритм обратной трассировки лучей

Метод обратной трассировки лучей исправляет главный недостаток прямой трассировки – большое количество испускаемых лучей.

Строится путь светового луча от приемника к объекту и источнику. Данный метод аккумулирует все лучи, в действительности приходящие в приемник из определенного направления независимо от их начала. Количество лучей ограничено числом точек на поверхностях объектов сцены, видимых из точки наблюдения, поэтому вычислительные затраты значительно уменьшаются по сравнению с методом прямой трассировки.

Вывод: Для реализации выбран алгоритм обратной трассировки, так как он позволяет учесть все особенности поверхностей фигур. Также данный метод является одним из этапов построения отражения в плоском зеркале, который будет рассмотрен далее.

1.3 Модели освещения

Для моделирования освещения трехмерных объектов используются различные модели освещения. Рассмотрим основные из них.

1.3.1 Модель Ламберта

Модель Ламберта — одна из простых моделей освещения, которая представляет собой комбинацию фоновой и диффузной составляющих. Поверхность при такой модели будет выглядеть одинаково ярко со всех направлений наблюдений. Интенсивность имеет вид:

$$I = I_a + I_d, \tag{1.1}$$

где I_a – фоновая составляющая (ambient), I_d – рассеянная составляющая (diffuse).

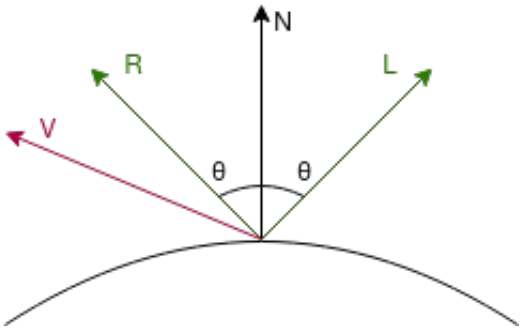

Рисунок 1.1 – Схема рассматриваемых при падении лучей

Фоновое освещение — постоянная в каждой точке величина надбавки к освещению. Вычисляется как:

$$I_a = k_a \cdot i_a, \tag{1.2}$$

где k_a – свойство материала воспринимать фоновое освещение, i_a – мощность фонового освещения. Чаще всего задается некое глобальное фоновое освещение всей сцены. Схожим образом рассчитывается и диффузная составляющая освещения:

$$I_d = k_d \cdot i_d \cdot cos(\vec{L}, \vec{N}), \tag{1.3}$$

где k_d – свойство материала воспринимать рассеянное освещение, i_d – мощность рассеянного освещения, $\vec{L}$ – луч, обратный падающему, $\vec{N}$ – нормаль к точке поверхности (рисунок 1.1).

Для удобства рассмотренные векторы можно взять как единичные, тогда косинус угла между ними совпадает со скалярным произведением и формула (1.3) примет вид:

$$I_d = k_d \cdot i_d \cdot (\vec{L}, \vec{N}). \tag{1.4}$$

1.3.2 Модель Фонга

Модель Фонга — классическая модель освещения. Представляет собой комбинацию фоновой, диффузной составляющей (модели Ламберта) и зеркальной составляющей, работает таким образом, что кроме равномерного

освещения на материале может еще появляться блик.Его местонахождение определяется из закона равенства углов падения и отражения.

В данной модели интенсивность имеет вид:

$$I = I_a + I_d + I_s, \tag{1.5}$$

где I_a – фоновая составляющая (ambient), I_d – рассеянная составляющая (diffuse), I_s – зеркальная составляющая (specular).

Формула для зеркальной составляющей имеет вид:

$$I_s = k_s \cdot i_s \cdot cos^{\alpha}(\vec{R}, \vec{V}), \tag{1.6}$$

где k_s – коэффициент зеркального отражения, i_s – мощность зеркального освещения, $\vec{R}$ – направление отраженного луча, $\vec{V}$ – направление на наблюдателя (рисунок 1.1), α – коэффициент блеска материала поверхности.

В случае единичных векторов формула (1.6) принимает вид:

$$I_s = k_s \cdot i_s \cdot (\vec{R}, \vec{V}). \tag{1.7}$$

Вывод: При решении будет использоваться модель Фонга, так как она учитывает все интересующие в ходе данной работы составляющие освещения.

1.4 Визуализация размытых отражений

В данной работе изучается диффузная составляющая покрытия зеркала, а значит показатель ее шероховатости будет изменять отраженные от объектов лучи, и изображение будет принимать другой вид.

При построении размытого отражения, в отличие от отражения идеального зеркала, для отраженного луча выбирается произвольное направление в некотором конусе, ось которого совпадает с вектором идеально отраженного луча, а радиус зависит от диффузной составляющей покрытия. Итоговое значение интенсивности может быть найдено, как среднее от значений,

полученных для каждого выпущенного луча.

Чтобы избежать трудоемких операций с синусами и косинусами, можно аппроксимировать конус четырехугольной пирамидой, высотой которого является вектор идеально отраженного луча. Основание будет перпендикулярно отраженному лучу, а длина его стороны будет зависеть от шероховатости поверхности – чем больше диффузная составляющая, тем больше сторона, а значит и отражение менее четкое.

1.5 Вывод из раздела

В данном разделе были рассмотрены алгоритмы построения реалистичного изображения. При реализации поставленной задачи будут использоваться алгоритмы обратной трассировки лучей, модель освещения Фонга. Обратная трассировка также ляжет в основу алгоритма построения размытых отражений.

2. Конструкторский раздел

Для реализации алгоритма обратной трассировки необходимо рассмотреть алгоритм поиска отраженного луча, вычислений нормалей к сфере и треугольнику, алгоритмы определения пересечения луча и сферы, луча и треугольника.

2.1 Поиск отраженного луча

При пересечении падающего луча с поверхностью необходимо найти направление отраженного луча (если объект не поглощает падающие лучи полностью). Падающий и отраженный лучи, а также нормаль к поверхности в точке касания лежат в одной плоскости (рисунок 2.1), если объект является идеальным зеркалом, то угол падения равен углу отражения.

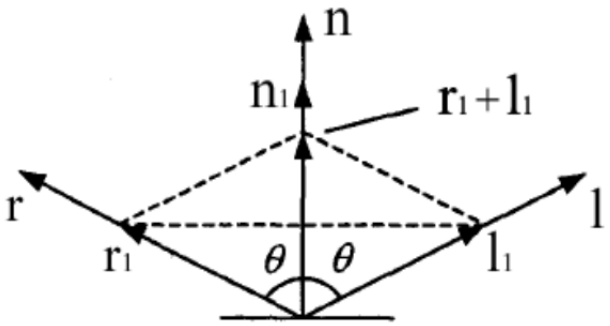

Рисунок 2.1 – Отражение луча от поверхности

На рисунке 2.1 представлены радиус-векторы от точки касания. $\vec{l}$ направлен на источник света, $\vec{n}$ – нормаль. Нужно найти вектор отраженного луча $\vec{r}$.

Используем для рассмотрения единичные вектора $\vec{r_1}$ и $\vec{l_1}$, справедливо записать следующее:

$$\vec{r_1} + \vec{l_1} = \vec{n'}, \tag{2.1}$$

где $\vec{n'}$ – вектор, соответствующий диагонали ромба и совпадающий по направлению с нормалью. Длина этого вектора равна $2cos\theta$.

Так как $\vec{n'}$ по направлению совпадает с $\vec{n_1}$, можно записать следующее:

$$\vec{n'} = 2cos\theta \cdot \vec{n_1} = \vec{r_1} + \vec{l_1}. \tag{2.2}$$

Из (2.2) можно найти единичный вектор отраженного луча:

$$\vec{r_1} = 2cos\theta \cdot \vec{n_1} - \vec{l_1} = 2cos\theta \cdot \frac{\vec{n}}{|\vec{n}|} - \frac{\vec{l}}{|\vec{l}|}. \tag{2.3}$$

Используя скалярное произведение векторов $\vec{n}$ и $\vec{l}$ нетрудно найти $cos\theta$:

$$cos\theta = \frac{(\vec{l}, \vec{n})}{|\vec{l}| \cdot |\vec{n}|}. \tag{2.4}$$

Подставим это значение в (2.3):

$$\vec{r_1} = 2 \cdot \frac{\vec{n} \cdot (\vec{l}, \vec{n})}{|\vec{l}| \cdot |\vec{n}^2|} - \frac{\vec{l}}{|\vec{l}|}. \tag{2.5}$$

Положим, что $\vec{n}$ и $\vec{l}$ нормализованы. Таким образом получим:

$$\vec{r} = 2\vec{n} \cdot (\vec{l}, \vec{n}) - \vec{l}. \tag{2.6}$$

2.2 Вычисление нормалей

В алгоритме обратной трассировки для поиска отраженного луча вычисляются нормали к объектам. Рассмотрим для используемых в работе полигонов (треугольников) и сферы.

2.2.1 Вычисление нормали к полигону (треугольнику)

Чтобы найти нормаль к плоскости, которую можно задать двумя векторами, достаточно найти векторное произведение. Можно посчитать нормально только один раз, при инициализации сцены, т.к. в каждой точке

треугольника нормали совпадают.

Пусть задан треугольник ABC, координаты его вершин известны. Нетрудно вычислить координаты векторов $\vec{AB}$ и $\vec{AC}$:

$$\vec{AB} = \{X_B - X_A, Y_B - Y_A, Z_B - Z_A\}, \vec{AC} = \{X_C - X_A, Y_C - Y_A, Z_C - Z_A\}. \tag{2.7}$$

Векторным произведением будет:

$$\vec{n} = \vec{AB} \times \vec{AC}. \tag{2.8}$$

Так как мы рассматриваем треугольник как исходную плоскость, необходимо нормализовать нормаль. Тогда координаты вектора нормали вычисляются следующим способом:

$$X_n = Y_{AB}Z_{AC} - Y_{AC}Z_{AB}, \tag{2.9}$$

$$Y_n = X_{AB}Z_{AC} - X_{AC}Z_{AB}, \tag{2.10}$$

$$Z_n = X_{AB}Y_{AC} - X_{AC}Y_{AB}. \tag{2.11}$$

2.2.2 Вычисление нормали к сфере

Для вычисления нормали к сфере в заданной точке необходимо вычесть координаты центра сферы из координат точки поиска:

$$n = \{p_x - o_x, p_y - o_y, p_z - o_z\}, \tag{2.12}$$

где $\vec{n}$ – вектор нормали, p – точка, к которой ищется нормаль, o – центр сферы. Нормализовать данный вектор можно поделив каждую из координат на радиус сферы.

2.3 Поиск пересечения с объектами

Положение луча в пространстве спустя некое время t можно описать следующим уравнением:

$$\vec{r} = o + \vec{d} \cdot t, \tag{2.13}$$

где o – начальная точка луча, $\vec{d}$ – единичный вектор, задающий направление. Можно записать это в другом виде:

$$x(t) = x_0 + tx_d, \tag{2.14}$$

$$y(t) = y_0 + ty_d, \tag{2.15}$$

$$z(t) = z_0 + tz_d. \tag{2.16}$$

2.3.1 Поиск пересечения со сферой

Сфера задается уравнением вида:

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = r^2, \tag{2.17}$$

где c – центр сферы. Вместо (x, y, z) выполним подстановку уравнений луча:

$$(x_0 + tx_d - x_c)^2 + (y_0 + ty_d - y_c)^2 + (z_0 + tz_d - z_c)^2 = r^2. \tag{2.18}$$

После раскрытия скобок и ряда преобразований получается квадратное уравнение:

$$At^2 + Bt + C = 0, \tag{2.19}$$

где коэффициенты A, B и C принимают значения:

$$A = x_d^2 + y_d^2 + z_d^2, \tag{2.20}$$

$$B = 2(x_d(x_0 - x_c) + y_d(y_0 - y_c) + z_d(z_0 - z_c)), \tag{2.21}$$

$$C = (x_0 - x_c)^2 + (y_0 - y_c)^2 + (z_0 - z_c)^2 - r^2. \tag{2.22}$$

Коэффициент $A = 1$, так как вектор $\vec{d}$ является единичным.

Если дискриминант уравнения меньше нуля, то луч не пересекает сферу, иначе наименьший положительный корень будет расстоянием от центра сферы до точки пересечения.

При обнаружении двух пересечений, программа будет выбирать ближайшее к наблюдателю. Таковое определяется путем сравнения параметров t. Ближней точке пересечения соответствует меньший параметр.

Если одно или оба значения получились отрицательными, это говорит о том, что точка пересечения лежит «сзади» относительно начала луча. Такие точки отбрасываются при построении, так как не принадлежат картинной плоскости.

2.3.2 Поиск пересечения с треугольником

Для поиска пересечения используется алгоритм Моллера-Трумбора [4], работающий оптимально по скорости и по сей день.

Данный алгоритм использует параметризацию точки пересечения P с помощью барицентрических координат:

$$P = w\vec{A} + u\vec{B} + v\vec{C}. \tag{2.23}$$

Из равенства (2.24) можно вывести следующие уравнения:

$$w = 1 - u - v, \tag{2.24}$$

$$P = (1 - u - v)\vec{A} + u\vec{B} + v\vec{C}, \tag{2.25}$$

$$P = \vec{A} + u(\vec{B} - \vec{A}) + v(\vec{C} - \vec{A}), \tag{2.26}$$

где $(\vec{B} - \vec{A})$ и $(\vec{C} - \vec{A})$ являются сторонами треугольника ABC. Последнее равенство можно преобразовать, заменив левую часть на уравнение (2.13):

$$o + t\vec{do} - \vec{A} = \vec{A} + u(\vec{B} - \vec{A}) + v(\vec{C} - \vec{A}) - t\vec{d} + u(\vec{B} - \vec{A}) + v(\vec{C} - \vec{A}). \tag{2.27}$$

Перепишем полученное уравнение в более удобной форме:

$$[-\vec{d}(\vec{B} - \vec{A})(\vec{C} - \vec{A})] \begin{bmatrix} t \\ u \\ v \end{bmatrix} = o - \vec{A}. \tag{2.28}$$

Можно выразить положение P в пространстве t, u, v, где t указывает

расстояние от точки пересечения до начала луча и параллельно оси t. Если P лежит в треугольнике, то $0 \leq u, v \leq 1, u + v \leq 1$.

Применив метод Крамера, получим:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{[|-\vec{d}\vec{e_1}\vec{e_2}|]} \begin{bmatrix} |\vec{T} & \vec{e_1} & \vec{e_2}| \\ |-\vec{d} & \vec{T} & \vec{e_2}| \\ |-\vec{d} & \vec{e_1} & \vec{T}| \end{bmatrix}, \tag{2.29}$$

где $\vec{T} = o - \vec{A}, \vec{e_1} = \vec{B} - \vec{A}, \vec{e_2} = \vec{C} - \vec{A}$. Проведя преобразования, можно записать в следующем виде:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{\vec{P} \cdot \vec{e_1}} \begin{bmatrix} \vec{Q} \cdot \vec{e_2} \\ \vec{P} \cdot \vec{T} \\ \vec{Q} \cdot \vec{d} \end{bmatrix}, \tag{2.30}$$

где $\vec{P} = (\vec{d} \times \vec{e_2}), \vec{Q} = (\vec{T} \times \vec{e_1})$. Из (2.30) может быть получено необходимое значение t.

2.4 Алгоритм обратной трассировки

На входе алгоритма имеется положение наблюдателя, направление его взгляда, угол обзора, описание сцены (присутствующие объекты и их характеристики), описание источника света. Изображение формируется трассировкой лучей через каждый пиксель изображения и установкой возвращаемого лучом цвета.

Для расчета цвета необходимо найти ближайшую точку пересечения с объектом сцены, вычислить нормаль к нему в данной точке и рассчитать интенсивность по формуле (1.5). В случае зеркальной поверхности луч отражается и яркость, возвращаемся отраженным лучом, суммируется с яркостью в данной точке.

Для каждого пикселя задается луч *ray* из камеры, и цвет пикселя вычисляется с помощью функции *trace(ray)*. Рассмотрим этапы ее выполнения.

1. Поиск точки ближайшего пересечения луча с объектом сцены.

2. Если такая точка существует, то вызывается функция *shade(point,*

normal), где *point* – точка пересечения, *normal* – нормаль к поверхности в данной точке. Иначе цвет пикселя = цвет фона и переход к следующему пикселю.

Рассмотрим этапы выполнения функции *shade(point, normal)*. Изначально цвет пикселя = (0, 0, 0) и осуществляется следующий расчет для каждого источника света.

1. Испустить теневой луч из точки пересечения к источнику света.
2. Если луч не пересекает объект сцены, то цвет = цвет * прямое освещение от источника.
3. Если поверхность пересечения зеркальная, то если поверхность не является идеальным зеркалом, то внести произвольное отклонение в отраженный луч, тогда цвет = цвет + *trace(отраженный луч)*.

Чтобы выполнение алгоритма занимало меньше времени, можно использовать потоки для параллельности вычислений. Каждый поток может обрабатывать горизонтальную полосу, толщина которой в пикселях определяется количеством потоков:

$$w = height/n\_threads, \tag{2.31}$$

где $height$ – высота изображения в пикселях, $n\_threads$ – количество потоков.

2.5 Структуры данных

Рассмотрим необходимые структуры данных. Для хранения геометрических объектов сцены будет использован динамический массив, так как количество элементов не велико, а перебор всех объектов будет достаточно частой операцией.

Для хранения сферы необходимы координаты центра и радиус. Остальные геометрические объекты будут храниться в виде массива полигонов, описываемых тремя вершинами. Для оптимизации вычислений можно также хранить нормаль к полигонам.

Об объектах также необходимо хранить цвет в формате RGB-тройки и характеристики материала, от которых зависит размытость изображения.

2.6 Вывод из раздела

В данном были описаны необходимые алгоритмы и математические основы для их выполнения, а также выбраны необходимые структуры данных.

3. Технологический раздел

В данном разделе программирования, будут выбраны сконструированы среда разработки используемые классы, и а язык также спроектирован интерфейс программного продукта.

3.1 Выбор и обоснование языка программирования и среды разработки

В качестве языка программирования выбран язык C++, так как он обладает рядом преимуществ среди которых:

- поддержка технологии объектно-ориентированного программирования, позволяющая:
  - использовать наследование, полиморфизм и прочие возможности;
  - представлять объекты сцены как объекты класса, что упростит организацию взаимодействия между ними и повысит читабельность;
  - ускорить отрисовку сцены;
- возможность использования широкого ряда шаблонов и библиотек;
- имеющийся опыт работы с ним в ранних курсах.

В качестве среды разработки выбран QtCreator, достоинствами которого являются:

- простота в использовании;
- возможность создавать интерфейс и вести разработку внутри одного приложения;
- имеющийся опыт работы в данной среде.

3.2 Структура и состав классов

На рисунке 3.1 представлена диаграмма классов.

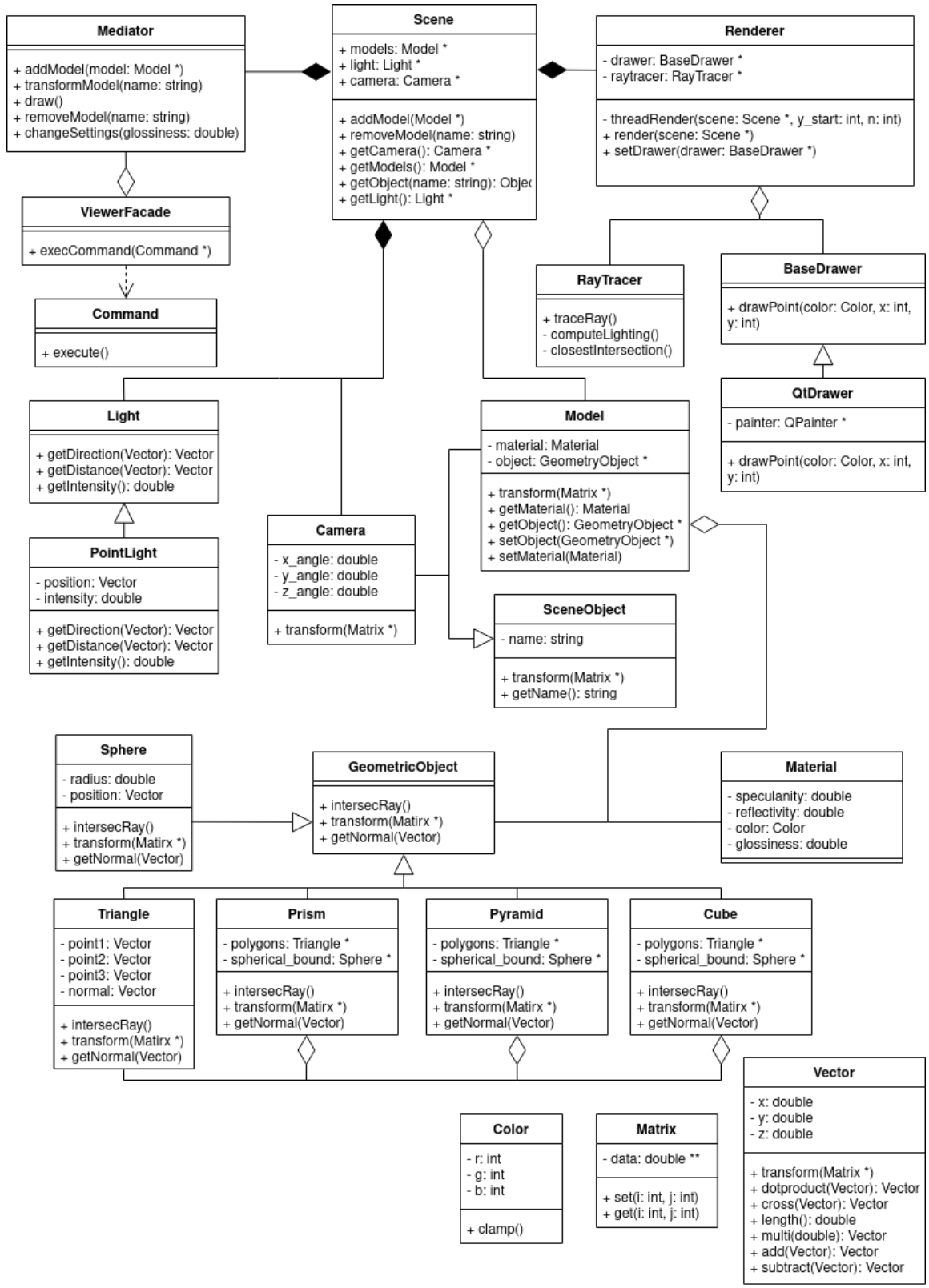

Рисунок 3.1 – Диаграмма классов программы

Краткое описание классов:

- ViewerFacade – класс для взаимодействия пользовательского интерфейса и вычислительной части программы;
- Command – абстракция команды пользовательского интерфейса, позволяет изолировать графический интерфейс от остальной программы;
- Mediator – класс, упрощающий взаимодействие классов между собой, снижает межмодульную зависимость;
- Renderer – класс для отрисовки изображения;
- BaseDrawer – класс-обертка над QtDrawer, обеспечивает гибкую подмену используемых библиотек;
- RayTracer – класс, реализующий алгоритм обратной трассировки лучей.

Для хранения информации о сцене и ее объектах были выделены следующие классы:

- Scene – класс, хранящий информацию о сцене;
- Light – базовый класс источников света, позволит при необходимости добавлять различные типы источников освещения;
- PointLight – класс точечного источника освещения;
- Camera – класс, хранящий информацию о камере и позволяющий управлять ею;
- Model – класс для хранения информации об объекте сцены;
- Material – класс, хранящий информацию о материале объекта;
- Color – класс, хранящий информацию о цвете объекта в формате RGB-тройки.

Также следует отдельно выделить математические классы:

- GeometryObject – базовый класс геометрических фигур, позволяет однообразно обрабатывать все типы геометрических объектов сцены;
- Sphere, Prism, Cube, Pyramid – классы для представления конкретных геометрических объектов;
- Vector, Matrix, Triangle – вспомогательные классы для упрощения работы с трехмерными объектами.

3.3 Интерфейс программы

На рисунке 3.2 представлено главное окно программы, содержащее следующие поля:

- поля ввода для изменения координат и интенсивности источника света, которая может лежать в интервале от 0 до 1;
- поле ввода диффузной составляющей способности зеркала (неотрицательная величина);
- поля изменения направления и координат положения камеры;
- поле ввода количества лучей, обрабатываемых для отображения одного пикселя поверхности с коэффициентом диффузности отражения, отличным от 0.

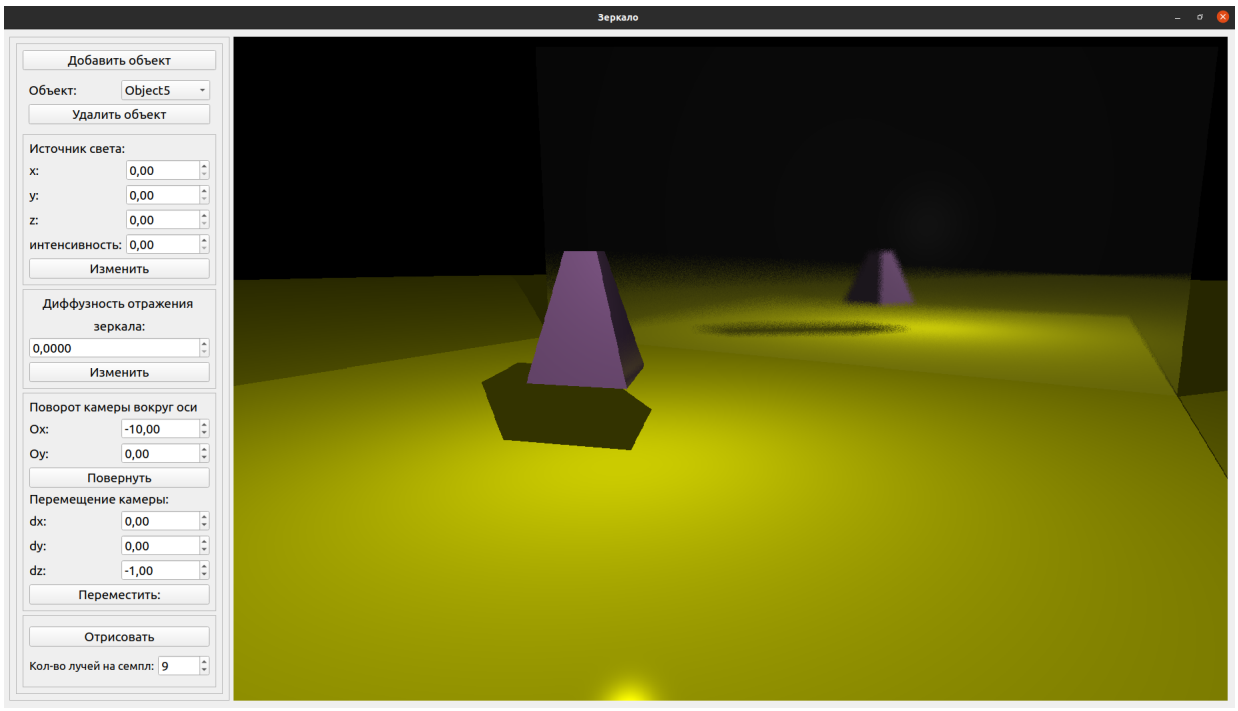

Рисунок 3.2 – Интерфейс главного окна программы

Окна добавления геометрических объектов содержат следующие поля для ввода атрибутов:

- сфера – координаты центра и радиус;
- куб – координаты левой нижней ближайшей вершины, длина ребра и угол поворота вокруг оси Oy;
- усеченная пирамида – координаты левой нижней ближайшей вершины, длина основания, высота неусеченной версии пирамиды и высота усеченной (если исходная высота оказывается меньше усеченной, строится обычная четырехгранная пирамида высотой, равной наименьшему из двух значений), угол поворота вокруг оси Oy;
- призма – координаты левой нижней ближайшей вершины, длина основания, высота призмы, угол поворота вокруг оси Oy.

Также каждое окно содержит поля для ввода свойств материала поверхности объектов: цвет, зеркальность, коэффициент отражения и шероховатость.

На рисунке 3.3 представлено окно добавления куба.

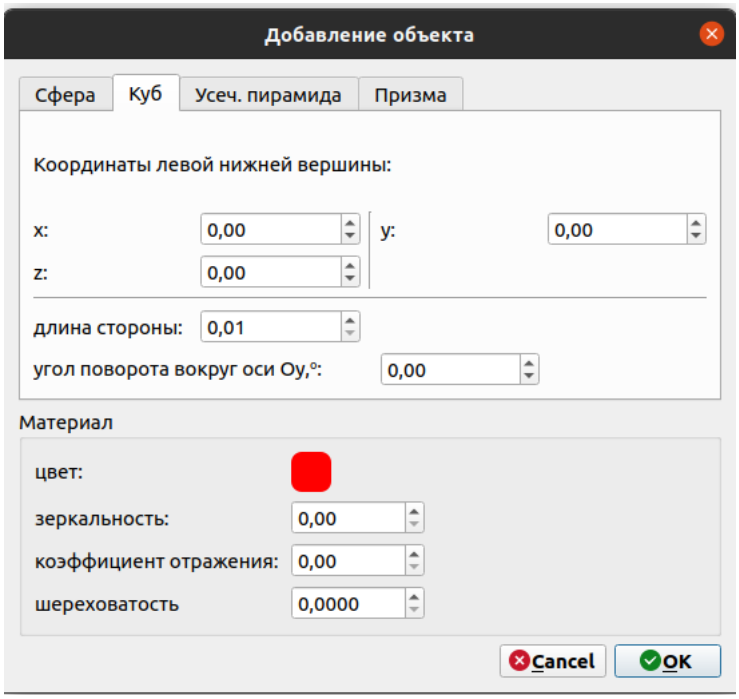

Рисунок 3.3 – Окно добавления куба

3.4 Вывод из раздела

В данном разделе были выбраны язык программирования и среда разработки, разработана структура классов, а также интерфейс главного окна и окон добавления геометрических объектов.

4. Экспериментальный раздел

4.1 Цель эксперимента

Целью эксперимента является проверка правильности выполнения поставленной задачи, оценка эффективности при многопоточной реализации алгоритма обратной трассировки лучей.

4.2 Апробация

На рисунках 4.1 – 4.2 представлена одна и та же сцена с различных ракурсов. Перемещение камеры и отображение теней не исказилось.

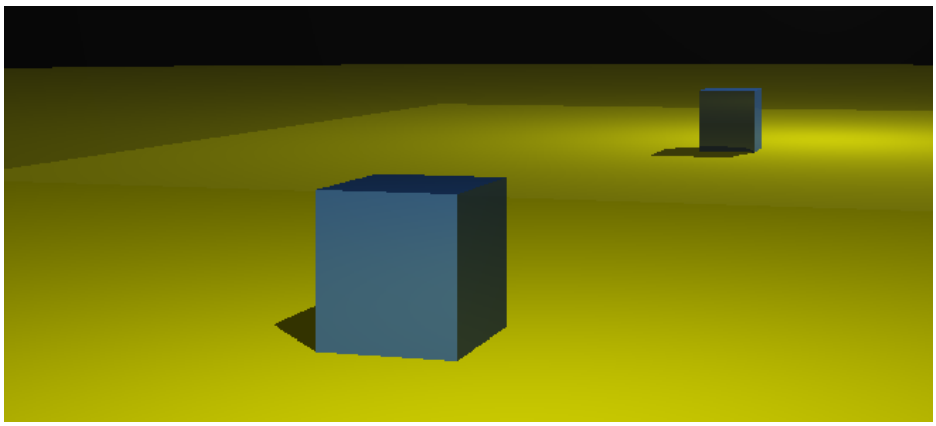

Рисунок 4.1 – Проверка камеры и теней - 1

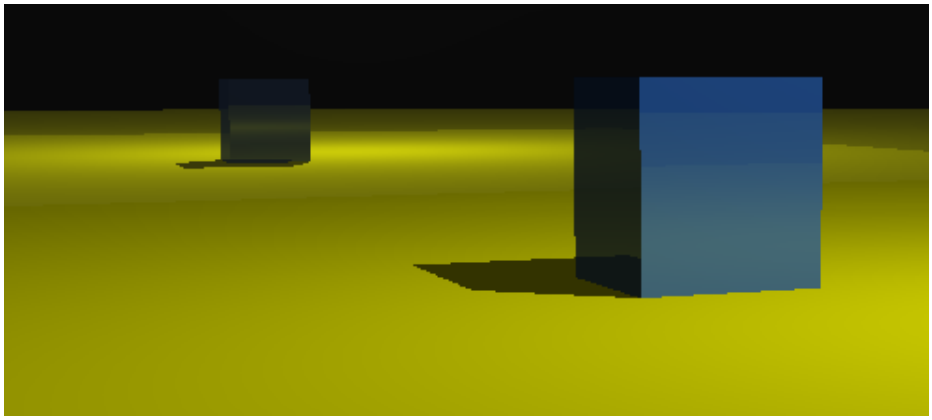

Рисунок 4.2 – Проверка камеры и теней - 2

На рисунках 4.3 – 4.4 представлена одна и та же сцена при разных положениях источника света. Изменение его положения работает корректно.

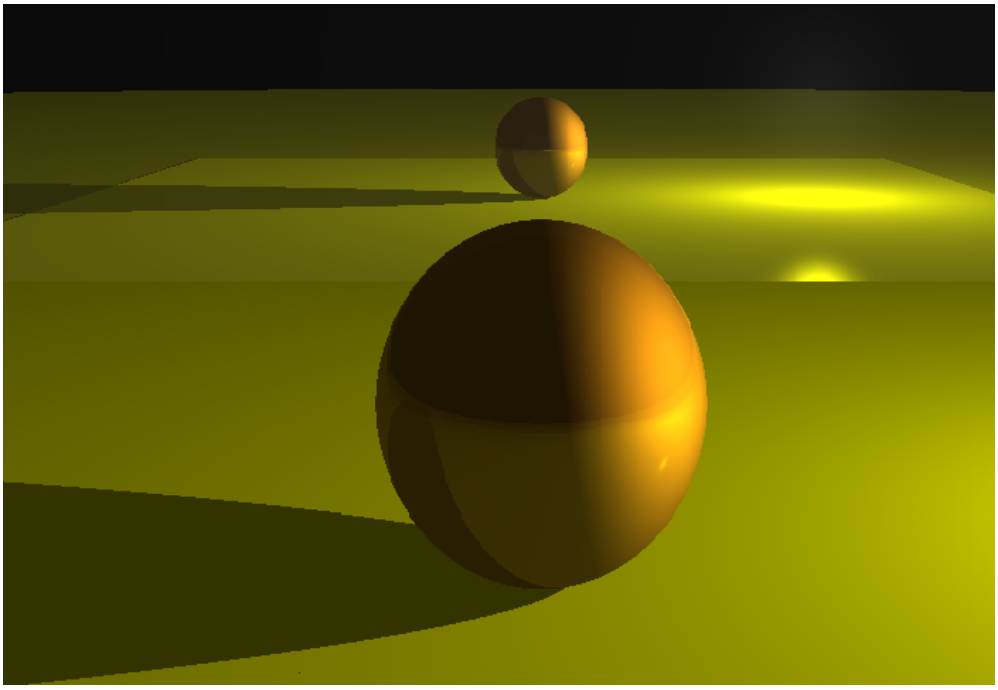

Рисунок 4.3 – Проверка освещения - 1

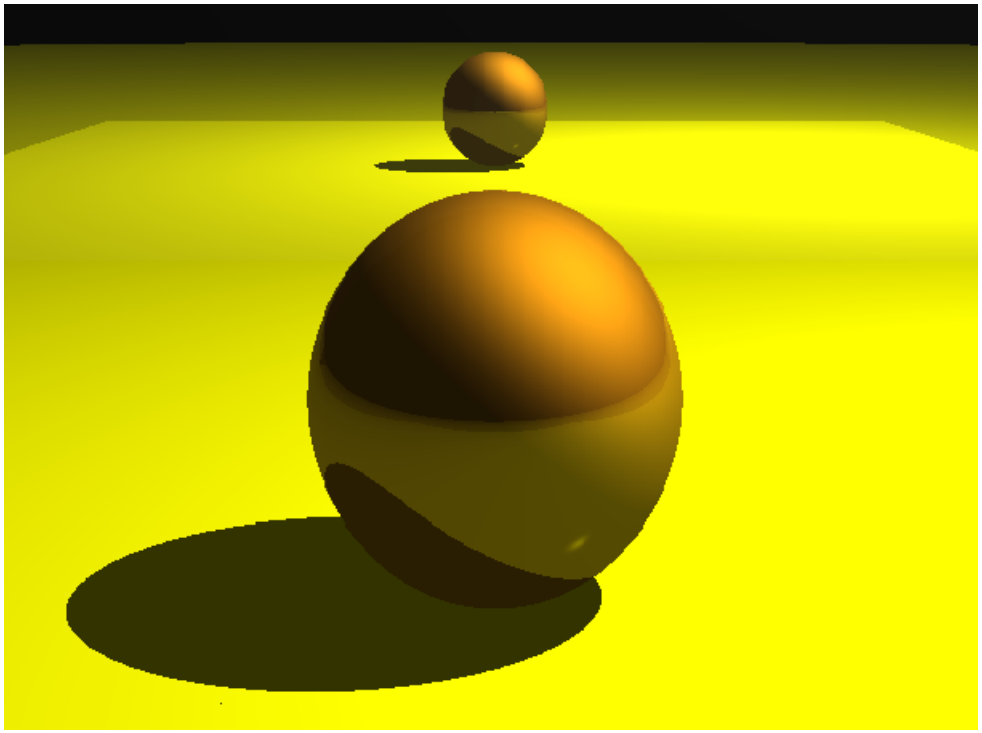

Рисунок 4.4 – Проверка освещения - 2

На рисунках 4.5 – 4.6 представлена сцена с разными значениями диф-

фузной составляющей покрытия зеркала. На первом рисунке она равна 0, на втором – 0.3. В случае ненулевой диффузной составляющей отражение становится нечетким, ребра перестают быть четкими в отражении, сильно проявляется гранулярность.

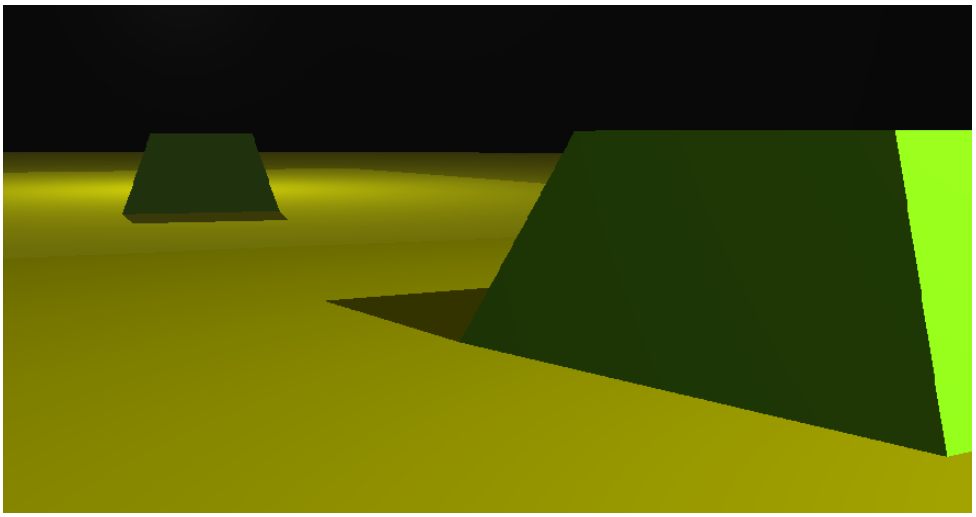

Рисунок 4.5 – Проверка диффузной составляющей зеркала - 1

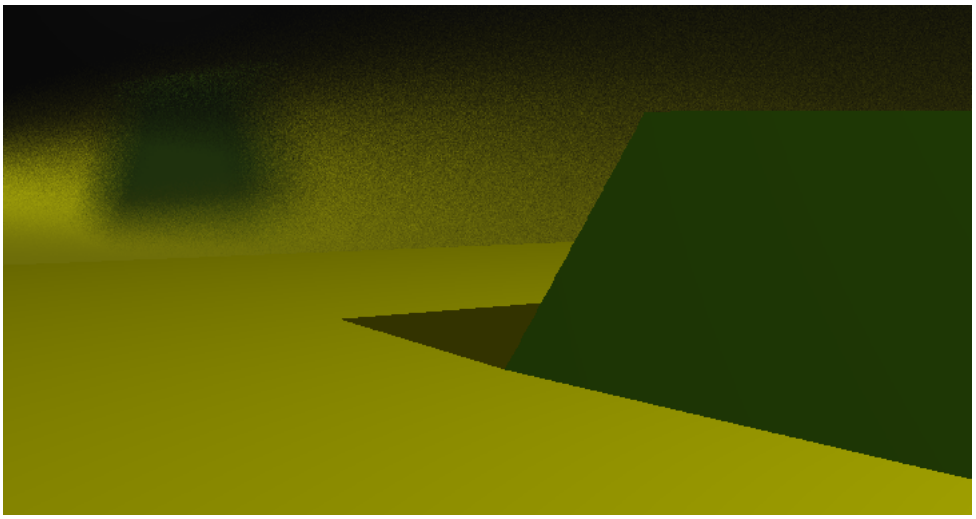

Рисунок 4.6 – Проверка диффузной составляющей зеркала - 2

4.3 Описание эксперимента

Была замерена скорость построения изображения с двумя сферами, одна из которых имеет коэффициент отражаемости 0.7 и степень шершавости

0.05, другая с нулевым коэффициентом отражаемости, призмой и усеченной пирамидой, обе из них со значением 0.2 зеркальности, а также куб с коэффициентом отражаемости 0.4 и степенью шершавости 0.03. Для реализации многопоточной обработки были использованы средства стандартной библиотеки C++ std::thread [5].

Замеры проводились на системе со следующими характеристиками:

- операционная система Ubuntu 20.04.3 LTS [6];
- память 16 Гб;
- процессор Intel Core i5-1135G7 11th Gen, 2.40 Гц [7].

На рисунке 4.7 представлен график зависимости времени выполнения обратной трассировки лучей при соответствующем количестве используемых потоков.

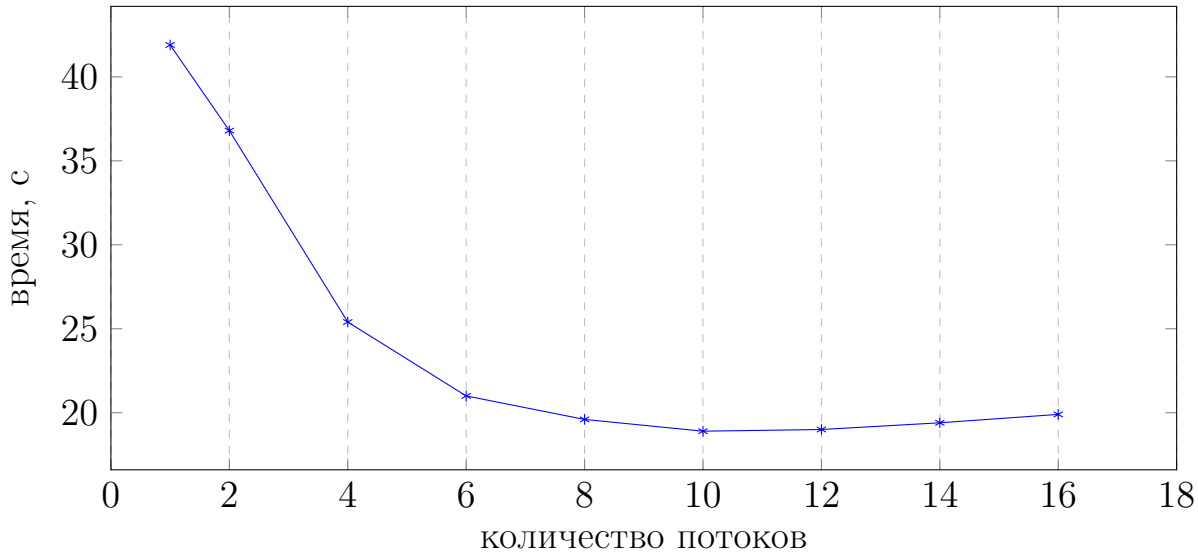

Рисунок 4.7 – Временная зависимость построения от количества используемых потоков

Лучшее время достигается при распределении выполнения отрисовки на 10 потоках, производительность относительно однопоточной реализации выросла на 55%. Дальнейшее увеличение числа потоков приводит к слишком частой смене контекста, что замедляет обработку сцены.

4.4 Выводи из раздела

В данном раздела была проведена проверка на корректность выполнения функций изменений объектов сцены, таких как камера, источник освещения и плоское зеркало. Также было установлено оптимальное число используемых потоков для реализации алгоритма обратной трассировки лучей.

Заключение

В ходе выполнения данной работы были выполнены следующие задачи:

- рассмотрены и выбраны алгоритмы трехмерной графики для визуализации объектов и построения их отражения в зеркале;
- изучена работа выбранных алгоритмов;
- спроектирована архитектуру программы и её интерфейс;
- реализованы выбранные алгоритмы и структуры данных.

В результате был реализован программный продукт для генерации реалистичного изображения сцены с зеркалом и трехмерным геометрическими объектами. Используя графический интерфейс, можно менять набор объектов сцены, характеристики источника света, положение и направление камеры.

Была проведена апробация работы перемещения камеры, изменения источника освещения, а также изменения диффузной составляющей отражения в плоском зеркале. В ходе проведенного эксперимента было установлено, что максимальная производительность достигается при реализации на 10 потоках, а выигрыш в данном случае достигает 55%.

Список литературы

[1] В.В. Селянкин. Алгоритмы трехмерной графкики. – ТТИ ЮФУ, 2007. – 100 с.

[2] Д. Роджерс. Алгоритмические основы машинной графики: Пер. с англ. – Мир, 1989. – 512 с.

[3] Е.А. Никулин. Компьютерная геометрия и алгоритмы машинной графики. – БВХ-Петербург, 2003. – 479 с.

[4] T. Möller B. F. T. Minimum Storage Ray/Triangle Intersection // Journal of Graphics Tools. 1972. Vol. 2, no. 1. P. 21–28.

[5] C++ std::thread [Электронный ресурс]. Режим доступа: `https://en.cppreference.com/w/cpp/thread/thread` (дата обращения: 12.01.2022).

[6] Ubuntu 20.04 LTS [Электронный ресурс]. Режим доступа: `https://releases.ubuntu.com/20.04/` (дата обращения: 27.01.2022).

[7] Процессор Intel Core i5-1135G7 [Электронный ресурс]. Режим доступа: `https://www.intel.ru/content/www/ru/ru/products/sku/208658/intel-core-i51135g7-processor-8m-cache-up-to-4-20-ghz/specifications.html` (дата обращения: 27.01.2022).