СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В области компьютерной графики, значительное внимание уделяется разработке алгоритмов для создания более реалистичных изображений. Однако, по мере усложнения самих алгоритмов, возрастают требования к вычислительным ресурсам системы. Эта исследовательская работа актуальна, поскольку она стремится найти оптимальные алгоритмы, которые могут быть использованы для аппаратного создания реалистичных изображений, с учетом ограниченных ресурсов компьютерной системы.

Цель курсовой работы: разработать программу моделирования блочного конструктора из перечня геометрических объектов: куб, шар, параллелепипед, наклонная усеченая пирамида.

Для достижения цели, требуется выполнить следующие задачи:

1) проанализировать имеющиеся алгоритмы и определить оптимальные методы для решения основной задачи;

2) разработать эффективную структуру программы;

3) выбрать наиболее подходящий язык программирования и интегрированную среду разработки для выполнения задачи;

4) создать программный продукт для решения задачи, реализовать выбранные алгоритмы;

5) создать понятный пользовательский интерфейс;

6) провести исследования, основанные на полученных результатах.

1 Аналитическая часть

В данном разделе предоставим описание трехмерной модели объектов в сцене, рассмотрим формализацию сценариев и критерии, которым должна соответствовать программа. Также исследуем методы создания трехмерных изображений и моделирования освещения.

1.1 Описание модели трёхмерного объекта в сцене

В данной работе трехмерный объект в сцене описывается с использованием полигональной сетки.

Полигональная сетка – это совокупность вершин, рёбер и граней, которые определяют форму многогранного объекта в трёхмерной компьютерной графике и объёмном моделировании [1] . Гранями обычно являются треугольники, четырёхугольники или другие простые выпуклые многоугольники (полигоны).

Этот метод является универсальным, так как позволяет описывать трехмерные объекты в различных формах. Однако количество полигонов существенно влияет на реалистичность визуализации модели, но при этом обработка большого числа граней требует больших вычислительных ресурсов.

1.2 Формализация объектов синтезируемой сцены

Сцена состоит из:

1) источников света – в сфере создания трехмерных изображений представляют собой материальные точки, из которых исходят лучи света во все стороны. В некоторых случаях источник света может быть расположен в бесконечности и иметь определенную направленность;

2) объектов – в сфере создания трехмерных изображений представляют собой геометрической фигуры, которые касаются между собой, из которых строится модель;

3) камера – характеризуется своим пространственным положением и направлением просмотра.

1.3 Анализ алгоритмов построения трёхмерного изображения

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

1.3.1 Алгоритм, использующий Z - буфер

Алгоритм Z - буфера — один из простейших алгоритмов, который работает в пространстве изображения [1].

Буфер кадра выполняет функцию заполнения пикселей в изображении атрибутами интенсивности. Для этой задачи требуется буфер регенерации, в котором сохраняются яркостные значения, а также Z - буфер (буфер глубины), который хранит информацию о координатах z для каждого пикселя. В начале процесса Z - буфер заполняется минимальными значениями Z, а буфер регенерации заполняется фоновыми значениями. Затем многоугольники преобразуются в растровую форму и записываются в буфер регенерации. Важно отметить, что многоугольники не сортируются в начале процесса.

В ходе выполнения каждый новый пиксель сравнивается с пикселем в Z - буфере, и если новый пиксель находится ближе к наблюдателю, чем тот, который уже есть в буфере кадра, то новый пиксель перезаписывает существующий в буфере. Значение Z - буфера также корректируется, чтобы учитывать глубину нового пикселя. Если новый пиксель имеет меньшее значение Z, чем пиксель в буфере, то никакие действия не выполняются.

На рисунке 1.1 представлена иллюстрация работы алгоритма Z - буфера [2].

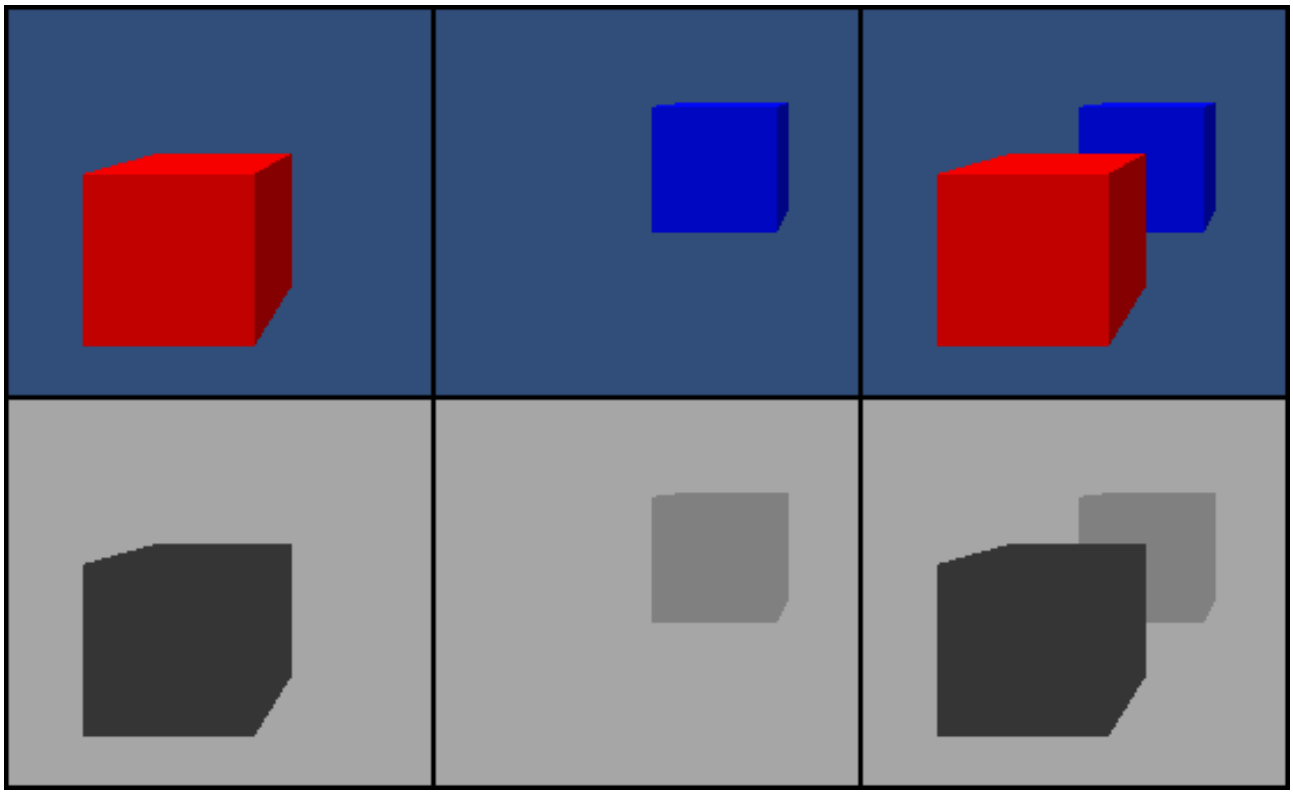

Рисунок 1.1 – Иллюстрация работы алгоритма Z - буфера

Преимущества этого алгоритма:

- Простота. Сцены могут быть любой сложности.
- Элементы сцены не нужно сортировать.

Недостатки:

- Большой объём памяти.
- Трудоёмкость устранения лестничного эффекта.
- Трудоёмкость реализации эффектов прозрачности и просвечивания.

1.3.2 Алгоритм обратной трассировки лучей

Алгоритм предлагает следующий подход: для каждого пикселя на изображении мы проводим луч, начинающийся от камеры, и программа должна

определить, где этот луч пересекается со сценой. Этот исходный луч называется первичным лучом [3].

На рисунке 1.2 показан пример, когда первичный луч пересекает объект и достигает точки H1:

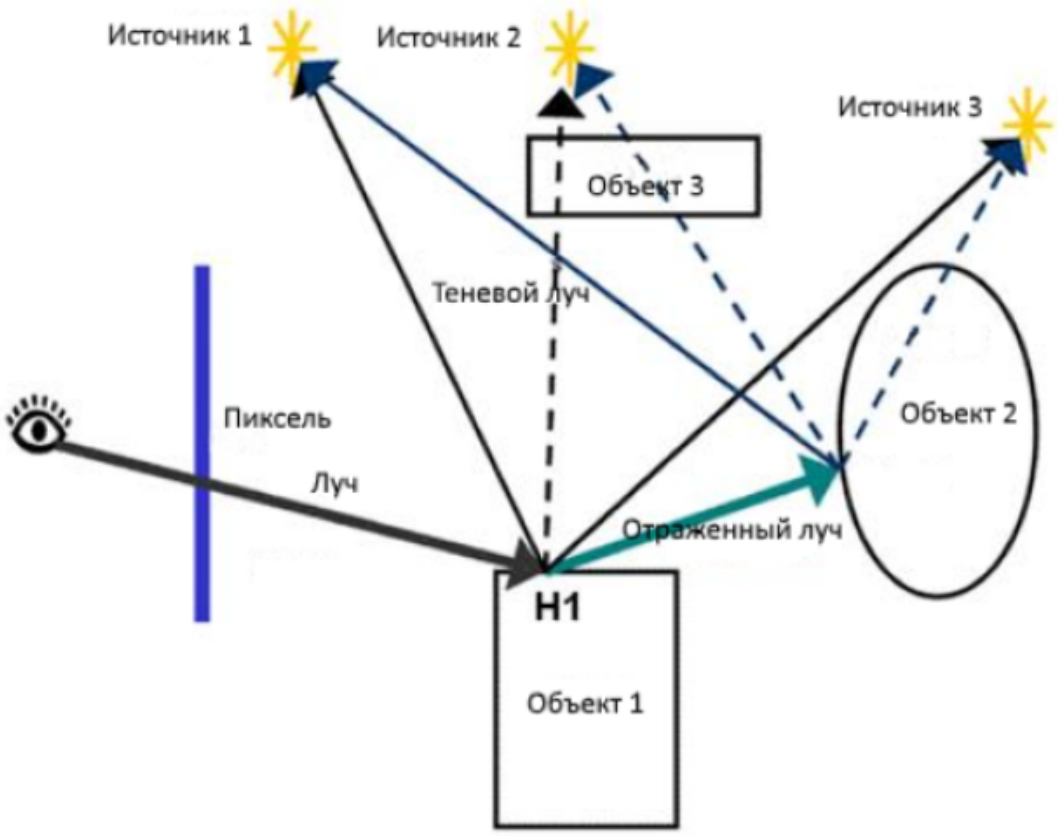

Рисунок 1.2 – Иллюстрация работы алгоритма обратной трассировки лучей

Алгоритм начинает с проверки видимости точки для источника света. Для этого он выпускает теневой луч из этой точки к источнику света. Если луч пересекает какой-либо объект сцены, это означает, что точка находится в тени и не нуждается в освещении. В противном случае, программа определяет степень освещенности точки.

Затем алгоритм анализирует отражающие и преломляющие свойства объекта. Если такие свойства присутствуют, из точки H1 испускается отраженный луч, и процесс повторяется рекурсивно. То же самое происходит при рассмотрении явления преломления. Этот алгоритм эффективно решает поставленную задачу и обеспечивает высокую степень реалистичности полученного изображения. Кроме того, он учитывает все физические явления, такие как отражение, преломление и создание теней [3].

Преимущества этого алгоритма:

- Высокая степень реализма. Метод обратной трассировки лучей позволяет создавать изображения с высокой степенью детализации и реализма, включая отражения, преломления, тени и другие эффекты.
- Гибкость. Метод обратной трассировки лучей позволяет создавать различные типы изображений, включая статические и анимированные, а также работать с различными типами материалов и источников света.
- Возможность использования параллельных вычислений. Метод обратной трассировки лучей может быть эффективно распараллелен на множество процессоров или ядер, что позволяет ускорить процесс создания изображения.

Недостатки:

- Высокая вычислительная сложность. Метод обратной трассировки лучей требует большого количества вычислительных ресурсов, что может привести к длительным временам обработки и высоким затратам на оборудование.
- Проблемы с прозрачностью. Метод обратной трассировки лучей может столкнуться с проблемами при работе с прозрачными объектами, такими как стекло или вода, которые могут создавать шум и артефакты на изображении.

Вывод

Был выбран алгоритм обратной трассировки лучей с целью исключения невидимых линий и поверхностей. Этот подход обеспечивает создание изображений с высокой степенью реалистичности, учитывая различные физические и оптические явления, и также предоставляет возможность взаимодействия с телами, поддерживающими вращение.

1.4 Анализ алгоритмов моделирования освещения

Реалистичность изображения сильно зависит от правильного выбора алгоритма освещения. Существует две основные группы моделей освещенности: локальные и глобальные. Локальные модели уделяют внимание только первичным источникам света, в то время как глобальные модели также учитывают физические явления, такие как отражение света от поверхностей и преломление света.

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

1.4.1 Модель Ламберта

В этой модели рассматривается идеальное диффузное освещение, где свет, попавший на поверхность, рассеивается равномерно во всех направлениях. При расчетах учитываются только ориентация поверхности (представленная нормалью $\overline{N}$) и направление света (представленное вектором $\overline{L}$). Обозначим I_d как рассеянную составляющую освещенности в конкретной точке, K_d как свойство материала отражать рассеянное освещение, и I_0 как интенсивность рассеянного освещения.

С учетом этих параметров, интенсивность освещения можно рассчитать по следующей формуле:

$$I_d = K_d(\overline{L}, \overline{N})I_0. \tag{1.1}$$

Модель Ламберта является одной из наиболее простых моделей освещения, и она часто используется в сочетании с другими моделями, так как диффузная составляющая присутствует в большинстве сценариев [4].

1.4.2 Модель Фонга

Модель Фонга разделяет освещенность в каждой точке на три компоненты [4]:

- Фоновое освещение: Это светлое фоновое освещение, которое всегда присутствует и не зависит от источников света. Оно считается постоянным для всей сцены.

- Рассеянный свет: Это освещение, которое зависит от угла, под которым свет падает на поверхность, и угла между нормалью поверхности и вектором, указывающим на источник света. Это отвечает за равномерное освещение поверхности.

- Бликовая составляющая: Это освещение, которое отражает свет от бликовых точек на поверхности. Оно зависит от того, насколько близки вектор отраженного света к вектору, указывающему на наблюдателя.

Свойства источников света определяют мощность каждой из этих компонент, а свойства материала объекта определяют, как материал взаимодействует со светом.

Пусть:

- $\overline{N}$ – вектор нормали к поверхности в точке;

- $\overline{L}$ – падающий луч;

- $\overline{R}$ – отражённый луч;

- $\overline{V}$ – вектор, направленный к наблюдателю;

- k_a – коэффициент фонового освещения;

- k_d – коэффициент диффузного освещения;

- k_s – коэффициент зеркального освещения;

- p – степень, аппроксимирующая пространственное распределение зеркально отражённого света.

Тогда интенсивность света вычисляется с использованием формулы 1.2:

$$I_a = K_a * I_a + K_d(\overline{N}, \overline{L}) + K_s(\overline{R}, \overline{V})^p. \quad (1.2)$$

На рисунке 1.3 показан пример работы модели Фонга:

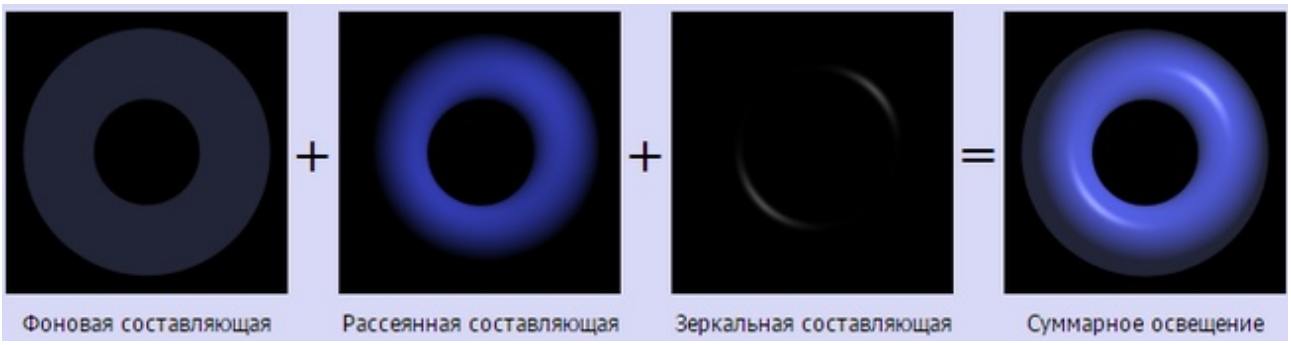

Рисунок 1.3 – Иллюстрация работы модели Фонга

После изучения различных алгоритмов в соответствии с поставленной задачей, принято решение использовать модель Фонга для вычисления интенсивности в точке. Эта модель позволяет учесть как матовые, так и блестящие поверхности.

Вывод

Было приведено описание трёхмерной модели объекта в сцене, а также рассмотрены формализация объектов сцены и необходимые функциональные характеристики программы. Далее были изучены алгоритмы, используемые для создания трёхмерных изображений и модели освещения.

2 Конструкторская часть

В данном разделе сосредоточимся на создании близких к реальности приближений трехмерных объектов. Рассмотрим методы преобразования в трехмерном пространстве, предоставим требования к работе программы, диаграмму классов и схемы алгоритмов, которые разрабатываем.

2.1 Требования к работе программы

Программа должна обеспечить созданию реалистичных изображений. Важно, чтобы пользователь мог легко добавлять объекты в сцену и изменять их характеристики, такие как положение в пространстве, поверхностные свойства и спектральные характеристики, без замедлений.

Для этого программа должна поддерживать два режима работы. В первом режиме акцент сделан на оперативности, чтобы пользователь мог быстро взаимодействовать с объектами. Во втором режиме уделяется внимание созданию реалистичных изображений.

2.2 Аппроксимация трёхмерных объектов

В рамках данной программы создадим икосферу, которая представляет собой геометрическое тело, состоящее из треугольных граней [5]. Преимущество использования икосферы заключается в ее изотропии, что означает, что ее характеристики сохраняются по всем направлениям одинаково. Более того, распределение треугольных граней на икосфере будет более равномерным по сравнению с другими методами разбиения, что предотвратит появление артефактов и особенностей в районе полюсов сферы. Поэтому алгоритм работы программы будет следующим:

- Исходя из данных о сфере, создать регулярный икосаэдр, который включает в себя 20 граней и 30 рёбер.
- Этот процесс повторяется до тех пор, пока не достигнута желаемая степень приближения.

На рисунке 2.1 представлен процесс пошагового приближения икосаэдра [6] к сфере:

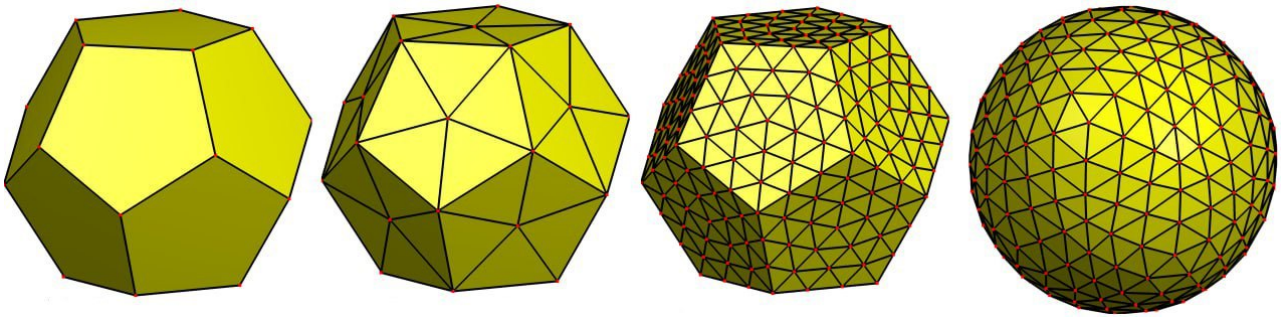

Рисунок 2.1 – Процесс пошагового приближения икосаэдра к сфере

Для корректной генерации икосаэдра важно учитывать коррекцию при вычислении координат средней точки треугольника, обозначенной как (x_0, y_0, z_0). Эта коррекция выполняется согласно формуле 2.1:

$$\begin{aligned} l &= \sqrt{x_0^2 + y_0^2 + z_0^2}; \quad x_1 = \frac{x_0}{l}; \\ y_1 &= \frac{y_0}{l}; \qquad\qquad\quad z_1 = \frac{z_0}{l}. \end{aligned} \tag{2.1}$$

2.3 Описание трёхмерных преобразований

2.3.1 Способ хранения декартовых координат

Для представления координат точек будет использоваться вектор, состоящий из четырёх компонентов: x, y, z, w, где w по умолчанию равен 1. Это позволяет удобно выполнять умножение векторов на матрицы трансформации, которые имеют размерность 4x4.

2.3.2 Преобразование трёхмерных координат в двухмерное пространство экрана

Экран имеет только две координаты, поэтому необходимо разработать метод для отображения трехмерных объектов в двумерном пространстве экра-

на. Каждый пиксель на экране имеет свой цвет, и поэтому требуется найти способ передачи объема и реализма в изображении. Алгоритм преобразования координат включает следующие этапы:

- преобразовать объект из его собственного пространства в мировое пространство;
- перевести объект из мирового пространства в пространство камеры;
- найти проекции всех точек из пространства камеры на видимые точки, где координаты x, y и z находятся в диапазоне от $-w$ до w, а w находится в диапазоне от 0 до 1;
- масштабировать все точки, полученные на шаге 3, для отображения на экране с требуемым разрешением.

Для выполнения всех этих преобразований необходимо применять матрицы преобразований. Сначала рассчитываются все необходимые матрицы, а затем они перемножаются в заданном порядке. Исходные координаты умножаются на конечный результат, что приводит к преобразованию координат в нужную систему.

2.3.3 Преобразования трёхмерной сцены в пространство камеры

Для приведения трехмерной сцены в пространство камеры необходимо умножить каждую вершину всех полигональных моделей на матрицу камеры. Камера сама по себе задается несколькими параметрами: положением в мировом пространстве, вектором направления взгляда и направлением верха камеры. Пусть:

- α – это координаты точки, на которую камера смотрит;
- β – вектор, указывающий, куда направлена верхняя часть камеры;
- γ – вектор, ортогональный векторам направления взгляда и верхнему направлению.

Таким образом, матрица камеры будет иметь следующий вид:

$$B = \begin{pmatrix} \alpha_x & \beta_x & \gamma_x & 0 \\ \alpha_y & \beta_y & \gamma_y & 0 \\ \alpha_z & \beta_z & \gamma_z & 0 \\ -(P * \alpha) & -(P * \beta) & -(P * \gamma) & 1 \end{pmatrix} \tag{2.2}$$

2.3.4 Матрица перспективной проекции

После перехода в камерное пространство, каждая вершина полигональных моделей подвергается умножению на матрицу проекции. Эта матрица изменяет усеченную пирамиду видимости в пространство отсечения и регулирует значение w-компоненты. При этом, чем дальше вершина находится от наблюдателя, тем больше становится значение w. После проецирования координат, значения x и y находятся в интервале от $-w$ до w, а z находится в интервале от 0 до w. Все объекты, которые находятся за пределами этого диапазона, будут отсечены.

Пусть:

- B – это отношение ширины изображения к его высоте;
- ϕ – угол обзора камеры;
- Zb – координата z ближайшей к камере плоскости отсечения в пирамиде видимости;
- Ze – координата z дальней от камеры плоскости отсечения в пирамиде видимости.

Тогда матрица перспективной проекции будет будет представлять собой:

$$A = \begin{pmatrix} \frac{cot(\frac{\phi}{2})}{B} & 0 & 0 & 0 \\ 0 & cot(\frac{\phi}{2}) & 0 & 0 \\ 0 & 0 & \frac{Z_e \times Z_b}{Z_e - Z_b} & 1 \\ 0) & 0 & \frac{Z_e}{Z_e - Z_b} & 0 \end{pmatrix} \tag{2.3}$$

На следующем этапе хотим проецировать все координаты на одну плоскость, путем разделения всех координат на значение z. После умножения вектора координат на матрицу перспективной проекции, реальная координата z перемещается в компоненту w. Вместо деления на z делим на w.

2.3.5 Преобразования трёхмерной сцены в пространство области изображения

Чтобы преобразовать спроецированные координаты в координаты области изображения, мы умножаем вектор координат на специальную матрицу. Пусть:

- W – ширина изображения;
- H – высота изображения;
- hW – половина ширины изображения;
- hH – половина высоты изображения.

Тогда матрица, которая выполняет это преобразование, будет представлять собой:

$$A = \begin{pmatrix} hW & 0 & 0 & hW \\ 0 & hH & 0 & hH \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

2.4 Алгоритм обратной трассировки лучей

На схеме в рисунке 2.2 представлено, как работает алгоритм обратной трассировки лучей.

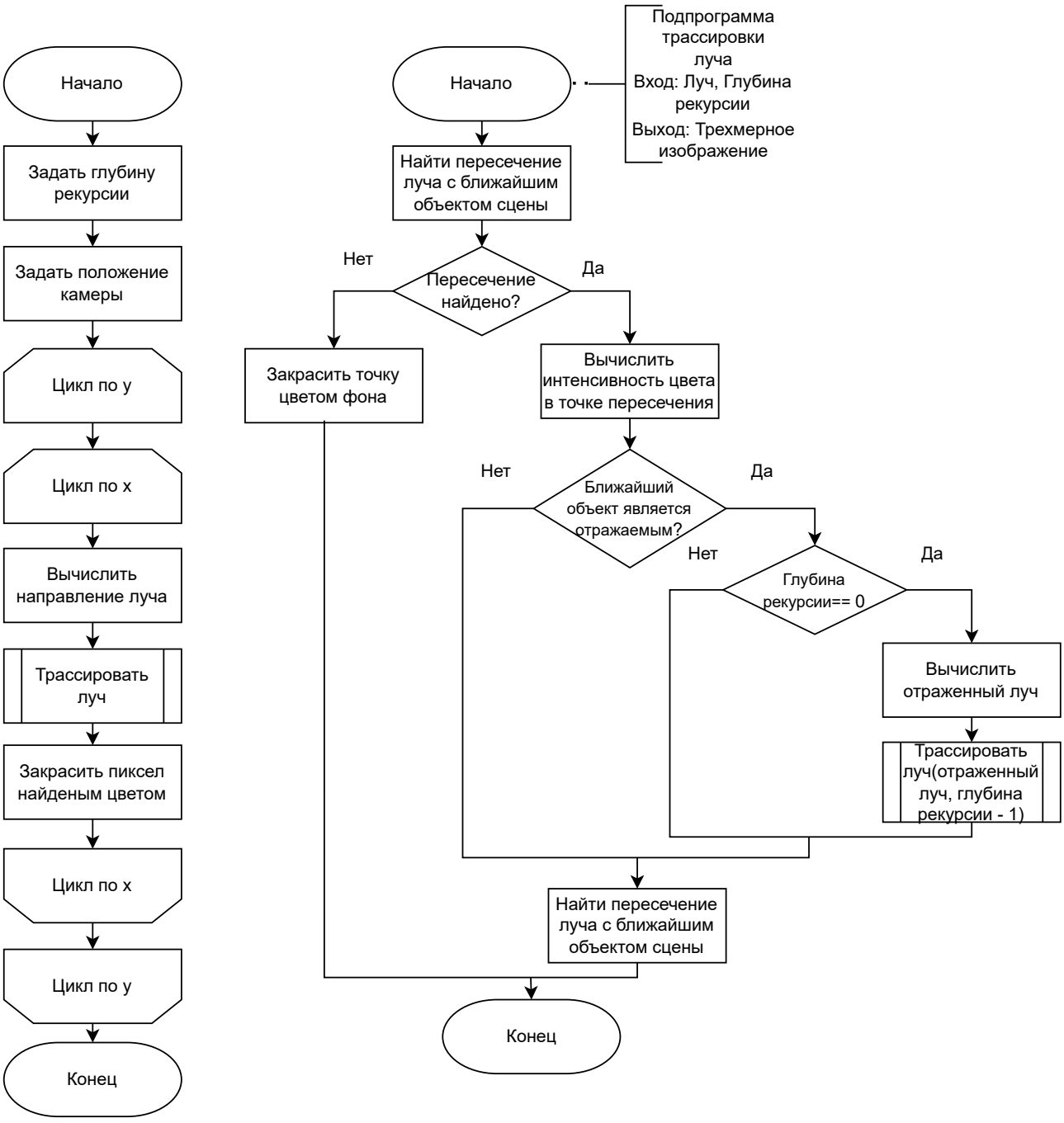

Рисунок 2.2 – Схема реализации алгоритма обратной трассировки лучей

На схеме в рисунке 2.3 представлено, как работает алгоритм Фонга.

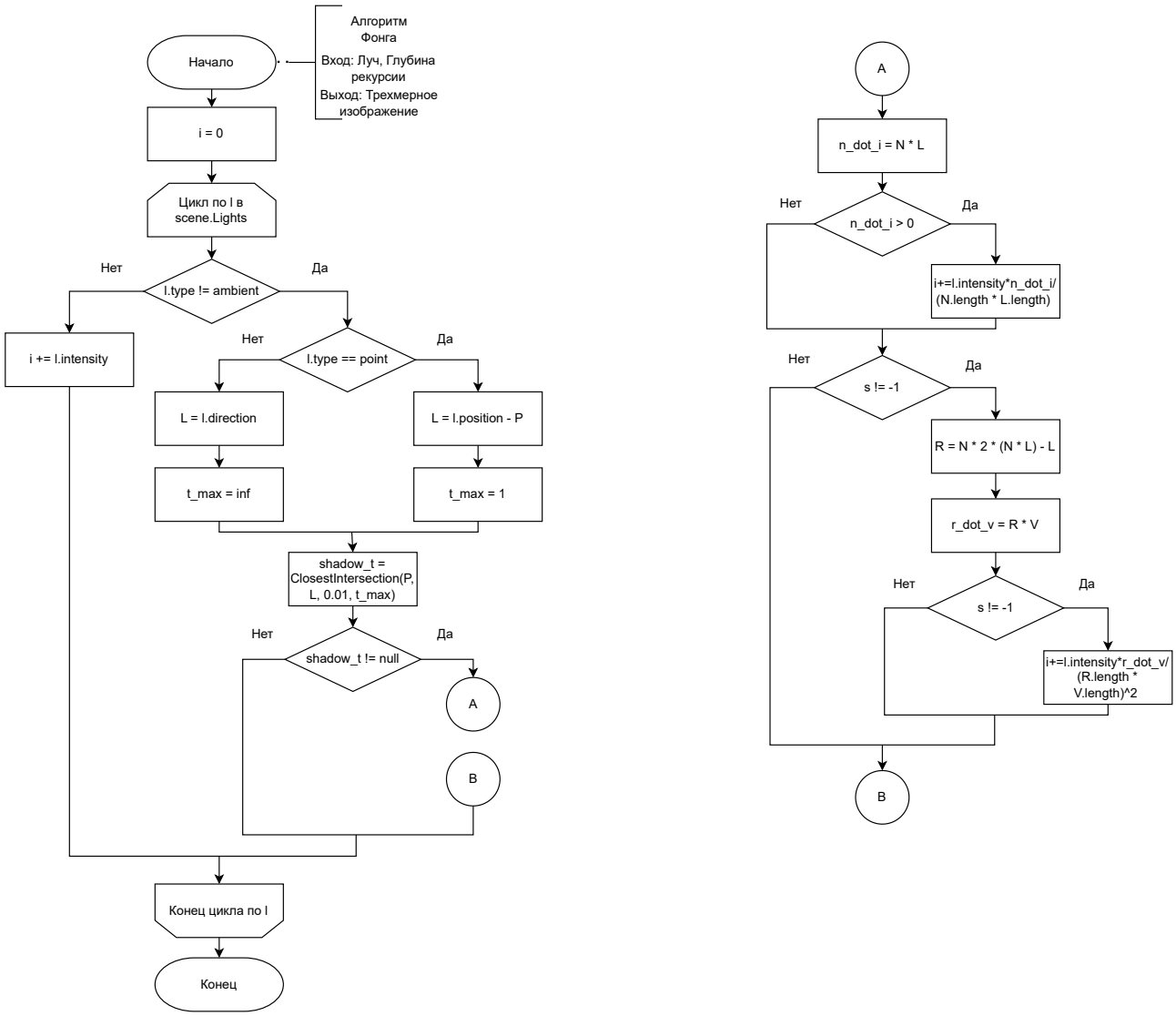

Рисунок 2.3 – Схема реализации алгоритма Фонга

Для эффективной реализации алгоритма обратной трассировки лучей, необходимо уметь вычислять направление отраженных и преломленных лучей, учитывая при этом модель освещения Фонга. Для нахождения отраженного луча, нужно знать направление падающего луча и нормаль к поверхности.

Пусть:

- $\overline{L}$ – направление луча;
- $\overline{n}$ – нормаль к поверхности.

Луч можно разбить на две части: $\overline{L_n}$ которая перпендикулярна нормали, и $\overline{L_n}$ – параллельна нормали.

Иллюстрация этой ситуации изображена на рисунке 2.4:

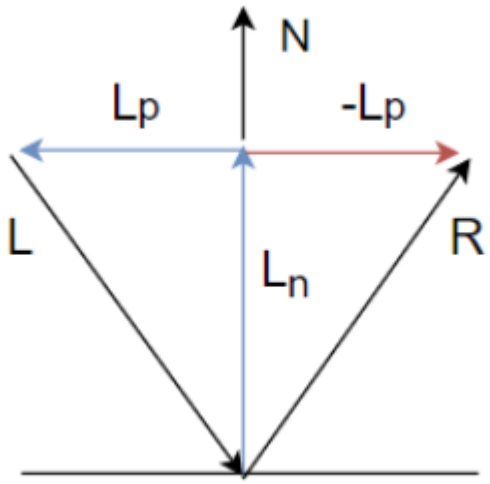

Рисунок 2.4 – Рассматриваемые векторы для расчёта отражённого луча.

Учитывая свойства скалярного произведения $\overline{L_n} = \overline{n} * (\overline{n}, \overline{L})$ и $\overline{L_p} = \overline{L} - \overline{n} * (\overline{n}, \overline{L})$, так как отражённый луч выражается через разность этих векторов, то отражённый луч выражается по формуле 2.5:

$$R = 2 * \overline{n} * (\overline{n}, \overline{L}) - \overline{L}. \tag{2.5}$$

Согласно закону преломления, луч света падающий на поверхность, луч преломлённого света и нормаль к этой поверхности лежат в одной плоскости. Обозначим показатели преломления сред как μ_i, а углы падения и отражения света как η_i. Применяя закон Снеллиуса, можем вычислить параметры преломлённого луча с использованием формулы 2.6:

$$\begin{aligned} R &= \frac{\mu_1}{\mu_2}\overline{L} + (\frac{\mu_1}{\mu_2}cos(\eta_1) - cos(\eta_2))\overline{n}; \\ cos(\eta_2) &= \sqrt{1 - (\frac{\mu_1}{\mu_2})^2 * (1 - cos(\eta_1))^2}. \end{aligned} \tag{2.6}$$

2.5 Алгоритм пересечения луча с параллелепипедом

Для эффективного выполнения алгоритма обратной трассировки лучей, нецелесообразно искать пересечения с каждым полигоном в сцене каждый раз при трассировке луча. Поэтому имеет смысл ограничивать объекты в сцене параллелепипедами, которые полностью их охватывают.

Эти параллелепипеды определяются координатами двух вершин: минимальной и максимальной по значениям координат x, y и z. Это позволяет задать шесть плоскостей, которые ограничивают параллелепипед, и все они параллельны координатным осям.

Рассмотрим пару плоскостей, которые параллельны плоскости yz: $X = x_1$ и $X = x_2$. Предположим, что есть вектор направления луча $\overline{D}$. Если x – координата вектора $\overline{D}$ равна нулю, это означает, что луч параллелен этим плоскостям. В таком случае, если x_0 (начальная x – координата луча) меньше x_1 или больше x_2, луч не пересекает рассматриваемый прямоугольный параллелепипед.

Однако, если $\overline{D_x}$ (проекция вектора $\overline{D}$ на ось x) не равна нулю, мы можем вычислить следующие отношения:

$$\begin{aligned} t_{1x} &= \frac{x_1 - x_0}{\overline{D_x}}; \\ t_{2x} &= \frac{x_2 - x_0}{\overline{D_x}}. \end{aligned} \tag{2.7}$$

Можно считать, что найденные величины связаны неравенством $t_{1x} < t_{2x}$. Пусть $t_n = t_{1x}$, $t_f = t_{2x}$. Считая, что D_y не равно нулю, и рассматривая вторую пару плоскостей, несущих грани заданного параллелепипеда, $Y = y_1$, $Y = y_2$, вычисляются величины:

$$\begin{aligned} t_{1y} &= \frac{y_1 - y_0}{D_y}; \\ t_{2y} &= \frac{y_2 - y_0}{D_y}. \end{aligned} \tag{2.8}$$

Если $t_{1y} > t_n$, то тогда $t_n = t_{1y}$. Если $t_{2y} < t_f$, то тогда $t_f = t_{2y}$. При $t_n > t_f$ или при $t_f < 0$ заданный луч проходит мимо прямоугольного

параллелепипеда.

Считая, что $\overline{D_z}$ не равно нулю, и рассматривая вторую пару плоскостей, несущих грани заданного параллелепипеда, $Z = z_1$, $Z = z_2$, вычисляются величины:

$$\begin{aligned} t_{1z} &= \frac{z_1 - z_0}{\overline{D_z}}; \\ t_{2z} &= \frac{z_2 - z_0}{\overline{D_z}}, \end{aligned} \tag{2.9}$$

и повторяются сравнения, приведённые для формулы 2.6.

Если после выполнения всех операций мы получаем такие значения, что $0 < t_n < t_f$ или $0 < t_f$, то это означает, что исходный луч пересекает ограничивающий параллелепипед, у которого стороны параллельны координатным осям.

Важно учесть, что при внешнем пересечении лучом параллелепипеда, значения t_n и t_f должны быть равны. В противном случае можно заключить, что луч пересекает параллелепипед изнутри.

2.6 Диаграмма классов

На рисунке 2.5 представлена диаграмма классов.

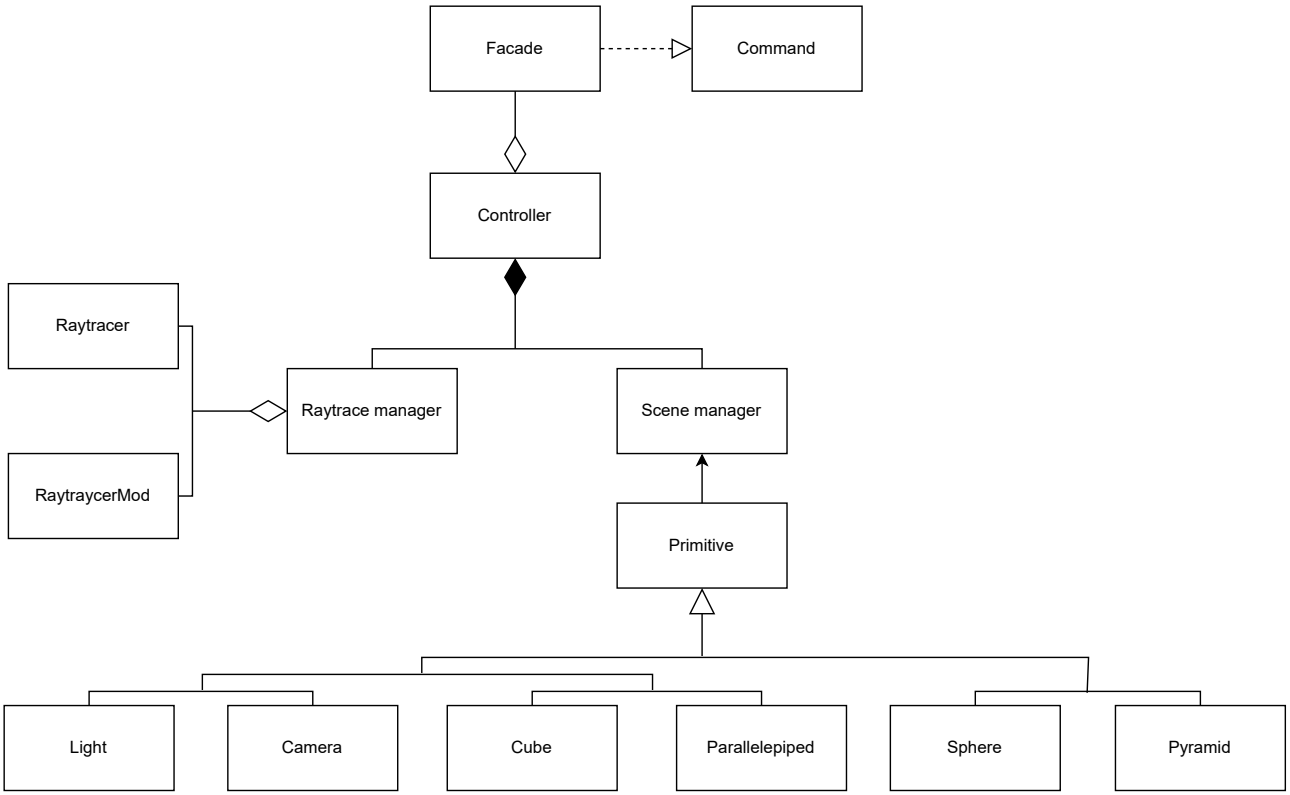

Рисунок 2.5 – Диаграмма классов

Классы объектов:

- Primitive – базовый класс объектов;
- Sphere – класс шара;
- Pyramid – класс пирамиды;
- Cube – класс куба;
- Parallelepiped – класс параллелепипед;
- Camera – класс камеры;
- Light – класс источника света;
- Facade – класс, предоставляющий интерфейс работы системы;

- Controller - класс, который позволяет взаимодействовать управляющие классы с классами интерфейса;
- Raytracer – класс визуализации сцены.

2.7 Проекция программного обеспечения

На рисунках 2.6 – 2.7 представлена проекция программного обеспечения.

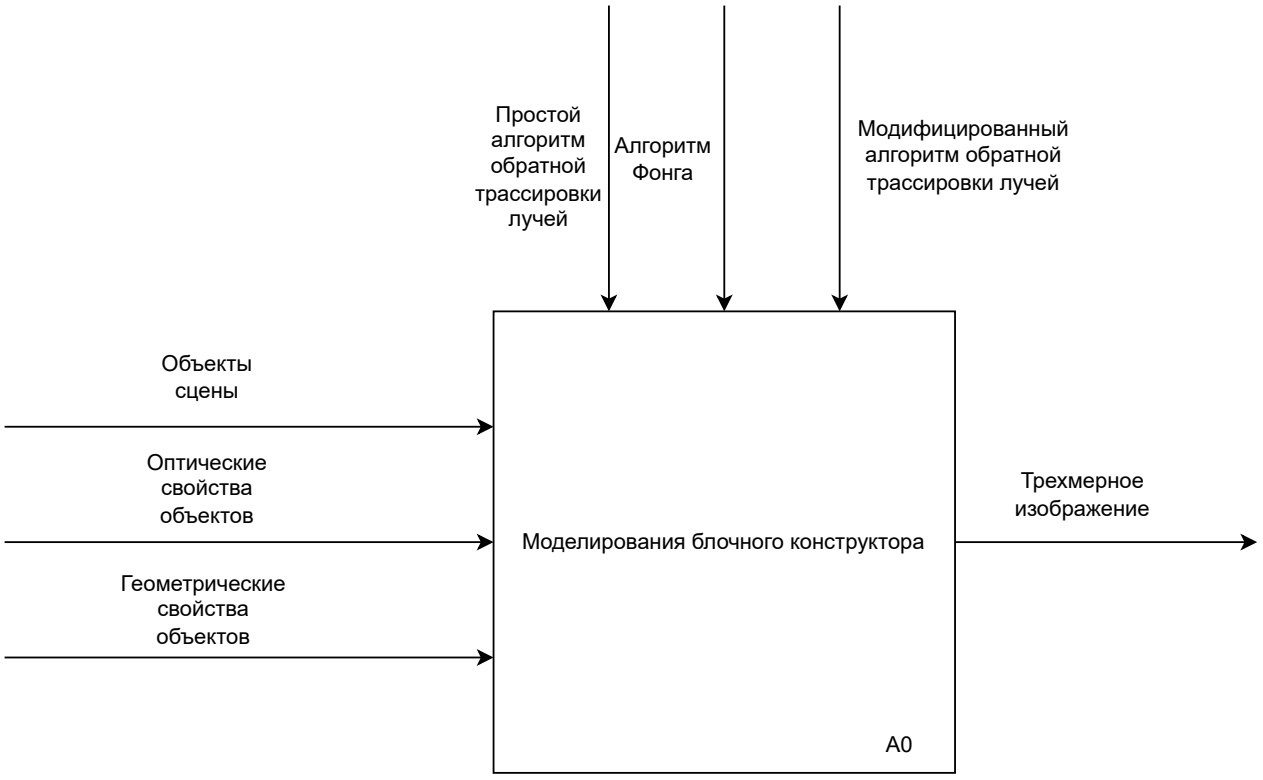

Рисунок 2.6 – Проекция программного обеспечения

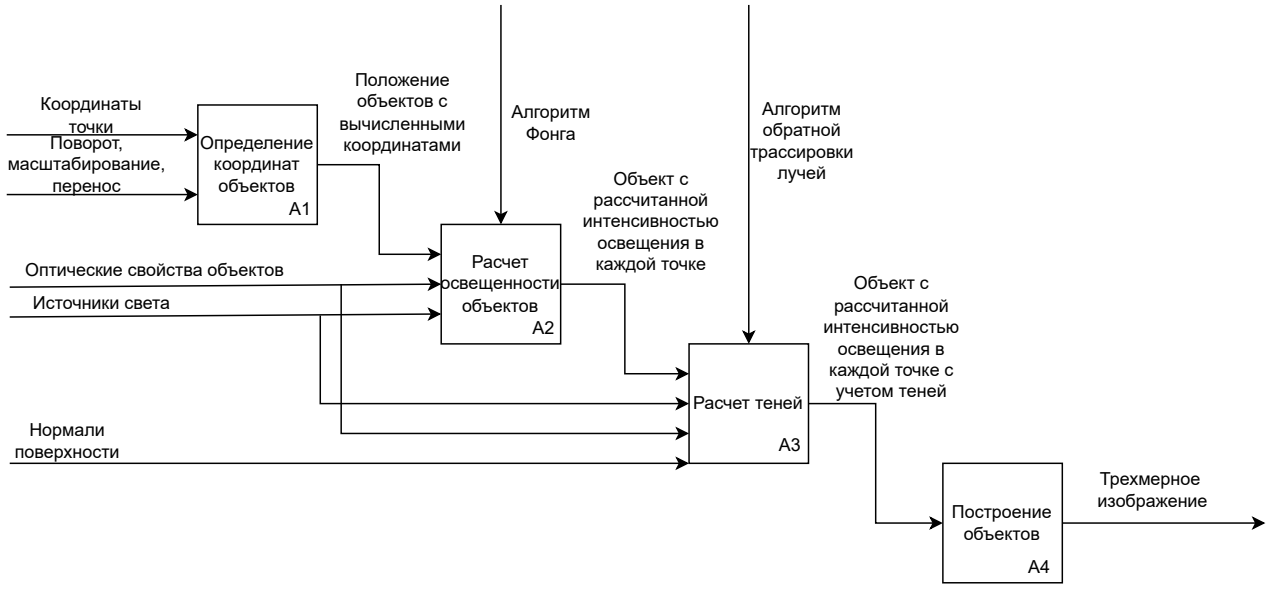

Рисунок 2.7 – Проекция программного обеспечения

Вывод

В данном разделе провели анализ приближения трехмерных объектов. Изучили трехмерные трансформации, указали требования к программному обеспечения. Также предоставили проекцию программного обеспечения, схему алгоритмов и диаграмму классов.

3 Технологическая часть

В этом разделе осуществляется выбор способов реализации программного обеспечения (ПО), также предоставляются реялизации выполенных алгоритмов.

3.1 Выбор и обоснование языка программирования и среды разработки

В качестве языка программирования был выбран `C#`, так как:

- я ознакомился с этим языком программирования во время обучения, что сократит время написания программы;
- этот язык программирования объектно-ориентирован, что позволяет:
  - использовать наследование, абстрактные классы и т.д.
  - представлять трехмерные объекты сцены в виде объектов классов, что позволяет легко организовать взаимодействие между ними, положительно влияя на читабильность, не снижая эффективности.

В качестве среды разработки был выбран Visual Studio, так как:

- он бесплатен в использовании студентам;
- я знаком с данной средой разработки, что сократить время изучения возможностей среды.

3.2 Реализация алгоритмов

В листингах 3.1 – 3.3 представлены реализации рассматриваемых алгоритмов.

```
1  public Vector3 TraceRay(Vector3 camera, Vector3 D, double t_min, double
    t_max, int x, int y)
2  {
3    double closest_t = Double.PositiveInfinity;
4    Primitive closest_object = null;
5
6    ClosestIntersection(ref closest_object, ref closest_t, camera, D, t_min,
    t_max);
7
8    if (closest_object == null)
9      return new Vector3();
10
11
12   Vector3 P = camera + closest_t * D;
13   Vector3 N;
14
15   if (closest_object is Parallelepiped)
16     N = Vec3dNormalParallelepiped(P, (Parallelepiped)closest_object);
17   else if (closest_object is Cube)
18     N = Vec3dNormalCube(P, (Cube)closest_object);
19   else if (closest_object is Triangle)
20     N = Vec3dNormalTriangle((Triangle)closest_object);
21   else
22     N = P - closest_object.position;
23   N = N / Vector3.Length(N);
24
25   double intensity = ComputeLighting(P, N, -D, closest_object.specular);
26
27   return intensity * closest_object.color;
28 }
```

Листинг 3.1 – Простой алгоритм обратной трассировки лучей

```
1  private Vector3 TraceRay(Vector3 camera, Vector3 dir, double t_min, double
     t_max, int depth)
2  {
3    double closest_t = Double.PositiveInfinity;
4    Primitive closest_object = null;
5
6    ClosestIntersection(ref closest_object, ref closest_t, camera, dir,
     t_min, t_max);
7
8    if (closest_object == null)
9      return new Vector3();
10
11   Vector3 P = camera + closest_t * dir;
12   Vector3 N;
13   if (closest_object is Parallelepiped)
14     N = Vec3dNormalParallelepiped(P, (Parallelepiped)closest_object);
15   else if (closest_object is Cube)
16     N = Vec3dNormalCube(P, (Cube)closest_object);
17   else if (closest_object is Triangle)
18     N = Vec3dNormalTriangle((Triangle)closest_object);
19   else
20     N = P - closest_object.position;
21   N = N / Vector3.Length(N);
22
23   double intensity = ComputeLighting(P, N, -dir, closest_object.specular);
24
25   Vector3 localColor = intensity * closest_object.color;
26
27   double r = closest_object.reflective;
28
29   if (depth <= 0 || r <= 0)
30     return localColor;
31
32   Vector3 R = ReflectRay(-dir, N);
33   Vector3 reflectedColor = TraceRay(P, R, 0.001, Double.PositiveInfinity,
     depth - 1);
34
35   Vector3 kLocalColor = (1 - r) * localColor;
36   Vector3 rReflectedColor = r * reflectedColor;
37
38   return kLocalColor + rReflectedColor;
39 }
```

Листинг 3.2 – Модифицированный алгоритм обратной трассировки лучей

```
1  private double ComputeLighting(Vector3 P, Vector3 N, Vector3 V, double
       specular)
2  {
3    double intensity = scene.ambient_light.intensity;
4    List<LightSource> sceneLight = scene.light_sources;
5
6    for (int i = 0; i < sceneLight.Count; i++)
7    {
8      Vector3 L;
9      double t_max;
10     L = sceneLight[i].position - P;
11     t_max = Double.PositiveInfinity;
12
13     double shadow_t = Double.PositiveInfinity;
14     Primitive shadow_object = null;
15     ClosestIntersection(ref shadow_object, ref shadow_t, P, L, 0.001, t_max)
       ;
16     if (shadow_object != null)
17       continue;
18
19     double n_dot_l = Vector3.ScalarMultiplication(N, L);
20
21     if (n_dot_l > 0)
22     {
23       intensity += sceneLight[i].intensity * n_dot_l / (Vector3.Length(N) *
       Vector3.Length(L));
24     }
25
26     if (specular != -1)
27     {
28
29       Vector3 R = 2 * N * n_dot_l - L;
30       double r_dot_v = Vector3.ScalarMultiplication(R, V);
31
32       if (r_dot_v > 0)
33       {
34         intensity += sceneLight[i].intensity * Math.Pow(r_dot_v / (Vector3.
       Length(R) * Vector3.Length(V)), specular);
35       }
36     }
37   }
38   return intensity;
39 }
```

Листинг 3.3 – Алгоритм Фонга

4 Исследовательская часть

4.1 Интерфейс приложения

На рисунке 4.1 приведено изображение интерфейса приложения.

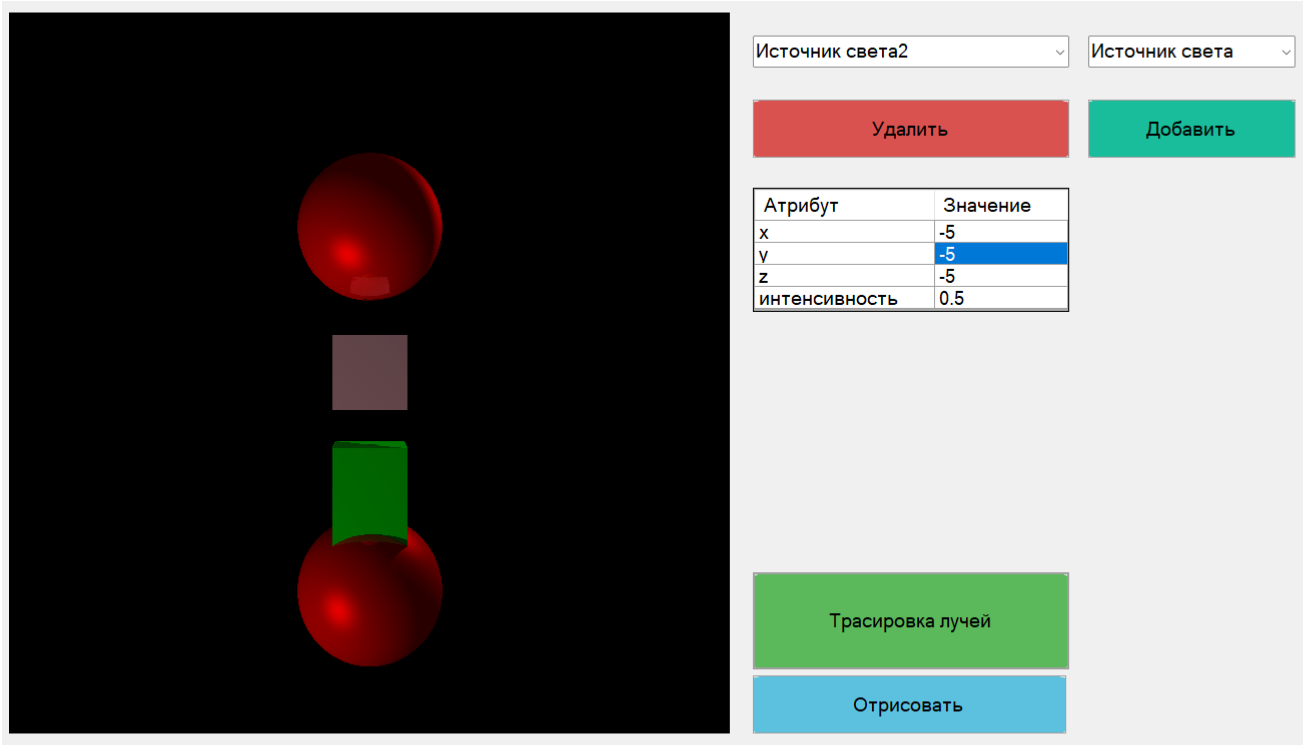

Рисунок 4.1 – Интерфейс

4.2 Сравнение реализаций трассирования лучей

В процессе выполнения курсового проекта были реализованы два метода трассировки лучей. Реализация этих методов различается по сложности. Для удобства их идентификации им были присвоены названия: простой и модифицированный. Простой метод отличается от модифицированного тем, что не учитывает тени, зеркальность объектов и их гладкость. Необходимо провести сравнение времени выполнения каждого метода в зависимости от количества объектов на сцене.

Для проведения сравнения между указанными алгоритмами были созданы специальные сцены, включающие в себя различное количество объектов.

На рисунке 4.2 приведен график, на котором отображается зависимость времени выполнения алгоритмов от количества объектов на сцене. Время построения сцены рассчитывалось как среднее значение из 20 измерений.

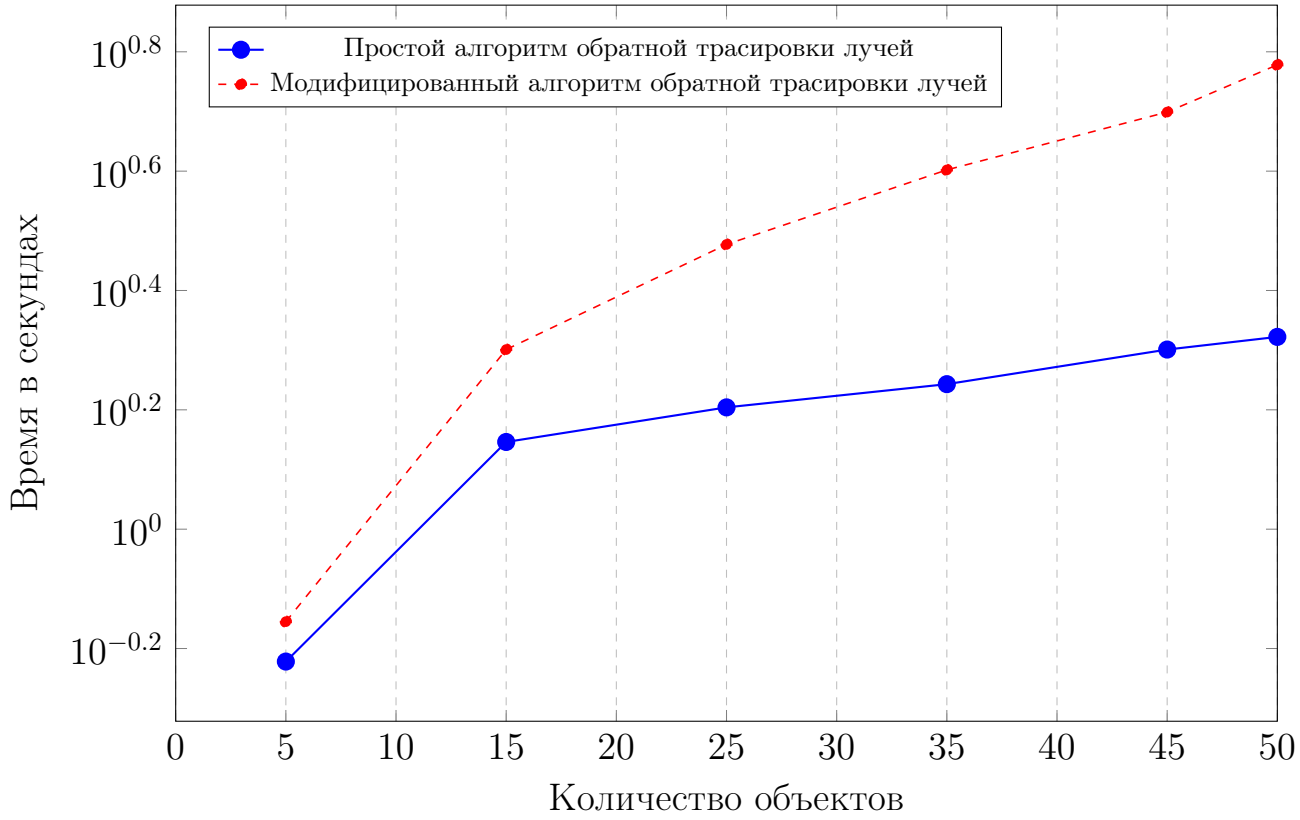

Рисунок 4.2 – Сравнение времени работы реализаций алгоритмов

Вывод

Явно заметно, что простая реализация требует меньше времени в сравнении со модифицированной. Однако следует учитывать, что простой алгоритм не полностью справляется с задачей создания реалистичного изображения. Простой алгоритм отлично подходит для работы со сценой, добавления объектов, изменения их свойств и прочего. Как только все объекты установлены и изменены, можно запустить модифицированный алгоритм для получения более реалистичного изображения.

ЗАКЛЮЧЕНИЕ

В результате выполнения данного курсового проекта проведен анализ методов представления объектов, алгоритмов удаления невидимых линий и поверхностей, а также моделей освещения. В работе указаны преимущества и недостатки этих методов.

На основе результатов анализа разработано программное обеспечение, которое позволяет создавать собственные сцены, используя сферы, параллелепипеды, кубы и пирамиды. В результате проектирования были созданы новые и адаптированы существующие структуры данных и алгоритмы для решения задачи.

Эксперименты по сравнению реализаций трассировки лучей выявили, что для ускорения процесса конструирования сцены не обязательно использовать реалистичные изображения, что позволяет упростить алгоритм трассировки лучей.

В дальнейшем развитии программы можно рассмотреть расширение ее функционала для взаимодействия не только с примитивами, но и с более сложными объектами, созданными другими программами. Это позволит конструировать более сложные сцены и получать их более реалистичные изображения.

В результате выполнения курсовой работы были выполнены следующие задачи:

1) проанализированы имеющиеся алгоритмы и определены оптимальные методы для решения основной задачи;

2) разработана эффективная структура программы;

3) выбран наиболее подходящий язык программирования и интегрированную среду разработки для выполнения задачи;

4) создан программны продукт для решения задачи и реализованы выбранные алгоритмы;

5) создан понятный пользовательский интерфейс;

6) проведены исследования, основанные на полученных результатах.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Роджерс Д. Алгоритмические основы машинной графики. —М.: Мир, 1989. Т. 512.

[2] Алгоритм, использующий Z-буфер [Электронный ресурс]. — Режим доступа: https://eccarrilloe.github.io/2017/09/24/Rasterization-Z-buffer-Algorithm/ (дата обращения: 26.10.2023).

[3] А.В. Куров. Конспект лекций по дисциплине «Компьютерная графика». 2023.

[4] Простые модели освещения. [Электронный ресурс]. — Режим доступа: http://grafika.me/node/344 (дата обращения: 19.10.2023).

[5] Creating an icosphere mesh in code. [Электронный ресурс]. — Режим доступа: http://blog.andreaskahler.com/2009/06/creating-icosphere-mesh-in-code.html (дата обращения: 26.10.2023).

[6] Шар. [Электронный ресурс]. — Режим доступа: https://en.wikipedia.org/wiki/Geodesic_polyhedron#/media/File:Geodesic_dodecahedral_polyhedron_example.png (дата обращения: 25.10.2023).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе

Презентация содержит 9 слайдов.