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 КУРСОВОЙ

РАБОТЕ НА ТЕМУ:

разработка ПО, которое предоставляет возможность:
а) построения реалистического изображения из трехмерных геометрических объектов
б) выбора и добавления в сцену трехмерных объектов
в) перемещения, поворота и масштабирования объектов
г) изменение текстуры объекта, его цвет, свойств поверхности.

Студент ИУ7-55Б _____ Фролов Е.
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ Романова Т.Н.
(Подпись, дата) (И.О.Фамилия)

2021 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 ___ИУ-7__
(Индекс)
_____ _И.В. Рудаков_
(И.О.Фамилия)
« _____ » _____ 2021 г.

ЗАДАНИЕ
на выполнение курсового проекта

по дисциплине _____Компьютерная графика_____

Студент группы _____ИУ7-55Б_____

_____Фролов Евгений Алексеевич_____
(Фамилия, имя, отчество)

Тема курсового проекта Построение трехмерной сцены изображения с учетом отражения от трехмерных геометрических объектов.

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

График выполнения проекта: 25% к __3_ нед., 50% к _6__ нед., 75% к _9_ нед., 100% к _14__ нед.

Задание Разработать программу для построения реалистического изображения из трехмерных геометрических объектов. Предусмотреть возможность выбора и добавления в сцену трехмерных объектов. Характеристики отражения и пропускания света задаются независимо для каждого из тел. Количество объектов в сцене с учетом источников света не более пяти. Реализовать возможность перемещения, поворота и масштабирования объектов. Добавить возможность изменения текстуры объекта, его цвет, свойств поверхности. Обзор сцены - камерой. Для рендеринга изображения использовать последний вид с камеры.
Оформление курсового проекта:

Расчетно-пояснительная записка на 25-35 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
Расчётно-пояснительная записка должна содержать постановку задачи, введение, аналитический раздел, конструкторский раздел, технологический раздел, экспериментально-исследовательский раздел, заключение, список литературы, приложения.
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 « ___ » _____ 2021 г.

Руководитель курсового проекта _____ _____Романова Т.Н.___
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____Фролов Е.____
(Подпись, дата) (И.О.Фамилия)

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

Содержание

Введение

Компьютерная графика - область деятельности, в которой компьютеры наряду со специальным программным обеспечением используются в качестве инструмента как для создания и редактирования изображений, так и для оцифровки визуальной информации, полученной из реального мира, с целью дальнейшей ее обработки и хранения.

На данный момент большую популярность в компьютерной графике имеют алгоритмы получения реалистических изображений. В тоже время, они являются довольно требовательными к ресурсам системы из-за того, что алгоритмам приходится учитывать множество физических явлений, например преломление, отражения и тд. От того, какую реалистичность мы требуем к изображению, зависит сложность вычислений. Стремление к созданию наиболее реалистичного изображения становится трудно выполнимым при моделировании динамических сцен, в которых требуется поддержание высокой частоты кадров.

Цель работы — провести исследование алгоритмов, использующих для построения реалистичных изображений и алгоритмов для работы с камерой наблюдателя, создавать освещение и проецировать полученный результат на экран.

Для достижения поставленной цели, необходимо:

- провести анализ выбранных алгоритмов для удаления невидимых линий, закраски, текстурирования, освещения моделей и выделить наиболее подходящие для будущей программы;
- реализовать выбранные алгоритмы и структуры данных;
- разработать программное обеспечение для отображения трехмерной сцены;
- реализовать интерфейс программного модуля;
- выполнить исследование на основе разработанной программы;

Результат работы — программное обеспечение для создания графических сцен из трехмерных моделей и их визуализация с учетом выбранных текстур, а также оптический эффект отражения света, преломления, прозрачности.

1 Аналитический раздел

Основную задачу отрисовки сцены можно разбить на составные задачи для получения результата, а именно:

- выбор способа представления модели;
- перевод координат модели из пространства объекта в пространство экрана и удаление невидимых линий;
- растеризация;
- добавление освещения;
- перемещение камеры;
- проведение преобразований над объектом;

Для генерации реалистического изображения будем использовать алгоритм обратной трассировки лучей, для получения красивой картинки.

1.1 Описание модели трехмерного объекта в сцене

В компьютерной графике существует множество способов задания трехмерных объектов. В данной работе нужно выбрать тот, который позволит воспринимать объекты объемными и накладывать текстуру на них. Учитывая это, модель можно представить в виде:

- геометрического примитива;
- полигональной сетки;

1.1.1 Модель как геометрический примитив

Примитив, как правило, может быть описан процедурой, которая принимает некоторые значения параметров, например, для построения сферы достаточно знать ее радиус и положение центра. В качестве таких примитивов могут выступать тела с простой формой: куб, цилиндр, пирамида, сфера, конус и призма.

Плюсы таких моделей:

- простота построения, следовательно увеличение скорости работы программы;
- минимальное количество ключевой информации для их отрисовки;

При этом, из минусов:

- модели могут описывать только простейшие объекты;
- наложение текстуры достаточно затруднительно;

1.1.2 Модель как полигональная сетка

Полигональная сетка это совокупность вершин, рёбер и граней, которые определяют форму многогранного объекта в трехмерной компьютерной графике и объемном моделировании. Гранями обычно являются треугольники, четырёхугольники или другие простые выпуклые многоугольники (полигоны), так как это упрощает рендеринг, но сетки могут также состоять и из наиболее общих вогнутых многоугольников, или многоугольников с отверстиями. С помощью этого способа, можно задать объект любой формы. В качестве полигона может быть выбран любой многоугольник.

От количества полигонов в модели зависит то, насколько детально-проработанной она будет выглядеть, соответственно чем их больше, тем лучше выглядит модель, однако с ростом их числа растет время, которое программа затрачивает на отрисовку всех моделей в сцене. Благодаря тому, что полигоны задают плоскости, можно изменять нормали для достижения необходимых эффектов имитации неровностей поверхности, а также при создании модели указать текстурные координаты, то есть то, как изображение будет накладываться на модель.

В отличии от примитивов, создание такой модели требует проведения предварительной работы, так как необходимо явно указать как будет выглядеть полигональная сетка, задать текстурные координаты.

1.1.3 Выводы

После анализа обоих вариантов, в качестве способа задания модели была выбрана полигональная сетка, так как процесс наложения текстуры для таких объектов наиболее прост, а также это дало возможность добавлять те объекты в сцену, которые нельзя или трудно задать параметрически.

1.2 Алгоритмы построения трехмерного изображения

1.2.1 Z-буфер

Это алгоритм, работающий в пространстве изображения, как показано на рисунке 1.2. Буфер кадра используется для запоминания интенсивности каждого пиксела в пространстве изображения, z-буфер - это отдельный буфер глубины, используемый для запоминания координаты z или глубины каждого видимого пиксела в пространстве изображения.

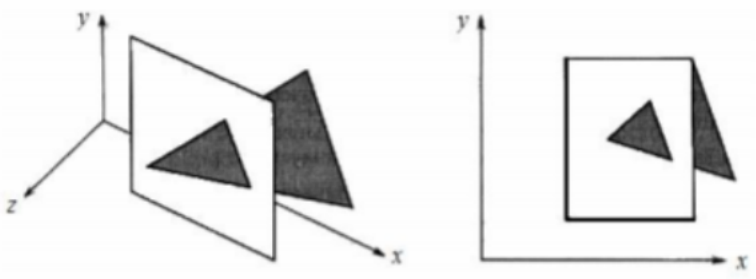


Рисунок 1 – Пример алгоритма с использованием Z-буфера.

В процессе работы значение z каждого пиксела, который нужно занести в буфер кадра, сравнивается с глубиной уже занесенного в z-буфер пиксела. Если новый пиксел расположен впереди пиксела, находящегося в буфере кадра, то новый пиксел заносится в этот буфер и производится корректировка z-буфера новым значением z.

К достоинствам алгоритма, использующего z-буффер можно отнести простоту реализации, а также отсутствие предварительной сортировки элементов сцены.

Недостатками алгоритма являются трудоёмкость реализации эффектов прозрачности, а также перерасход по памяти - алгоритм предполагает хранение двух двумерных массивов, размер которых увеличивается с увеличением размеров изображения.

1.2.2 Алгоритм обратной трассировки лучей

Алгоритм обратной трассировки лучей выглядит следующим образом: из камеры через каждый пиксел изображения испускается луч и находится точка его пересечения с поверхностью сцены. Лучи, выпущенные из камеры, называют первичными. Пусть, первичный луч пересекает некий объект 1 в точке H1, как показано на рисунке 2.

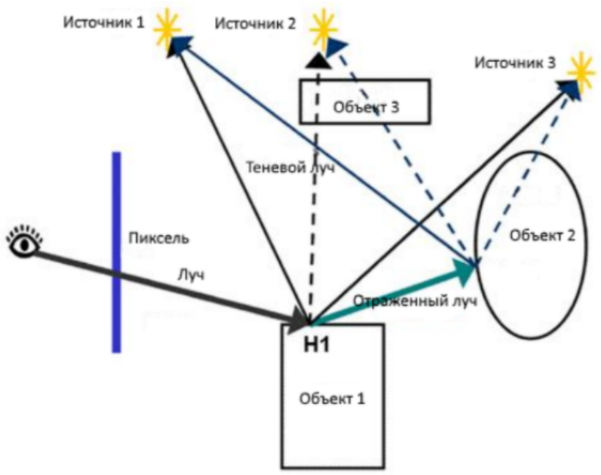


Рисунок 2 – Алгоритм обратной трассировки лучей.

Затем, мы определяем для каждого источника освещения, видна ли из него эта точка. Тогда в направлении каждого точечного источника света испускается теневой луч из точки H1. Это позволяет определить, освещается ли данная точка конкретным источником. Если теневой луч находит пересечение с другими объектами, расположенными ближе к точке H1, чем источник света, значит, точка H1 находится в тени от этого источника и освещать ее не надо, иначе освещение рассчитывается по некоторой локальной модели. Освещение со всех видимых (из точки H1) источников света складывается. Далее, если материал объекта 1 имеет отражающие свойства, из точки H1 испускается отраженный луч и для него вся процедура трассировки рекурсивно повторяется. Аналогичные действия должны быть выполнены, если материал имеет преломляющие свойства.

Техника рендеринга с трассировкой лучей отличается высоким реализмом получаемого изображения. Она позволяет воссоздавать тени, отражения и преломления света. Алгоритм трассировки позволяет выполнять вычисления параллельно. Также обеспечивает отсечение невидимых поверхностей, перспектива.

Недостатком метода обратного трассирования является производительность. Трассировка лучей каждый раз начинает процесс определения

цвета пикселя заново, рассматривая каждый трассируемый луч в отдельности.

1.2.3 Выводы

Для создания реалистичного изображения лучше всего подходит алгоритм обратной трассировки лучей, но из-за низкой производительности алгоритм фактически неприменим к созданию сцены в реальном времени. Поэтому в программе будет предусмотрено два режима работы. В первом режиме пользователь сможет добавлять новые объекты, редактировать их размер, положение, выбирать цвет и текстуру, на данном этапе важна скорость, поэтому в качестве алгоритма отсечения невидимых линий и поверхностей был выбран алгоритм z-буфера. Во втором режиме программа должна будет строить реалистичное изображение, учитывать тени, прозрачность, отражение объектов, поэтому в этом режиме будет использоваться алгоритм обратной трассировки лучей, как позволяющий достичь наибольшей реалистичности построенного изображения.

1.3 Анализ алгоритмов наложения текстур на объекты трехмерной сцены

1.3.1 Аффинное текстурирование

Этот метод текстурирования основан на приближении u, v линейными функциями, где u, v - координаты текстуры. Пусть u - линейная функция, $u = k_1 \cdot sx + k_2 \cdot sy + k_3$, где sx, sy - координаты принадлежащие проекции текстурируемого треугольника. Можно посчитать $k1$, $k2$, $k3$ исходя из того, что в вершинах грани u, v известны - это дает три уравнения, из которых находятся эти коэффициенты. Однако в таком случае вычисление цвета пикселя получается медленным. Оптимизацией данного

подхода является расчет начальных значений координат текстур для каждого полигона, а затем билинейная интерполяция значений и текстуры. Основным недостатком этого метода является игнорирование координаты z, вследствие чего данные искажаются, и текстура накладывается нереалистично.

1.3.2 Перспективно-корректное текстурирование

Этот метод основан на приближении u, v кусочно-линейными функциями. При отрисовке каждая сканирующая строка разбивается на части, в начале и конце каждого куска считаются точные значения u, v, а в каждой части они интерполируется линейно. Точные значения u и v можно считать по формулам точного текстурирования, но обычно используют более простой путь. Он основан на том факте, что значения $1/Z$, u/Z и v/Z зависят от sx, sy линейно. Таким образом, достаточно для каждой вершины посчитать $1/Z$, u/Z, v/Z и линейно их интерполировать - точно так же, как интерполируются u и v в аффинном текстурировании. Причем, поскольку эти значения зависят от sx, sy строго линейно, то интерполяция дает не приближенные результаты, а точные.

Сами же точные значения u, v считаются, как:

$$u = (u/z)/(1/z), v = (v/z)/(1/z) \tag{1}$$

1.3.3 Выводы

Наиболее приемлемой является перспективно-корректная реализация, поскольку в ее основе лежит точное текстурирование, что делает ее результаты более приближенными к действительным, нежели у аффинной, к тому же она учитывает перспективу изображения.

1.4 Алгоритмы закраски треугольников

1.4.1 Алгоритм заполнения треугольника с использованием барицентрических координат

Барицентрические координаты - это координаты, в которых точка треугольника описывается как линейная комбинация вершин (формально, подразумевая, что точка является центром масс треугольника, при соответствующем весе вершин). Чаще всего используют нормализованный вариант - т.е. суммарный вес трех вершин равен единице:

$$\begin{aligned} p = b_0 \cdot v_0 + b_1 \cdot v_1 + b_2 \cdot v_2 \\ b_0 + b_1 + b_2 = 1 \end{aligned} \tag{2}$$

У этой системы координат так же есть очень полезное свойство, которое позволяет их вычислять: барицентрические координаты равны отношению площадей треугольников, к общей площади треугольника (рисунок 3).

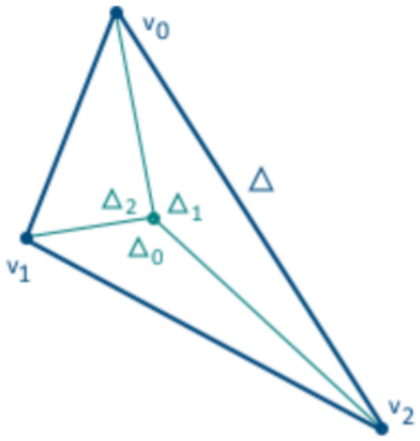


Рисунок 3 – Представление барицентрических координат.

$$
\begin{aligned}
b_0 &= \frac{\delta_0}{\delta} \\
b_1 &= \frac{\delta_1}{\delta} \\
b_2 &= 1 - b_1 - b_0
\end{aligned}
\tag{3}
$$

Третья координата может вычисляться не через площади треугольников, поскольку сумма трех координат равна единице - фактически, мы имеем только две степени свободы. Главной особенностью данных координат является то, что с их помощью можно интерполировать значение любого атрибута в произвольной точке треугольника: значение атрибута в заданной точке треугольника равно линейной комбинации барицентрических координат и значений атрибута в соответствующих вершинах:

$$
T = b_0 \cdot T_0 + b_1 \cdot T_1 + b_2 \cdot T_2 \tag{4}
$$

Для коррекции перспективы используют другую формулу - сначала вычисляют интерполированное значение - затем интерполированное значение и затем делят их друг на друга, чтобы получить итоговое значение T с учетом перспективы:

$$
\begin{aligned}
\frac{T}{z} &= \frac{T_0}{z} \cdot b_0 + \frac{T_1}{z} \cdot b_1 + \frac{T_2}{z} \cdot b_2 \\
\frac{1}{z} &= \frac{1}{z_0} \cdot b_0 + \frac{1}{z_1} \cdot b_1 + \frac{1}{z_2} \cdot b_2 \\
T &= \frac{\frac{T}{z}}{\frac{1}{z}}
\end{aligned}
\tag{5}
$$

Сам же алгоритм закраски с использованием данных координат достаточно прост:

1. Для начала надо найти минимальный по площади прямоугольник, который бы содержал в себе данный треугольник.

2. Далее для каждой строки прямоугольника проводим его сканирование слева направо.

3. Для каждого выбранного пикселя находим его барицентрические координаты.

4. Если значение каждой из них находится в отрезке от 0 до 1 и их сумма не превышает 1, то закрашиваем пиксель.

1.4.2 Выводы

Алгоритм с использованием барицентрических координат оказался самым эффективным и легко реализуемым из выше перечисленных, а также предоставил возможность корректно интерполировать атрибуты треугольника, что было бы затруднительно или невозможно при использовании других алгоритмов.

1.5 Описание трехмерных преобразований

1.5.1 Способы хранения и обработки декартовых координат

Координаты можно хранить в форме вектор-столбца x, y, z. Однако в этом случае неудобно применять преобразования поворота, так как такой вектор нельзя умножить на соответствующие квадратные матрицы трансформации размерности четыре на четыре. Целесообразнее использовать вектор-столбцы размерности четыре - [x, y, z, w], где координата w=1.

Преобразования координат выполняются умножением слева преобразуемого вектора-строки на соответствующую матрицу линейного оператора.

1.5.2 Преобразование трехмерного пространства в двумерное пространство экрана

Пространство экрана компьютера обладает только двумя из трех измерений, а именно шириной и высотой. Трехмерное пространство описывается тремя единичными ортогональными векторами.

В тоже время экран компьютера имеет лишь две координаты и состоит из пикселей, изменяя цвет которых получается итоговое изображение. Каждый пиксель может быть однозначно задан двумя координатами x и y, в силу чего возникает проблема изображения трехмерного объекта на экране с сохранением эффекта объемности.

Обычно многие графические программы делают это путем применения четырех преобразований:

1. Перевод объекта из собственного пространства в мировое.
2. Перевод объекта из мирового пространства в пространство камеры.
3. Проекция всех точек из пространства камеры во все видимые точки, где координаты x, y, находятся в диапазоне [-w;w], в диапазоне [0;w].
4. Масштабирование точек, полученных после предыдущего преобразования на картинку необходимого разрешения.

Для перевода объекта из собственного пространства в пространство экрана компьютера используют матрицы преобразований. Процесс обычно строится следующим образом:

1. Вычисляются все необходимые матрицы.
2. Вычисленные матрицы перемножаются.
3. Вектор-строка, описывающий положение точки в пространстве, умножается на результирующую матрицу.

1.5.3 Матрицы аффинных преобразований декартовых координат

1. Сдвиг точки на dx, dy, dz по координатным осям.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{pmatrix}$$

2. Масштабирование относительно начала координат с коэффициентами sx, sy, sz.

$$\begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3. Поворот относительно осей x, y, z на угол α.

$$Rz(\alpha) = \begin{pmatrix} cos(\alpha) & sin(\alpha) & 0 & 0 \\ -sin(\alpha) & cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$Rx(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha) & sin(\alpha) & 0 \\ 0 & -sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$Ry(\alpha) = \begin{pmatrix} cos(\alpha) & 0 & -sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ sin(\alpha) & 0 & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

1.5.4 Преобразования трехмерной сцены в пространство камеры

Для того, чтобы преобразовать сцену в пространство камеры нужно умножить каждую вершину всех полигональных моделей на матрицу камеры. Сама камера задается набором следующих атрибутов: положение центра камеры в мировом пространстве, вектора направления взгляда, направления верха камеры.

$$\begin{pmatrix} Right, & Up, & Look, & 0 \\ Right, & Up, & Look, & 0 \\ Right, & Up, & Look, & 0 \\ -(Position \cdot Right) & -(Position \cdot Up) & -(Position \cdot Look) & 1 \end{pmatrix}$$

- Look - координаты точки в пространстве, на которую смотрит камера;
- Up - вектор, который указывает куда смотрит верх камеры;
- Right - ортогональный вектор к векторам направления взгляда и вектору направления верха камеры.

1.5.5 Матрица перспективной проекции

После перехода в пространство камеры следует умножить каждую вершину всех полигональных моделей на матрицу проекции. Матрица проекции отображает заданный диапазон усеченной пирамиды в пространство отсечения, и при этом манипулирует w-компонентой каждой вершины таким образом, что чем дальше от наблюдателя находится вершина, тем больше становится это w-значение. После преобразования координат в пространство отсечения x, y попадают в диапазон от -w до w, а вершина z от 0 до w (вершины, находящиеся вне этого диапазона, отсекаются).

$$\begin{pmatrix} \frac{cot(\frac{FOV}{2})}{aspect\_ratio} & 0 & 0 & 0 \\ 0 & cot(\frac{FOV}{2}) & 0 & 0 \\ 0 & 0 & \mathrm{Z}_f \frac{}{Z_f - Z_n} & 1 \\ 0 & 0 & -\mathrm{Z}_f \cdot Z_n \frac{}{Z_f - Z_n} & 0 \end{pmatrix}$$

- aspect ratio - отношение ширины изображения к его высоте;
- FOV - угол обзора камеры;
- Zn - координата z ближней к камере плоскости отсечения пирамиды видимости;
- Zf - координата z дальней от камеры плоскости отсечения пирамиды видимости.

После перевода в пространство камеры, все координаты необходимо спроецировать на одну плоскость путем деления на координату z. Стоит отметить, что после применения умножения вектора координат на матрицу перспективной проекции, истинная координата z автоматически заносится в координату w, поэтому вместо деления на z делят на w.

1.5.6 Преобразования трехмерной сцены в пространство области изображения

Для того, чтобы преобразовать спроецированные координаты в координаты области изображения, достаточно умножить вектор координат на следующую матрицу:

$$\begin{pmatrix} hW & 0 & 0 & hW \\ 0 & hH & 0 & hH \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- W - ширина изображения,
- H - высота изображения,
- hW - половина ширины изображения,
- hH - половина высоты изображения

1.5.7 Выводы

Для получения итогового изображения на экране, необходимо применить ряд преобразований к объектам в сцене с использованием матриц.

1.6 Анализ алгоритма закраски

1.6.1 Метод закраски Фонга

Метод закраски Фонга основан на интерполяции вектора нормали, который затем используется в модели освещения для вычисления интенсивности пиксела. Процесс закраски по методу Фонга осуществляется в четыре этапа:

1. Определяются нормали к граням.
2. По нормалям к граням определяются нормали в вершинах.
3. В каждой точке закрашиваемой грани определяется интерполированный вектор нормали.
4. По направлению векторов нормали определяется цвет точек грани.

Закраска Фонга требует больших вычислительных затрат, однако при этом достигается лучшая локальная аппроксимация кривизны поверхности, получается более реалистичное изображение, правдоподобнее выглядят зеркальные блики.

1.6.2 Метод закраски Гуро

Метод закраски Гуро основан на интерполяции интенсивности. Он позволяет устранить дискретность изменения интенсивности и создать иллюзию гладкой криволинейной поверхности. Процесс закраски по методу Гуро осуществляется в четыре этапа:

1. Вычисляются нормали ко всем полигонам.

2. Определяются нормали в вершинах путем усреднения нормалей по всем полигон нальным граням, которым принадлежит рассматриваемая вершина, как показано на рисунке 4.

3. Используя нормали в вершинах, и применяя определенную модель освещения, вычисляют значения интенсивностей в вершинах многоугольника.

4. Каждый многоугольник закрашивается путем линейной интерполяции значений интенсивностей в вершинах сначала вдоль каждого ребра, а затем и между ребрами вдоль каждой сканирующей строки.

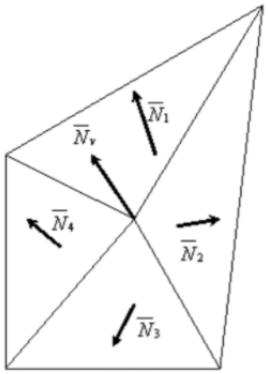


Рисунок 4 – Определение нормалей

Метод Гуро применим только для небольших граней, расположенных на значительном расстоянии от источника света. Если же размер грани достаточно велик, то расстояние от источника света до ее центра будет значительно меньше, чем до ее вершин, и, согласно закону освещенности, центр грани должен быть освещен сильнее ребер. Однако модель изменения освещенности, принятая в методе Гуро, предполагает линейное изменение яркости в пределах грани и не позволяет сделать середину грани ярче, чем края. В итоге на изображении появляются участки с неестественной освещенностью.

1.6.3 Выводы

Исходя из поставленной задачи, вместе с методом z-буфера предпочтительнее использовать алгоритм Гуро, сочетая приемлемую скорость работы и качество получаемого изображения.

1.7 Алгоритм, моделирующие освещение

1.7.1 Модель Ламберта

Модель Ламберта (рисунок 5) моделирует идеальное диффузное освещение. Считается, что свет при попадании на поверхность рассеивается равномерно во все стороны. При расчете такого освещения учитывается только ориентация поверхности (нормаль N) и направление на источник света (вектор L).

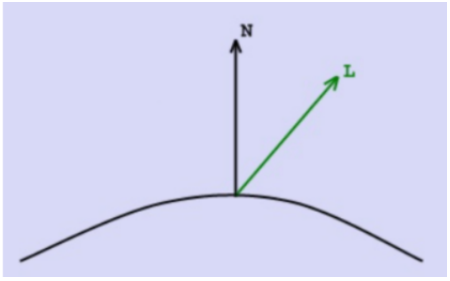


Рисунок 5 – Модель освещения Ламберта

Для удобства все векторы, описанные ниже, берутся единичными. В этом случае косинус угла между ними совпадает со скалярным произведением.

$$I_d = k_d \cdot cos(\bar{L}, \bar{N}) \cdot i_d = k_d \cdot (\bar{L}, \bar{N}) \cdot i_d \tag{6}$$

- I_d - рассеянная составляющая освещенности в точке;
- k_d- свойство материала воспринимать рассеянное освещение;
- i_d- интенсивность рассеянного освещения;
- L - направление на источник света;
- N - вектор нормали в точке.

Модель Ламберта является одной из самых простых моделей освещения. Данная модель очень часто используется в комбинации других моделей, так как практически в любой модели освещения можно выделить диффузную составляющую. Равномерная часть освещения, без присутствия какого-либо всплеска, будет представляться моделью Ламберта с определенными характеристиками. Данная модель может быть очень удобна для анализа свойств других моделей.

1.7.2 Вывод

В результате анализа алгоритмов, в соответствии с поставленной задачей для расчета интенсивности в точке была выбрана модель Ламберта при отрисовке сцены с использованием z-буфера и модель Фонга при использовании обратной трассировки лучей, так как она позволяет учитывать матовые и блестящие поверхности, тем самым делая изображение более реалистичным.

2 Конструкторский раздел

2.1 Алгоритм удаления невидимых граней с использованием z-буфера

После выбора алгоритмов, которые будут использоваться в курсовом проекте, составляется формальное описание выбранных алгоритмов в формате схем алгоритмов, а также приводятся математические выкладки.

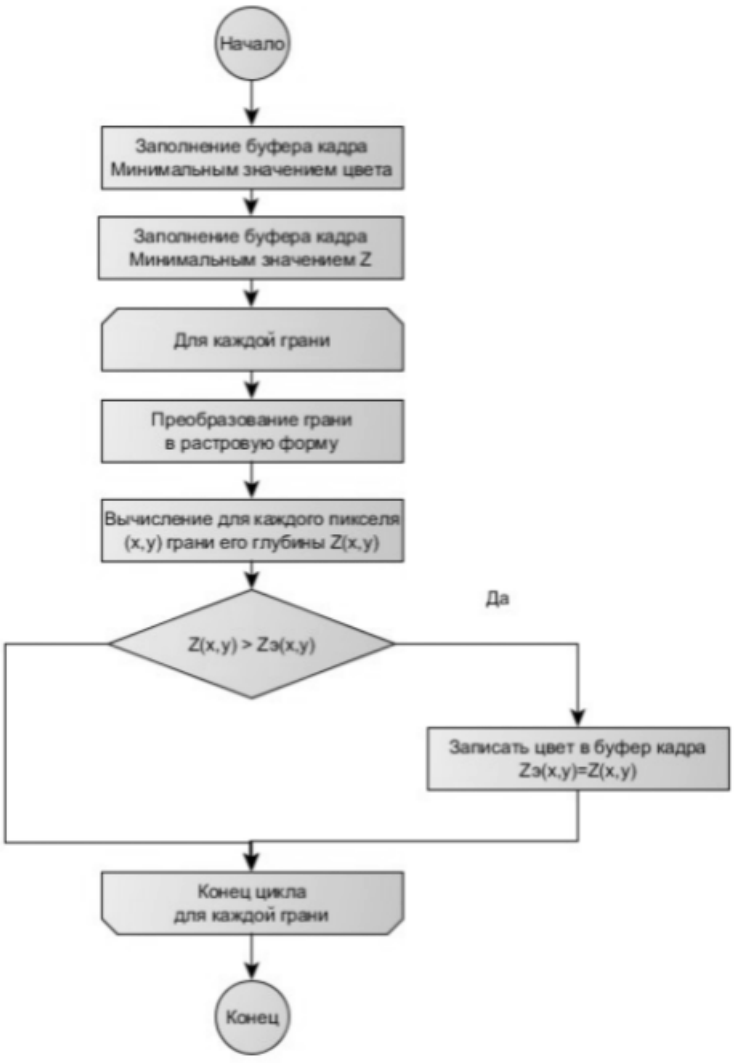


Рисунок 6 – Алгоритм z-буфера

2.2 Алгоритм обратной трассировки лучей

Рисунок 7 – Алгоритм обратной трассировки лучей

2.3 Пересечение трассирующего луча с треугольником

Для определения пересечения прямой (луча) и треугольника в трехмерном пространстве удобно использовать алгоритм Моллера - Трумбора,

для работы которого не требуется предварительное вычисление уравнения плоскости, содержащей треугольник. Пусть точка на треугольнике T(u, v) записана как:

$$T(u, v) = (1 - u - v) \cdot V_0 + u \cdot V_1 + v \cdot V_2 \tag{7}$$

(u, v) - барицентрические координаты, которые должны удовлетворять условиям: $u \geq 0$, $v_0 \geq 0$ и $u + v \leq 1$

Вычисление точки пересечения луча R(t) и треугольника T(u, v) эквивалентно решению уравнения R(t) = T(u, v), которое записывается как:

$$O + t \cdot D = (1 - u - v) \cdot V_0 + u \cdot V_1 + v \cdot V_2 \tag{8}$$

- O - точка начала луча;
- D - вектор направления луча;
- t - любое вещественное число;

Перегруппировав члены, получим:

$$\Big[-D, V_1 - V_0, V_2 - V_0\Big] \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - V_0 \tag{9}$$

Это значит, что барицентрические координаты (u, v) и расстояние t от начала луча до точки пересечения могут быть найдены из решения системы линейных уравнений выше.

Обозначим $E_1 = V_1 - V_0$, $E2 = V_2 - V_0$ и $T = O - V_0$. Проведя алгебраические преобразования можно получить формулу в следующем виде:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{dot(P, E_1)} \begin{bmatrix} dot(Q, E_2) \\ dot(P, T) \\ dot(Q, D) \end{bmatrix} \tag{10}$$

$$P = (D \times E_2)$$

$$Q = (T \times E_1)$$

2.4 Алгоритм нахождения пересечения луча с параллелепипедом

При работе алгоритма обратной трассировки лучей крайне неэффективно при каждой трассировке луча искать пересечения со всеми полигонами каждого объекта в сцене, поэтому имеет смысл заключить каждый объект в параллелепипед, который бы полностью его включал. Данный параллелепипед задается координатами двух вершин: с минимальными и максимальными значениями координат x, y, z. Таким образом это позволяет задать шесть

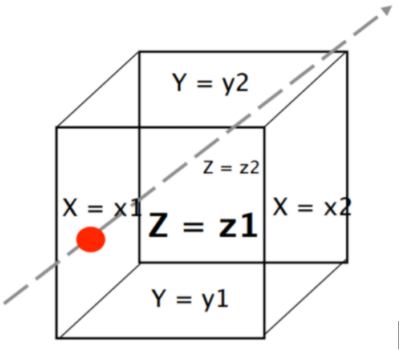


Рисунок 8 – Пересечение луча с параллелепипедом

плоскостей, ограничивающих параллелепипед, и при этом все они будут параллельны координатным плоскостям (рисунок 8).

Рассмотрим пару плоскостей, параллельных плоскости $y_z : X = x1$ и $X = x2$. Пусть D вектор направления луча. Если координата x вектора 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 Dx не равно 0, то вычисляются отношения:

$$\begin{aligned} t1x &= \frac{(x_1 - x_0)}{Dx} \\ t2x &= \frac{(x_2 - x_0)}{Dx} \end{aligned} \tag{11}$$

Можно считать, что найденные величины связаны неравенством tlx < t2x. Пусть tn = t1x, tf = t2x. Считая, что не равно 0, и рассматривая вторую пару плоскостей, несущих грани заданного параллелепипеда, Y = y1 и Y = y2, вычисляются величины:

$$\begin{aligned} t1y &= \frac{(y_1 - y_0)}{Dy} \\ t2y &= \frac{(y_2 - y_0)}{Dy} \end{aligned} \tag{12}$$

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 Dz не равно 0, и рассматривая вторую пару плоскостей, несущих грани заданного параллелепипеда, Z = z1 и Z = z2, вычисляются величины:

$$\begin{aligned} t1z &= \frac{(z_1 - z_0)}{Dz} \\ t2z &= \frac{(z_2 - z_0)}{Dz} \end{aligned} \tag{13}$$

и повторяются сравнения описанные выше. Если в итоге всех проведенных операций получается, что 0 < tn < tf или 0 < tf, то заданный луч пересечет исходный параллелепипед со сторонами, параллельными координатным осям. Следует отметить, что при пересечении лучом параллелепипеда

извне знаки tn и tf должны быть равны, в противном случае можно сделать вывод, что луч пересекает параллелепипед изнутри.

2.5 Нахождение отраженного луча

Для нахождения направления отраженного луча достаточно знать направление падающего луча и нормаль к поверхности в точке падения луча.

Можно разложить $\bar{L}$ на два вектора $\bar{L}_p$ и $\bar{L}_n$, таких что $\bar{L} = \bar{L}_p + \bar{L}_n$,где n параллелен $\bar{N}$, а $\bar{L}_p$ перпендикулярен, как изображено на рисунке 9.

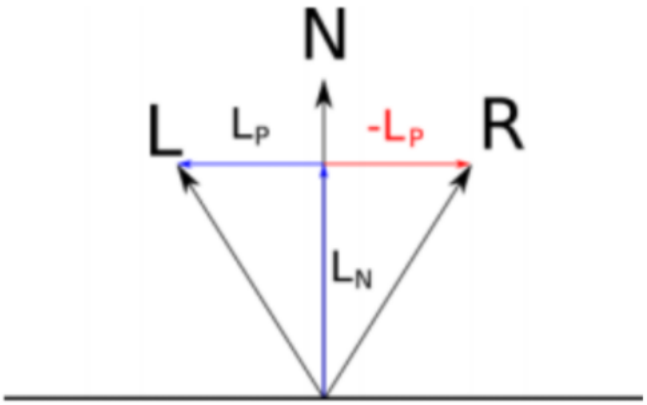


Рисунок 9 – Разложение вектора падающего луча

$\bar{L}_n$ - проекция $\bar{L}$ на $\bar{N}$. По свойства скалярного произведения и исходя из того, что |N| = 1, длина этой проекции равна $(\bar{N}, \bar{L})$, поэтому:

$$O + t \cdot D = (1 - u - v) \cdot V_0 + u \cdot V_1 + v \cdot V_2 \tag{14}$$

Отсюда

$$\bar{L}_p = \bar{L} - \bar{L}_n = \bar{L} - \bar{N}(\bar{N}, \bar{L}) \tag{15}$$

Очевидно, что:

$$\bar{R} = \bar{L}_n - \bar{L}_p \tag{16}$$

Подставив полученные ранее выражения и упростив, можно получить формулу отраженного луча:

$$\bar{R} = 2\bar{N}(\bar{N}n\bar{L}) - \bar{L} \tag{17}$$

2.6 Расчет интенсивностей

$$I_d = k_d \sum_{i=1}^{n} I(\overrightarrow{N}, \overrightarrow{L_i}) \tag{18}$$

Интенсивность света, диффузно отражающегося в точке поверхности, можно вычислить следующим образом (не зависит от положения наблюдателя):

- kd - коэффициент диффузного отражения,
- Ii - интенсивность света, попадающего в точку от i-го источника освещения,
- N - нормаль к поверхности в данной точке,
- Li - единичный вектор, совпадающий по направлению с вектором, проведенным из i-го источника в рассматриваемую точку
- n - количество всех источников в сцене

Интенсивность света, отраженного зеркально, может быть вычислена следующим образом (зависит от положения наблюдателя):

$$I_s = k_s \sum_{i=1}^{n} I(\overrightarrow{S}, \overrightarrow{R_i})^{n_p} \tag{19}$$

- ks - коэффициент зеркального отражения,
- S - единичный вектор, совпадающий по направлению с вектором из рассматриваемой точки в точку наблюдения,
- Ri - единичный вектор, задающий направление отраженного луча от i-го источника, - количество всех источников в сцене,
- np - степень, аппроксимирующая пространственное распределение зеркально отраженного света

Общую интенсивность можно определить по формуле:

$$I_r = k_a \sum_{i=1}^{n} I_i a + k_d \sum_{i=1}^{n} I(\overrightarrow{N}, \overrightarrow{L_i}) + k_s \sum_{i=1}^{n} I(\overrightarrow{S}, \overrightarrow{R_i})^{n_p} + k_r I_r + k_t I_t \qquad (20)$$

2.7 Описание входных данных

В данной программе входные данные подаются в виде файла с расширением .obj. В этом формате помимо координат вершин можно передавать информацию о текстурах и нормалях. Формат файла:

1. Список вершин с координатами (x,y,z).

v 0.46 0.13 1

2. Текстурные координаты (u,v).

vt 0.610 0.5

3. Координаты нормалей (x,y,z).

vn 0.8 0.0 0.4

4. Определение поверхности (сторон) задается в формате i1/i2/i3, где i1 - индекс координаты вершины, i2 - индекс координаты текстуры, i3 - индекс координаты нормали.

2/3/3 2/4/4 1/2/4

3 Технологический раздел

Необходимо сделать выбор средств разработки программного обеспечения, так как выбор языка программирования зависит от дальнейшей поддержки и возможностей разработки разрабатываемого программного обеспечения. Кроме того, опишите структуру программы, взаимодействие пользователя с графическим интерфейсом, предоставьте списки функций, описание которых потребуется в исследовательской части.

3.1 Средства реализации

Для решения поставленных в курсовом проекте задач был выбран язык программирования C++, поскольку:

- данный язык освоен ранее на практических занятиях;
- основной парадигмой в данном языке является ООП, что позволит разбить компоненты сцены на соответствующие классы;
- C++ достаточно производительный, это будет большим плюсом при решении такой трудозатратной задачи, как трассировка лучей.

В качестве IDE был выбран "QT Creator"по следующим причинам:

- знаком с данной IDE, т.к. делал в ней лабораторные работы по курсу «ООП»;
- позволяет очень быстро создавать графические интерфейсы любой сложности.

3.2 Описание структуры программы

В программе реализованы следующие классы:

- class Manager - описывает сцену и методы работы с ней;
- class Model - описывает способ хранения объекта и методы работы с ним;
- class Light - описывает атрибуты источников света различного света и содержит функции для работы со светом;
- class PixelShader - содержит функции для вычисления атрибутов объекта в конкретном пикселе;
- class VertexShader - содержит функции для преобразования атрибутов модели при переходе к мировому пространству из объектного;
- class TextureShader - содержит функции для интерполяции значения текстурных координат в конкретном пикселе;
- class GeometryShader - содержит функции для перехода из мирового пространства в пространство нормализованных координат;
- class Face - описывает полигон-треугольник;
- class Vertex - сожердит информацию о вершине полигона;
- class Vec3, class Vec4 - описывают вектора размерности 3 и 4, содержат функции для работы с ними;
- class Mat - описывает матрицы произвольного размера и содержит функции для работы с ними;
- class Camera - описывает камеру;

- class BoundingBox - описывает параллельный осям ограничивающий параллелепипед и содержит функции для работы с ним;
- class RayThread - содержит функции для выполнения трассировки.

На листинге 1 представлена функция, отрисовывающая модель

Листинг 1 – Функция отрисовки модели

```
void SceneManager::rasterize(Model& model){
    auto cam = camers[curr_camera];
    auto rotation_matrix = model.rotation_matrix;
    auto objToWorld = model.objToWorld();
    auto viewMatrix = cam.viewMatrix();
    auto projMatrix = cam.projectionMatrix;

    for (auto& face: model.faces){
        auto a = vertex_shader->shade(face.a, rotation_matrix, objToWorld,
    cam);
        auto b = vertex_shader->shade(face.b, rotation_matrix, objToWorld,
    cam);
        auto c = vertex_shader->shade(face.c, rotation_matrix, objToWorld,
    cam);

        if (backfaceCulling(a, b, c))
            continue;

        a = geom_shader->shade(a, projMatrix, viewMatrix);
        b = geom_shader->shade(b, projMatrix, viewMatrix);
        c = geom_shader->shade(c, projMatrix, viewMatrix);

        rasterBarTriangle(a, b, c);
    }
}
```

На листинге 2 представлена функция растеризации треугольника

Листинг 2 – Функция растеризации треугольника

```
#define Min(val1, val2) std::min(val1, val2)
#define Max(val1, val2) std::max(val1, val2)
void SceneManager::rasterBarTriangle(Vertex p1_, Vertex p2_, Vertex p3_){
    if (!clip(p1_) && !clip(p2_) && !clip(p3_))
        return;
    denormolize(width, height, p1_);
    denormolize(width, height, p2_);
    denormolize(width, height, p3_);
    auto p1 = p1_.pos;
    auto p2 = p2_.pos;
    auto p3 = p3_.pos;
    float sx = std::floor(Min(Min(p1.x, p2.x), p3.x));
    float ex = std::ceil(Max(Max(p1.x, p2.x), p3.x));
    float sy = std::floor(Min(Min(p1.y, p2.y), p3.y));
    float ey = std::ceil(Max(Max(p1.y, p2.y), p3.y));
    for (int y = static_cast<int>(sy); y < static_cast<int>(ey); y++){
        for (int x = static_cast<int>(sx); x < static_cast<int>(ex); x++){
            Vec3f bary = toBarycentric(p1, p2, p3, Vec3f(static_cast<float
>(x), static_cast<float>(y)));
            if ( (bary.x > 0.0f || fabs(bary.x) < eps) && (bary.x < 1.0f
|| fabs(bary.x - 1.0f) < eps) &&
                 (bary.y > 0.0f || fabs(bary.y) < eps) && (bary.y < 1.0f
|| fabs(bary.y - 1.0f) < eps) &&
                 (bary.z > 0.0f || fabs(bary.z) < eps) && (bary.z < 1.0f
|| fabs(bary.z - 1.0f) < eps)){
                auto interpolated = baryCentricInterpolation(p1, p2, p3,
bary);
                interpolated.x = x;
                interpolated.y = y;
                if (testAndSet(interpolated)){
                    auto pixel_color = pixel_shader->shade(p1_, p2_, p3_,
bary) * 255.f;
                    img.setPixelColor(x, y, qRgb(pixel_color.x,
pixel_color.y, pixel_color.z));
                }}}}}
```

На листинге 3 представлена функция нахождения пересечения с ограничивающим параллелепипедом

Листинг 3 – Функция нахождения пересечения с ограничивающим параллелепипедом

```
bool BoundingBox::intersect(const Ray &r) const{
    float tmin, tmax, tymin, tymax, tzmin, tzmax;

    tmin = (bounds[r.sign[0]].x - r.origin.x) * r.invdirection.x;
    tmax = (bounds[1-r.sign[0]].x - r.origin.x) * r.invdirection.x;
    tymin = (bounds[r.sign[1]].y - r.origin.y) * r.invdirection.y;
    tymax = (bounds[1-r.sign[1]].y - r.origin.y) * r.invdirection.y;

    if ((tmin > tymax) || (tymin > tmax))
        return false;
    if (tymin > tmin)
        tmin = tymin;
    if (tymax < tmax)
        tmax = tymax;

    tzmin = (bounds[r.sign[2]].z - r.origin.z) * r.invdirection.z;
    tzmax = (bounds[1-r.sign[2]].z - r.origin.z) * r.invdirection.z;

    if ((tmin > tzmax) || (tzmin > tmax))
        return false;
    if (tzmin > tmin)
        tmin = tzmin;
    if (tzmax < tmax)
        tmax = tzmax;

    return (SIGN(tmin) == SIGN(tmax));

}
```

3.3 Описание интерфейса

На рисунке 10 представлен интерфейс программы. Он предусматривает возможность вращения, перемещения, масштабирования объектов, изменения свойств, цвета, текстуры их поверхности; добавления и удаления моделей и источников света, изменение параметров источников. Для обзора сцены используется камера, управление которой осуществляется посредством нажатия клавиш на клавиатуре.

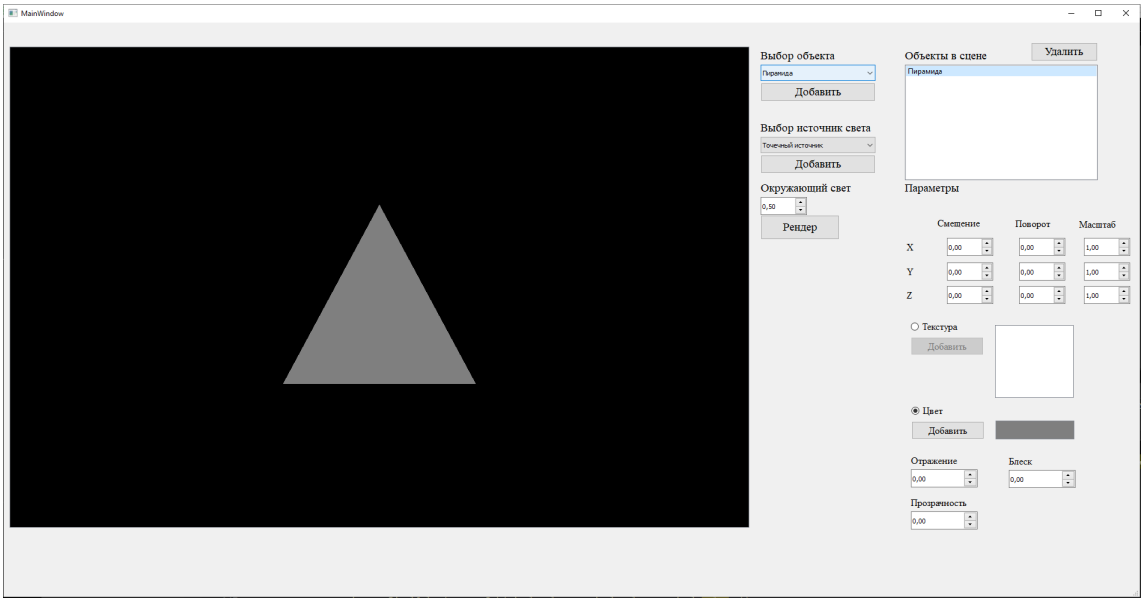


Рисунок 10 – Интерфейс программы

Для добавления модели на сцену нужно выбрать ее из списка (рисунок 11) и нажать на кнопку "добавить"под ним, после чего она будет отрисована на экране и попадет в список объектов в сцене в правом углу (рисунок 12).

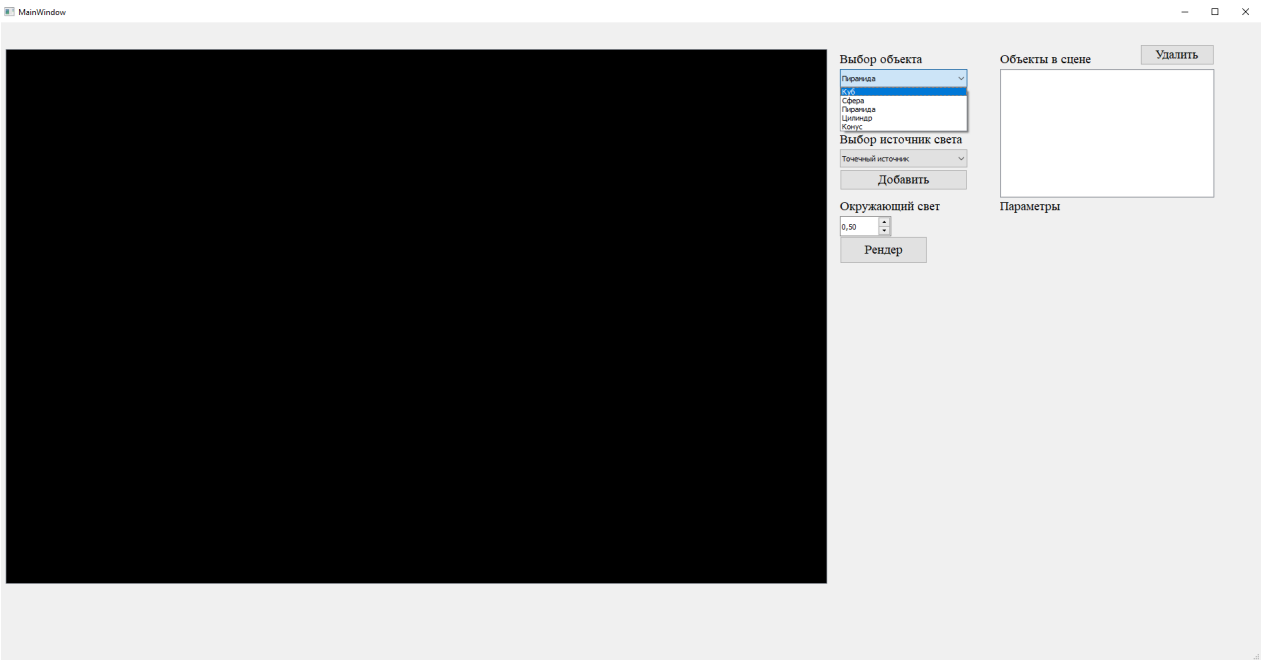


Рисунок 11 – Список моделей для добавления

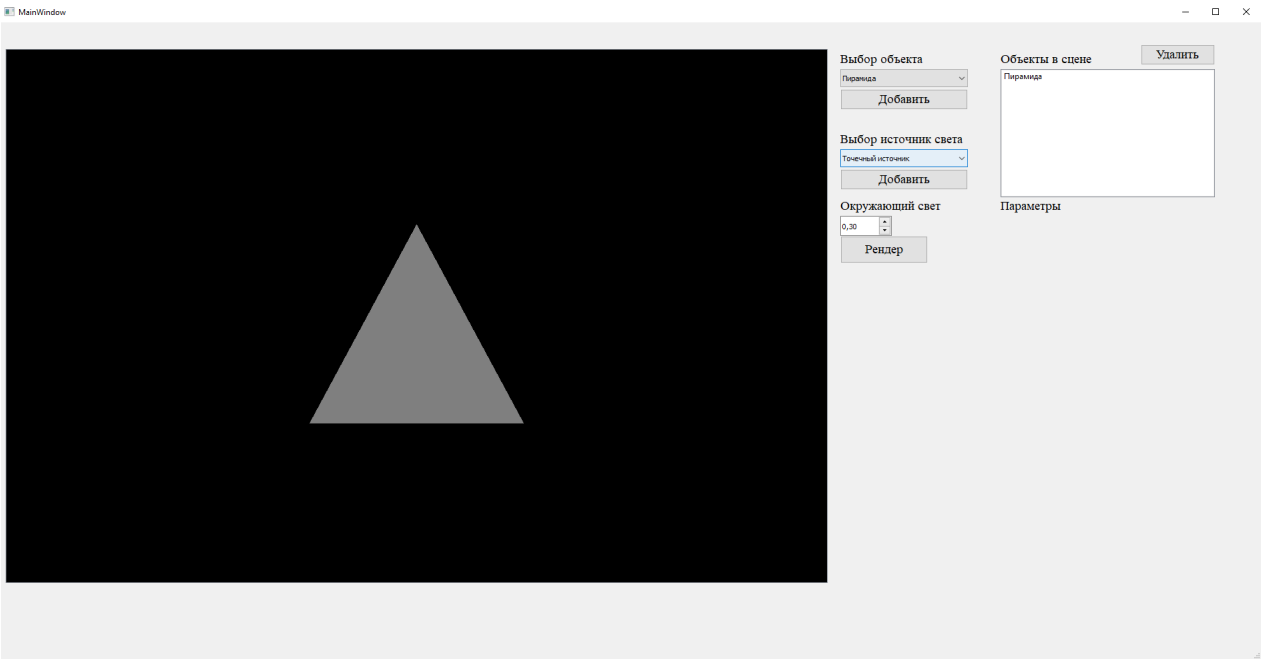


Рисунок 12 – Список объектов в сцене в правом верхнем углу

Для изменения параметров модели нужно выбрать его из списка объектов в сцене, который находится в правом верхнем углу. После нажатия на название модели, в правой части экрана появится список параметров, доступных для изменения (рисунок 13). Изменение цвета и текстуры объекта

происходит путем взаимодействия с диалоговыми окнами как показано на рисунке 14 и 15, которые открываются при нажати на соответствующие кнопки "Добавить". Для переключения между цветом и текстурой используются элементы RadioButton (рисунок 13).

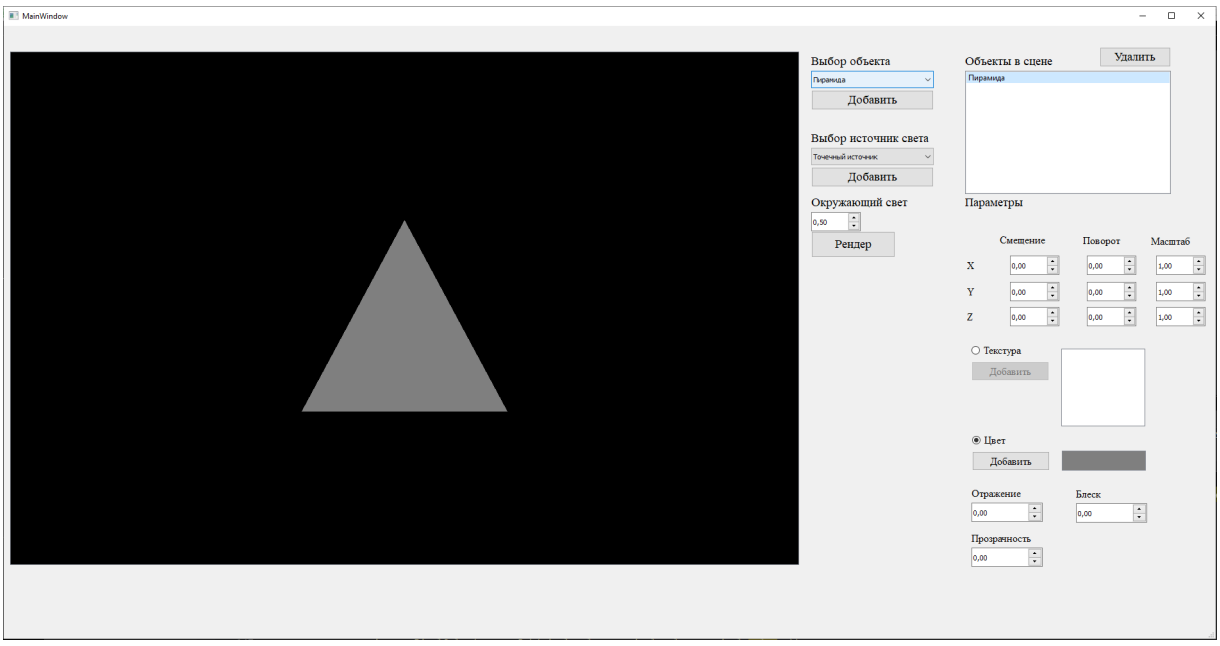


Рисунок 13 – Список параметров модели

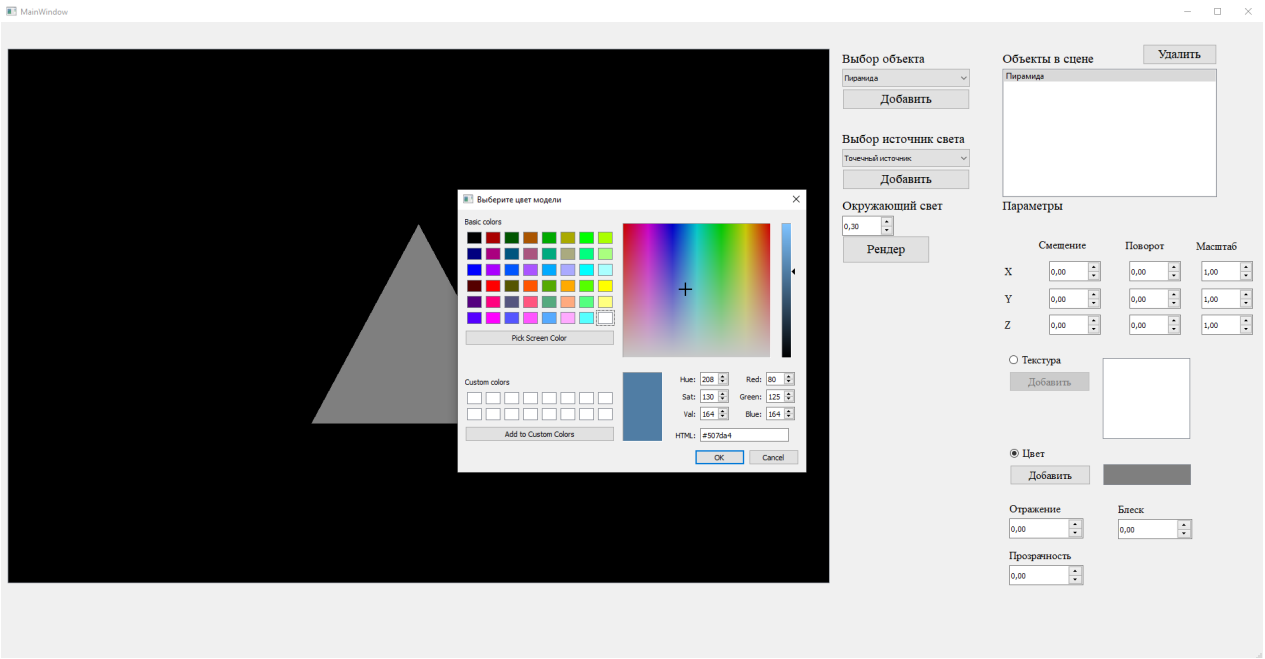


Рисунок 14 – Диалоговое окно для выбора цвета

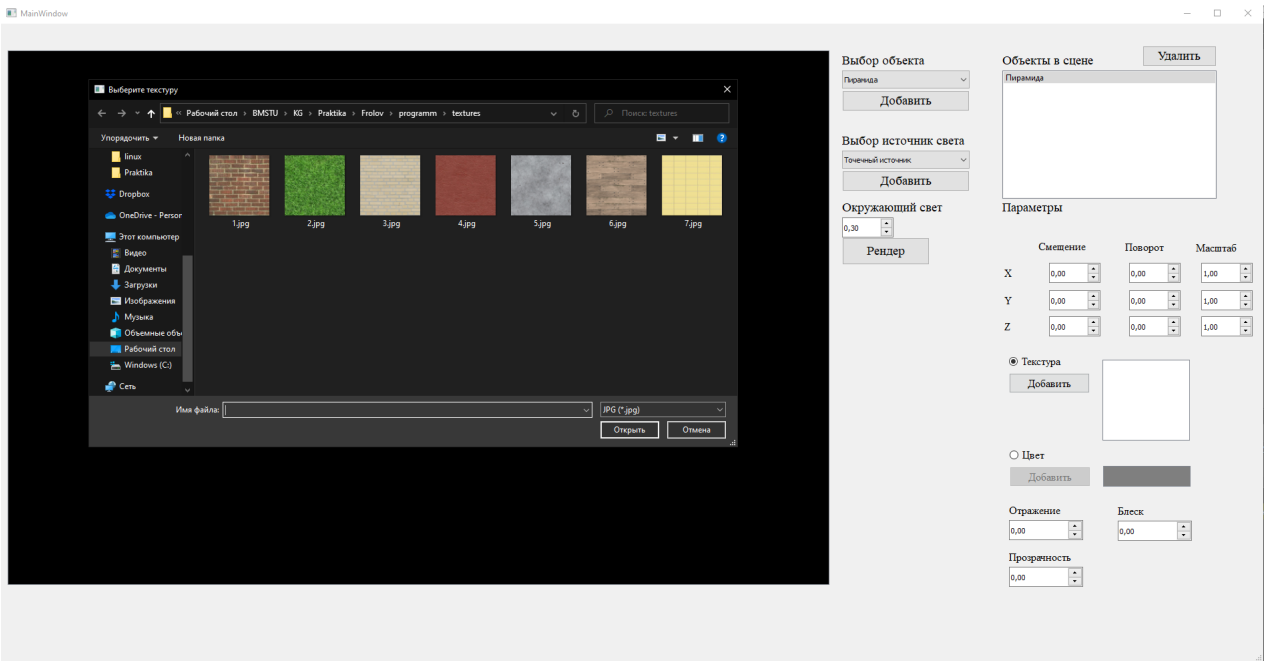


Рисунок 15 – Диалоговое окно для выбора текстуры

Источники света также выбираются из списка (рисунок 16) и при нажатии на кнопку "добавить"добавляются в список объектов, расположенный в правом верхнем углу, и отрисовываются на сцене. Точечный источник имеет форму сферы, а направленныыформу "пули"(рисунок 17). При нажатии на источник в правой части появляется меню изменения его параметров(рисунок 17). Интенсивность окружающего освещения меняется в левой части экрана посредством взаимодействия с SpinBox.

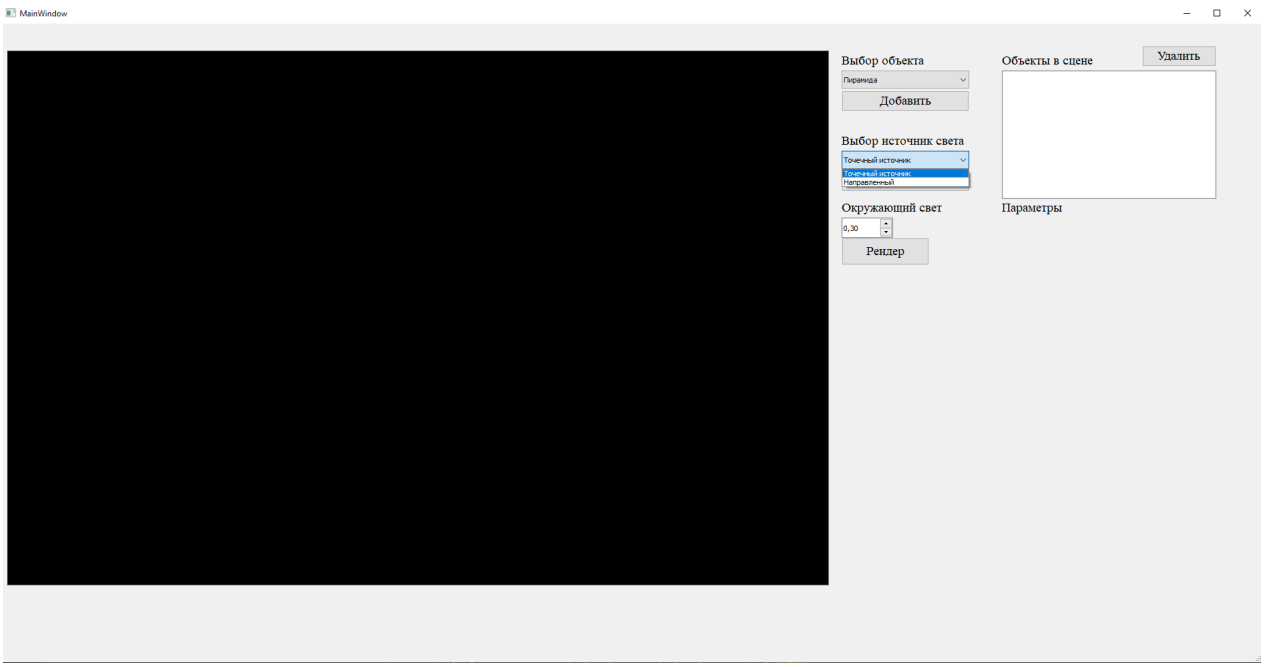


Рисунок 16 – Список источников света

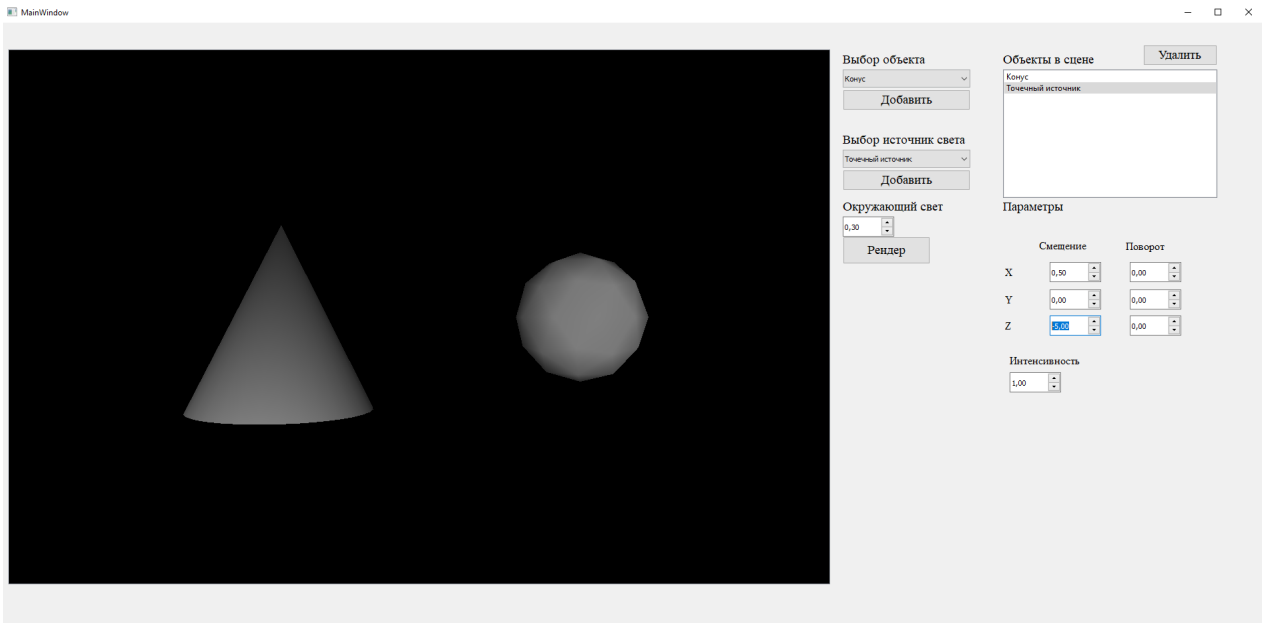


Рисунок 17 – Вид источников света в сцене

После добавления объектов и источников в сцену, задания необходимых параметров и выбора положения обзора, нажатие на кнопку "Рендер"запустит процесс генерации изображения с применением алгоритма обратной трассировки лучей, интерфейс будет заблокирован пока изображение не будет готово (рисунок 18). После получения результата (рисунок

19), к предыдущему режиму можно вернуться, нажав на кнопку "Рендер".

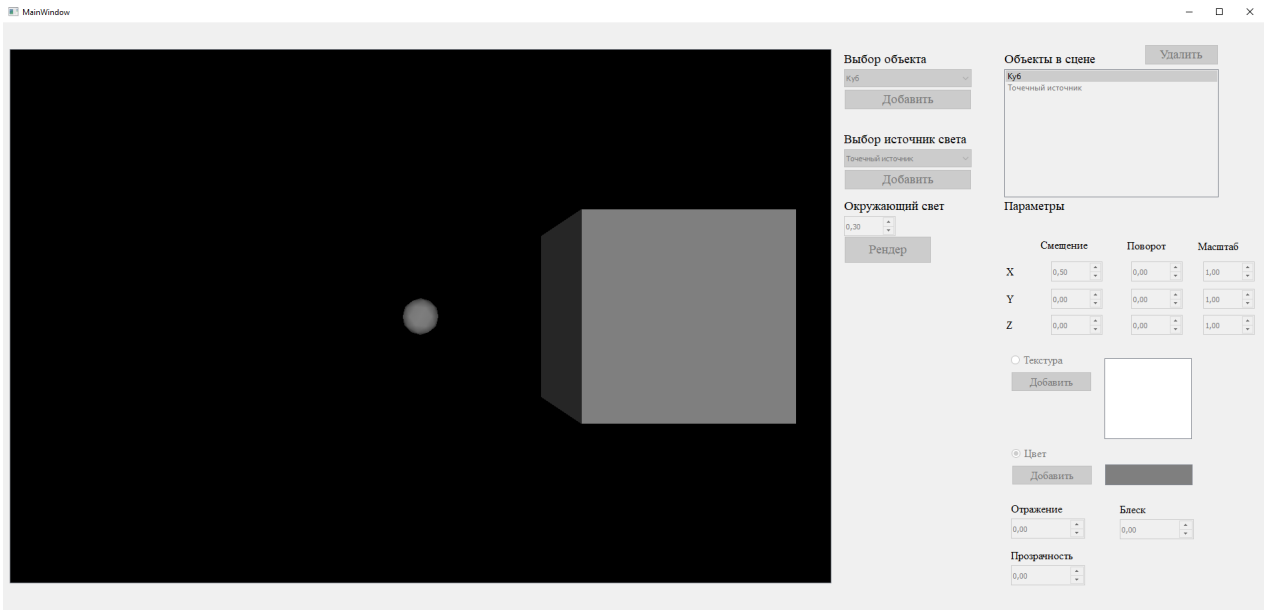


Рисунок 18 – Блокировка интерфейса в процессе выполнения рендеринга

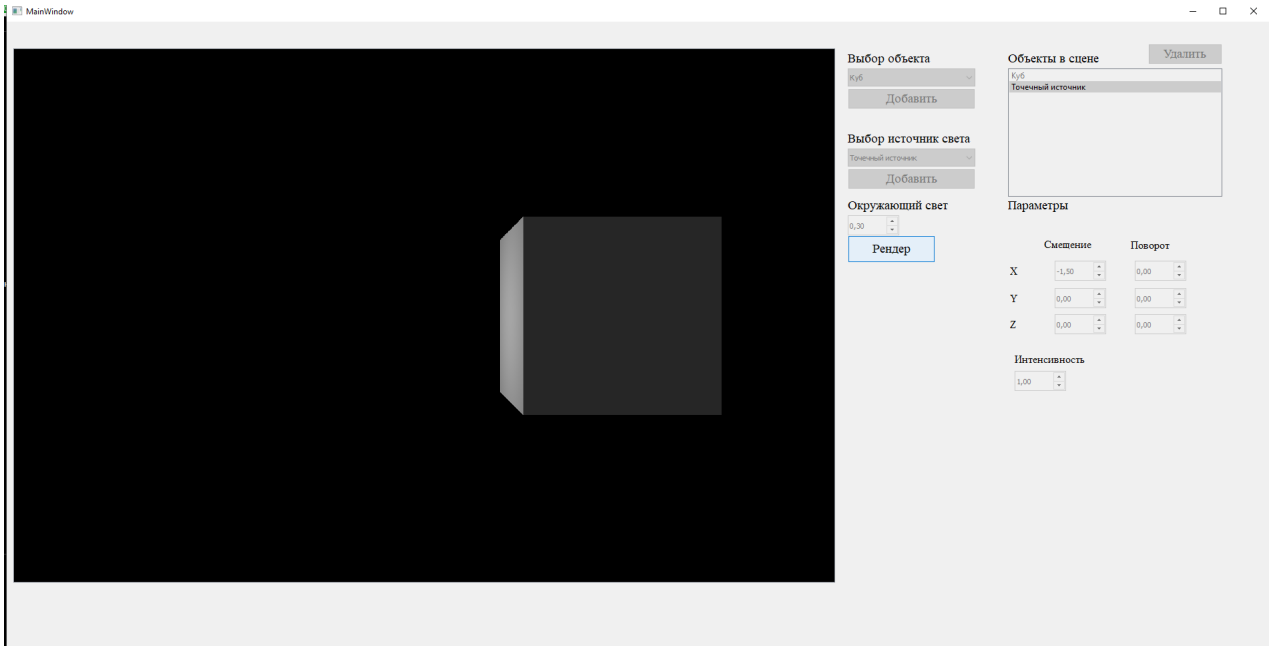


Рисунок 19 – Результат рендеринга

4 Исследовательский раздел

В алгоритме обратной трассировки лучи трассируются независимо друг от друга, это дает возможность проводить отрисовку изображения параллельно. Помимо этого приростапроизводительности можно достичь при использовании улучшений, призванных уменьшить количество обрабатываемых полигонов на каждом трассировании луча.

Исследование проводится для поиска таких улучшений и определения числа потоков, при котором достигается минимальное время рендеринга сцены.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Windows 10 64-bit;
- Память 16 GiB; Процессор: AMD Ryzen 5 4600H with Radeon Graphics.

4.2 Описание экспериментов

Для проведения замеров времени в экспериментах будет использована формула.

$$t = \frac{T_n}{N} \tag{21}$$

где

- N - количество потоков;
- t - время выполнения реализации алгоритма;
- T_n - время выполнения N замеров

4.2.1 Зависимость времени работы алгоритма обратной трассировки лучей от количества потоков программы

Одним из преимуществ алгоритма обратной трассировки лучей является то, что его можно распараллелить, так как лучи не влияют на работу друг-друга и могут трассироваться параллельно[3].

На рисунке 20 изображен график зависимости рендеринга сцены, состоящей из конуса и куба с одним точечным источником света, от количества потоков.

Принцип разделения обязанностей был следюущим: картинка делилась на n частей по диагонали, где n - количество потоков. В итоге каждый поток получал строчку, в рамках которой и трассировал лучи.

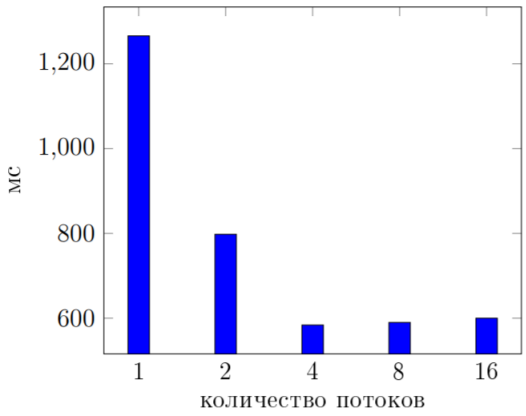


Рисунок 20 – Диаграмма времени выполнения алгоритма в зависимости от количества потоков

Как видно из рисунка, начиная с четырех, увеличение числа потоков не дало сильного прироста в скорости, это связано с тем, что число потоков, которые работают параллельно, равно числу логических процессоров, которых на экспериментальном процессоре четыре.

4.2.2 Зависимость времени работы алгоритма обратной трассировки лучей от алгоритмов пересечения

Так как каждая модель состоит из полигонов, крайне неэффективно на каждом трассировании луча искать пересечение с объектом путем установления факта пересечения с его полигонами.

Для оптимизации можно использовать следующую идею: ограничить каждый объект параллелепипедом, который параллелен координатным осям (AABB - axis-aligned bounding box). Таким образом при каждом трассировании луча пересечение будет вычисляться сначала с этим параллелепипедом (листинг 3.3), и если оно было найдено, то далее с полигонами этого объекта.

Наибольшее количество полигонов у модели сферы, поэтому эксперимент будет проводиться на ней (21). Замеры проводились 5 раз при 4 потоках.

Как видно из рисунка, использование AABB существенно ускоряет время работы программы, а применение ряда оптимизаций сокращает время отрисовки в среднем в 1.5 раза.

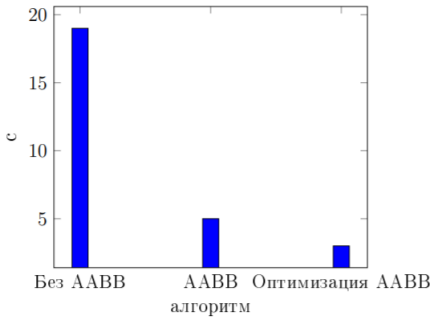


Рисунок 21 – Диаграмма времени отрисовки сферы в зависимости от алгоритма поиска пересечений луча с объектом

4.3 Демонстрация работы программы

На рисунках 22 и 23 представлены результаты работы программы.

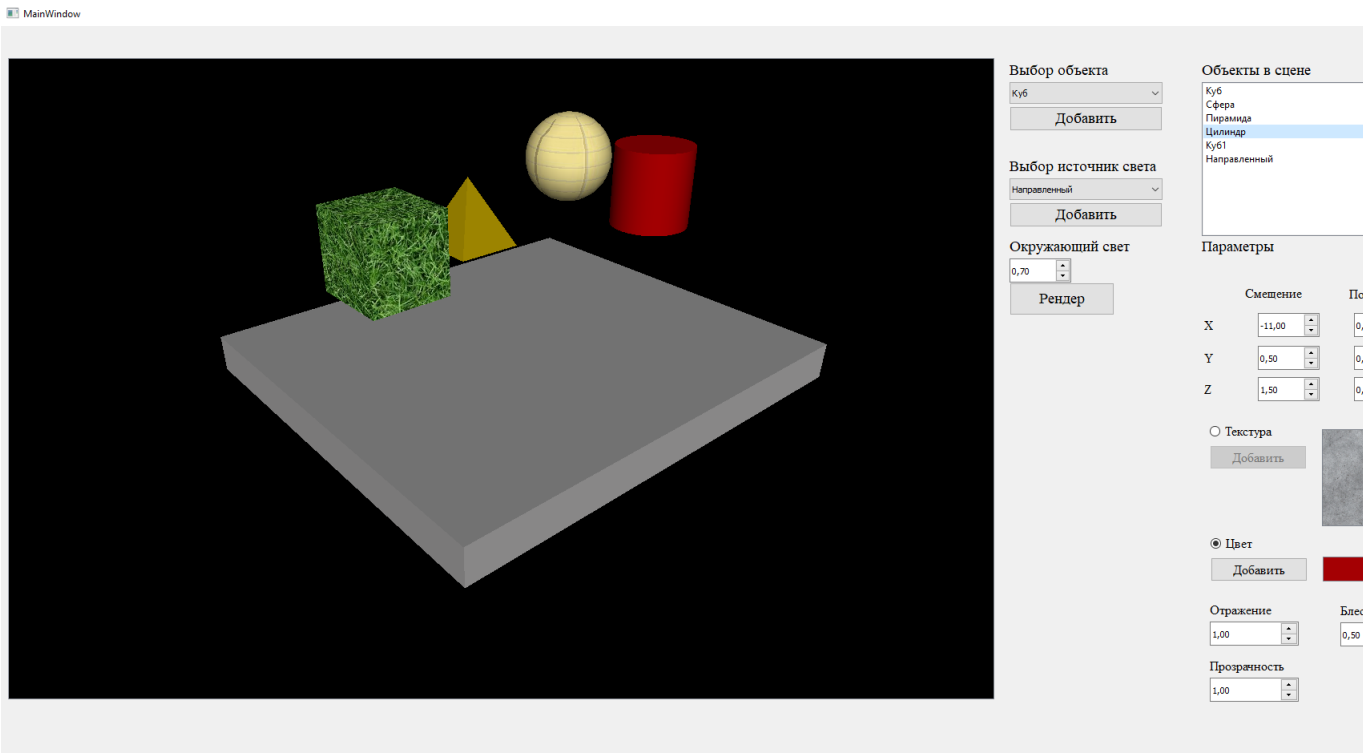


Рисунок 22 – Созданная сцена в конструкторе

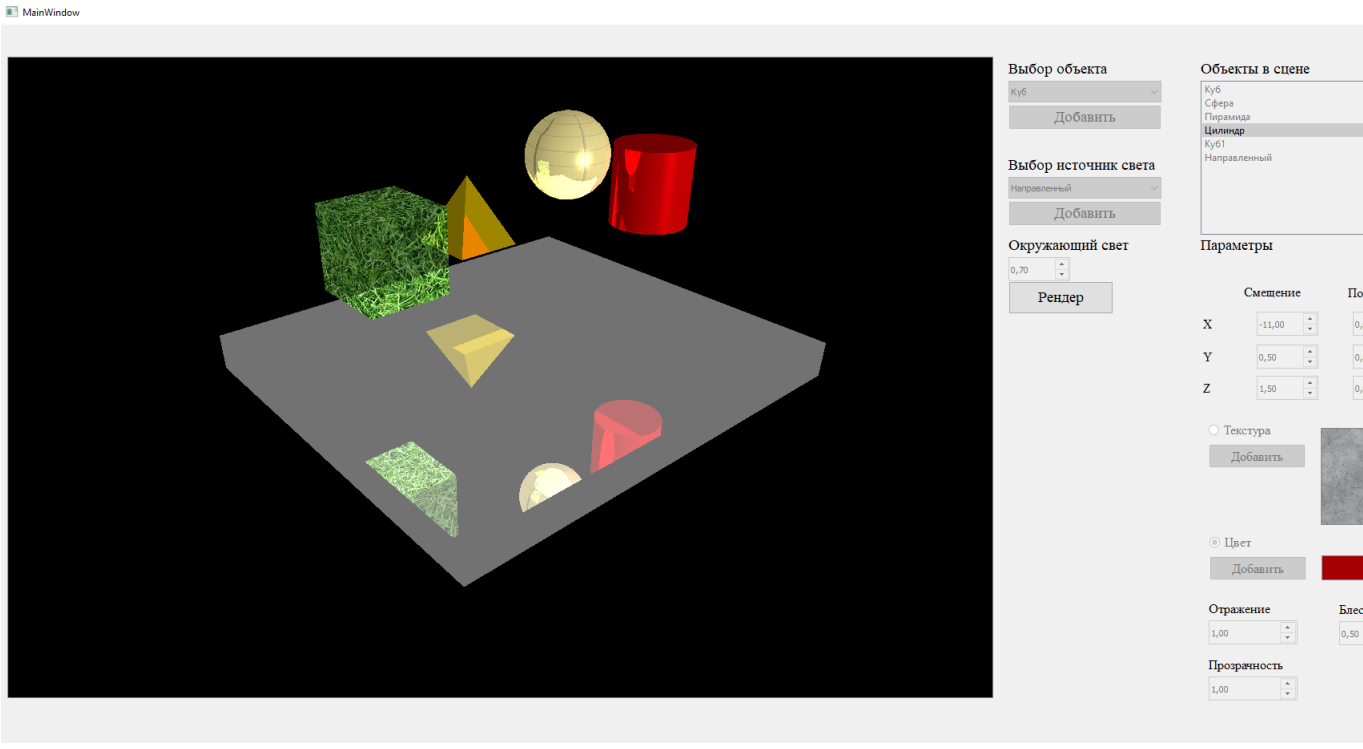


Рисунок 23 – Построенное реалистическое изображение

Заключение

В рамках курсового проекта было создано программное обеспечение для создания графических сцен из готовых трехмерных моделей и их визуализации с учетом выбранной текстуры или цвета, а также оптических эффектов отражения, преломления, прозрачности, блеска.

Были выполнены следующие задачи:

- проведен анализ существующих алгоритмов удаления невидимых линий и поверхностей, закраски, текстурирования, а также моделей освещения и выбраны подходящие для выполнения проекта;
- реализованы выбранные алгоритмы и структуры данных;
- разработано программное обеспечение, позволяющее отобразить трехмерную сцену;
- реализован интерфейс программного модуля;
- проведено исследование на основе разработанной программы.

Список литературы

[1] Роджерс Д. Математические основы машинной графики. / Роджерс Д., Адамс Дж. - М.: Мир, 1989. - 512с.

[2] Аксенов, Андрей. Компьютерная графика. [электронный ресурс]. Режим доступа: http://algolist.ru/graphics/3dfaq/index.php

[3] Проблемы трассировки лучей - из будущего в реальное время. [Электронный ресурс]. - Режим доступа: https://nvworld.ru/articles/ray-tracing/3/

[4] RayTracing - царь света и теней, Лев Дымченко [Электронный ресурс]. - Режим доступа: https://old.computerra.ru/206167/

Построение трехмерной сцены объектов

Студент:

- Фролов Евгений ИУ7-55Б

Научный руководитель:

- Романова Т.Н.

Москва 2021

Цель работы - разработать программу для построения реалистического изображения из трехмерных геометрических объектов

Реализованное ПО предоставляет возможность:

а) выбора и добавления в сцену трехмерных объектов;

б) перемещения, поворота и масштабирования объектов;

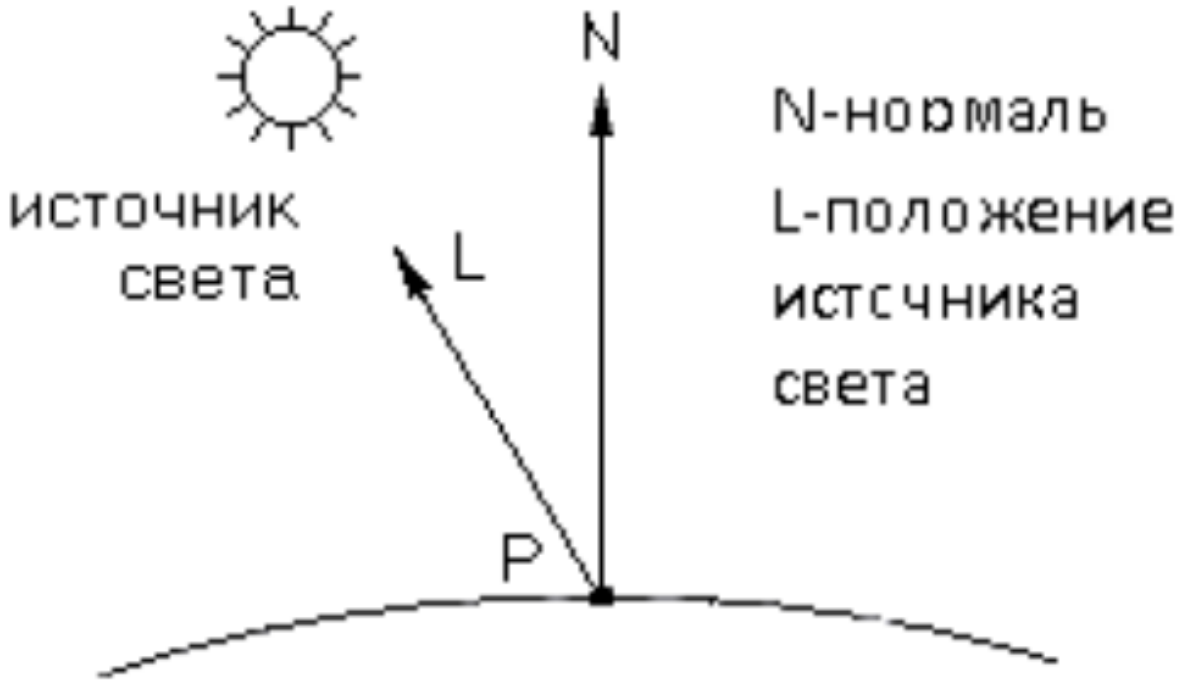
в) изменение текстуры объекта, его цвет, свойств поверхности.

Фролов Евгений ИУ7-55Б

Формализация сцены

- Трехмерные геометрические объекты;
- источник света;
- отражение фигур;
- текстуры фигуры.

Простой метод освещения



В простом методе освещения интенсивность рассчитывается по закону Ламберта:

$$I = I_0 * \cos(\alpha), \text{ где}$$

- I – результирующая интенсивность света в точке
- I_0 – интенсивность источника
- α – угол между нормалью к поверхности и вектором направления света

Фролов Евгений ИУ7-55Б

Перспективно-корректное текстурирование

При отрисовке каждая сканирующая строка разбивается на части, в начале и конце каждого куска считаются точные значения u, v, а в каждой части они интерполируется линейно.

Для каждой вершины посчитать 1/Z, u/Z, v/Z и линейно их интерполировать - точно так же, как интерполируются u и v в аффинном текстурировании

$$u = (u/z)/(1/z)$$
$$v = (v/z)/(1/z)$$

Фролов Евгений ИУ7-55Б

Алгоритм Z-буфера

1. Всем элементам буфера кадра присвоить фоновое значение

2. Инициализировать Z буфер минимальными значениями глубины

3. Выполнить растровую развертку каждого многоугольника сцены:

    a. Для каждого пикселя, связанного с многоугольником вычислить его глубину z(x, y)

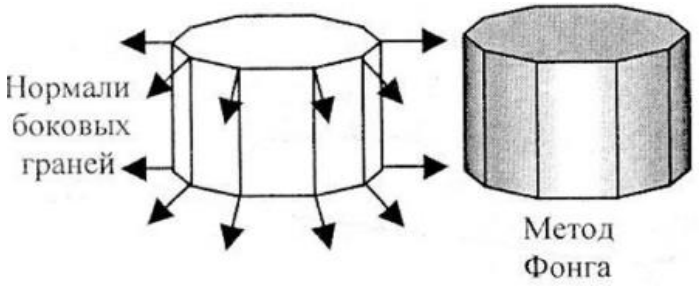
    b. Сравнить глубину пискселя со значением, хранимым в Z буфере.

        Если z(x, y) > zбуф(x, y), то zбуф(x,y) = z(x,y), цвет(x, y) = цветПикселя.

4. Отобразить результат

Фролов Евгений ИУ7-55Б

Процесс закраски по методу Фонга



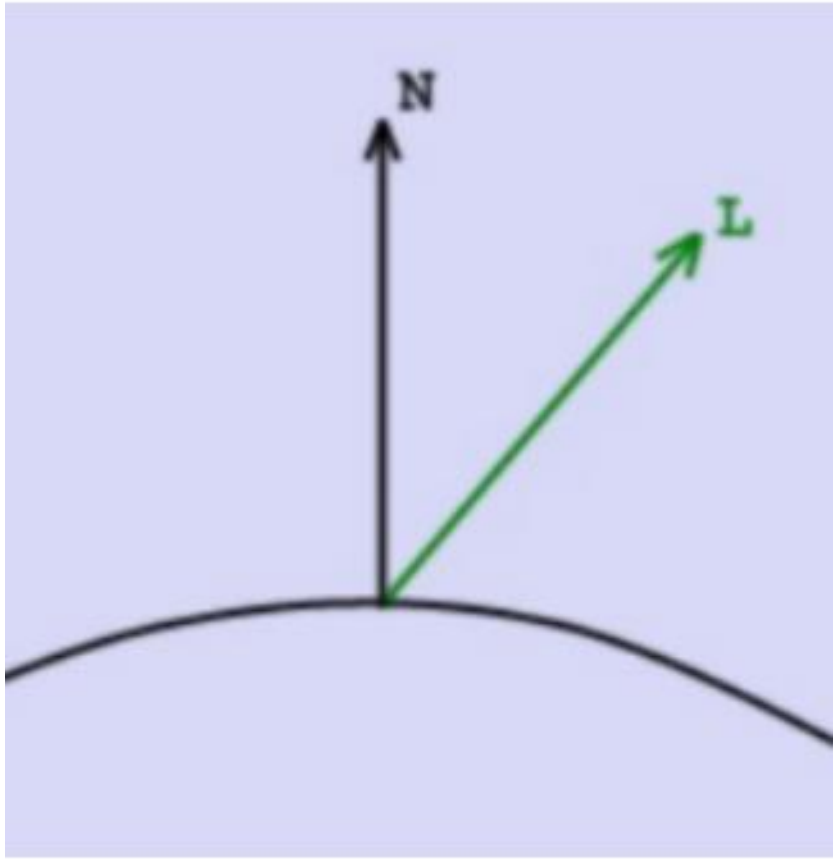
1. Определяются нормали к граням.
2. По нормалям к граням определяются нормали в вершинах.
3. В каждой точке закрашиваемой грани определяется интерполированный вектор нормали.
4. По направлению векторов нормали определяется цвет точек грани.

Модель Ламберта

Модель Ламберта моделирует идеальное диффузное освещение. Свет при попадании на поверхность рассеивается равномерно во все стороны.

Учитывается:

- ориентация поверхности (нормаль N)
- направление на источник света (вектор L).



Фролов Евгений ИУ7-55Б

Вывод по алгоритмам, моделирующих освещение

Для расчета интенсивности в точке выбрана модель Ламберта при использованиии z-буфера

Модель Фонга, при использовании обратной трассировки лучей

Фролов Евгений ИУ7-55Б

Выбор языка программирования и среды разработки

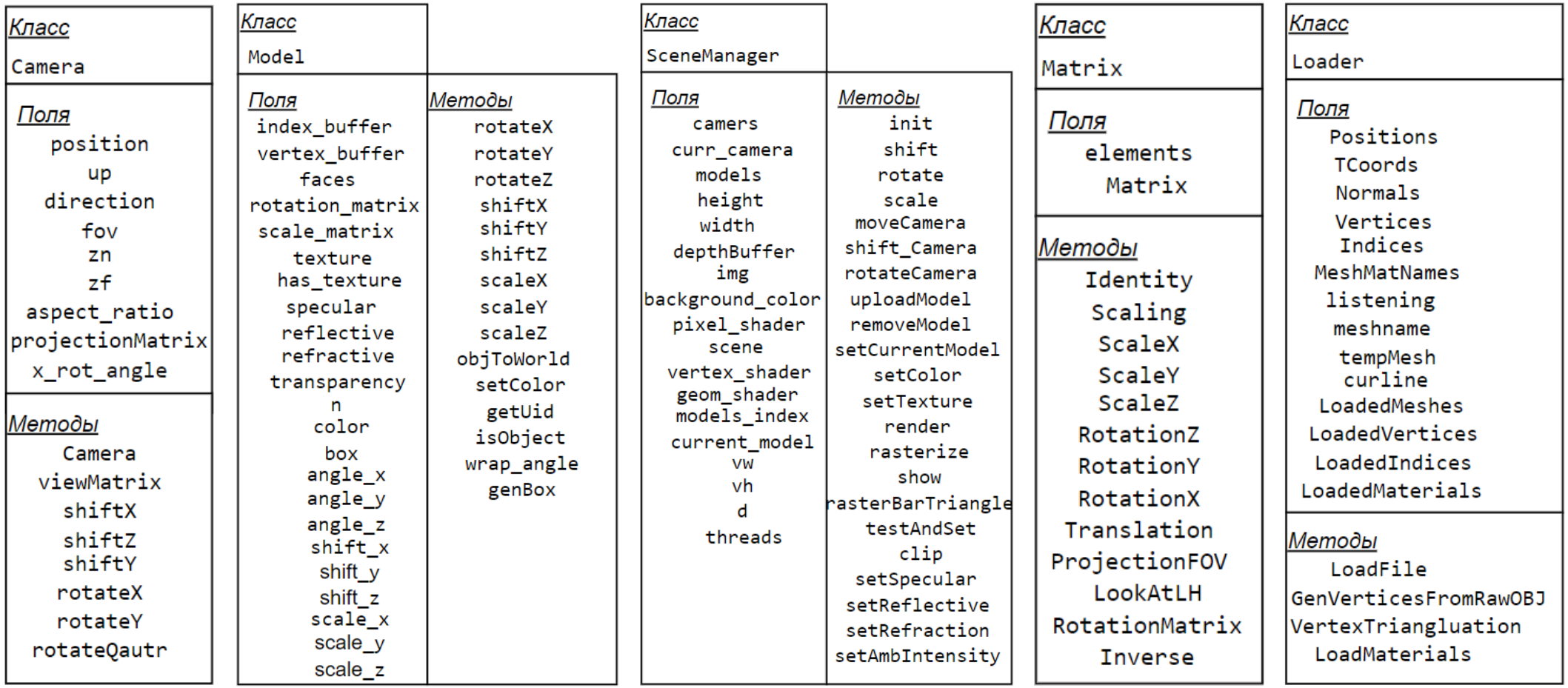
В качестве языка программирования был выбран язык C++:

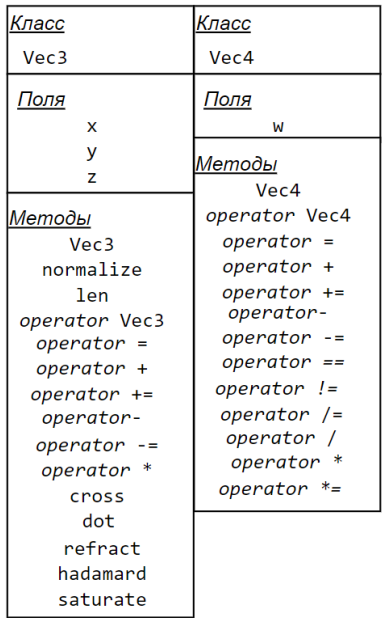
- т.к. работал с этим языком во время курса по предмету ООП.

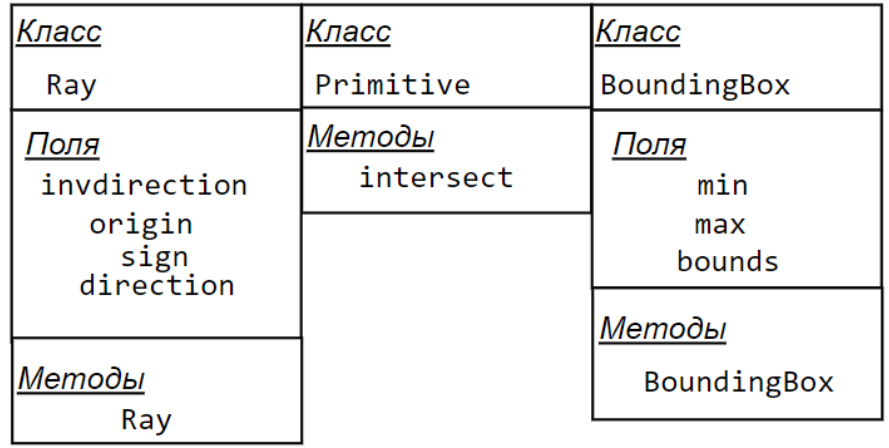
В качестве среды разработки была выбрана QtCreator:

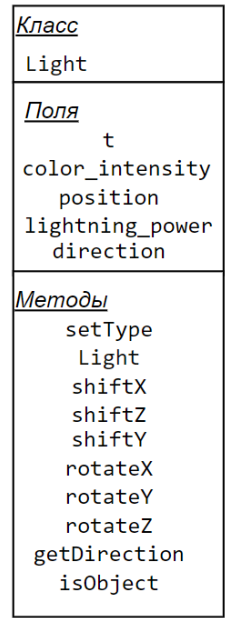
- бесплатный для студентов;
- удобства отладки и написания кода;
- удобное создание интерфейса.

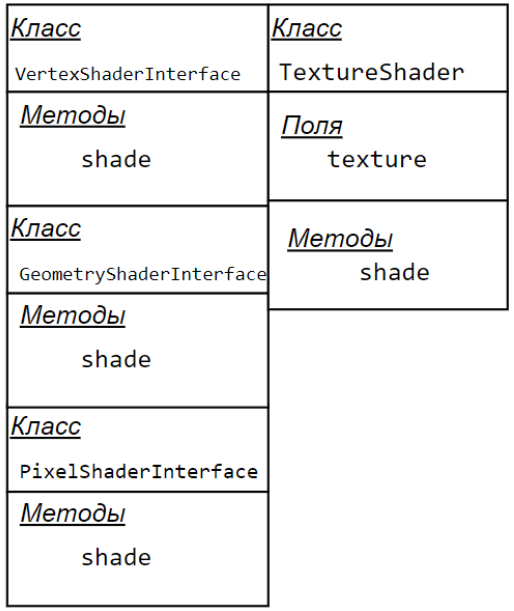
Структура и состав классов

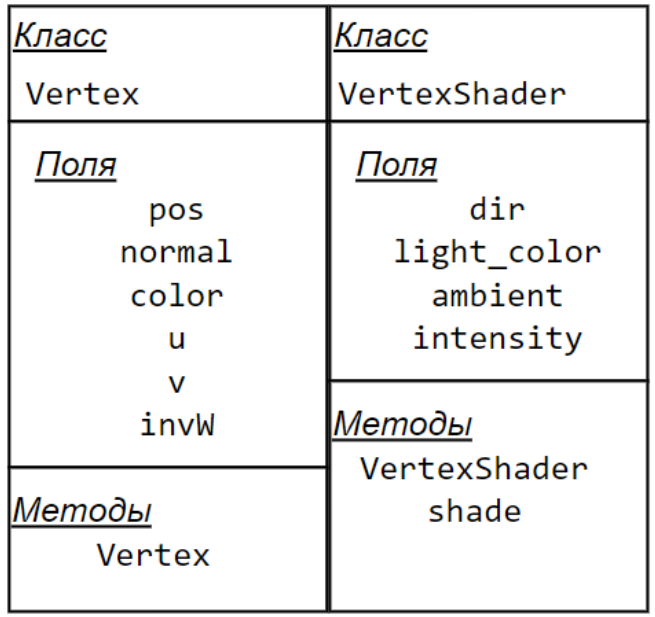


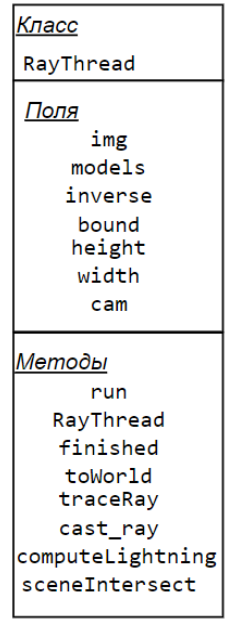

Класс
Vec3
Поля
x
y
z
Методы
Vec3
normalize
len
operator Vec3
operator =
operator +
operator +=
operator-
operator -=
operator *
cross
dot
refract
hadamard
saturate
Класс
Vec4
Поля
w
Методы
Vec4
operator Vec4
operator =
operator +
operator +=
operator-
operator -=
operator ==
operator !=
operator /=
operator /
operator *
operator *=


Класс
Ray
Поля
invdirection
origin
sign
direction
Методы
Ray
Класс
Primitive
Методы
intersect
Класс
BoundingBox
Поля
min
max
bounds
Методы
BoundingBox

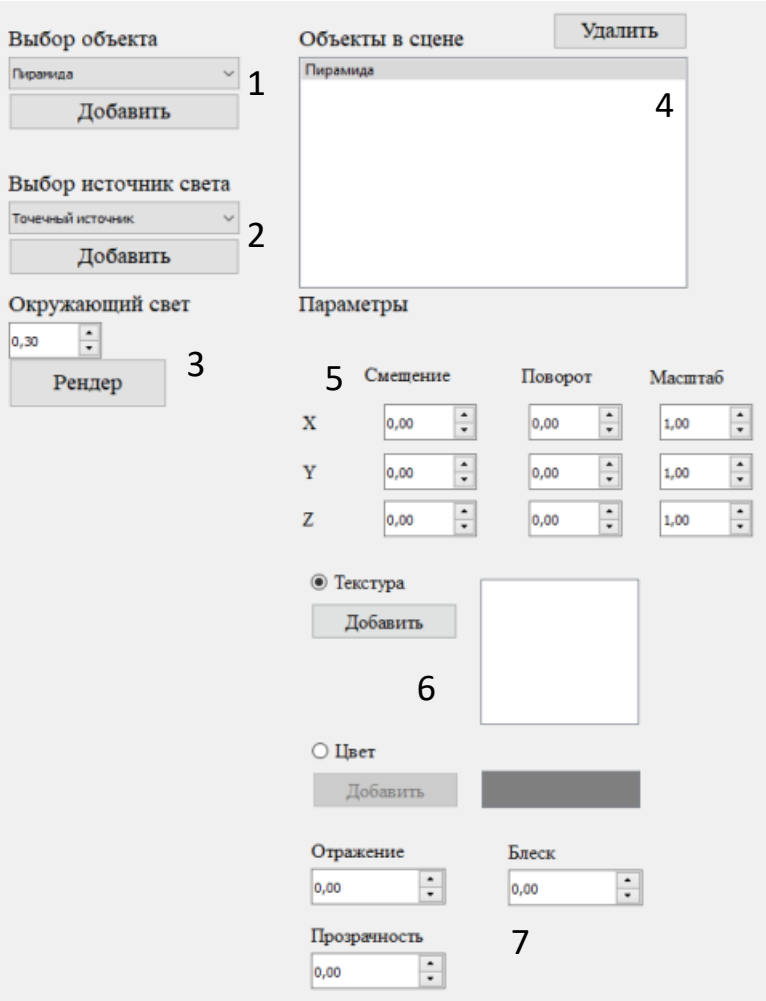

Класс
Light
Поля
t
color_intensity
position
lightning_power
direction
Методы
setType
Light
shiftX
shiftZ
shiftY
rotateX
rotateY
rotateZ
getDirection
isObject


Класс
VertexShaderInterface
Методы
shade
Класс
GeometryShaderInterface
Методы
shade
Класс
PixelShaderInterface
Методы
shade
Класс
TextureShader
Поля
texture
Методы
shade


Класс
Vertex
Поля
pos
normal
color
u
v
invW
Методы
Vertex
Класс
VertexShader
Поля
dir
light_color
ambient
intensity
Методы
VertexShader
shade


Класс
RayThread
Поля
img
models
inverse
bound
height
width
cam
Методы
run
RayThread
finished
toWorld
traceRay
cast_ray
computeLightning
sceneIntersect

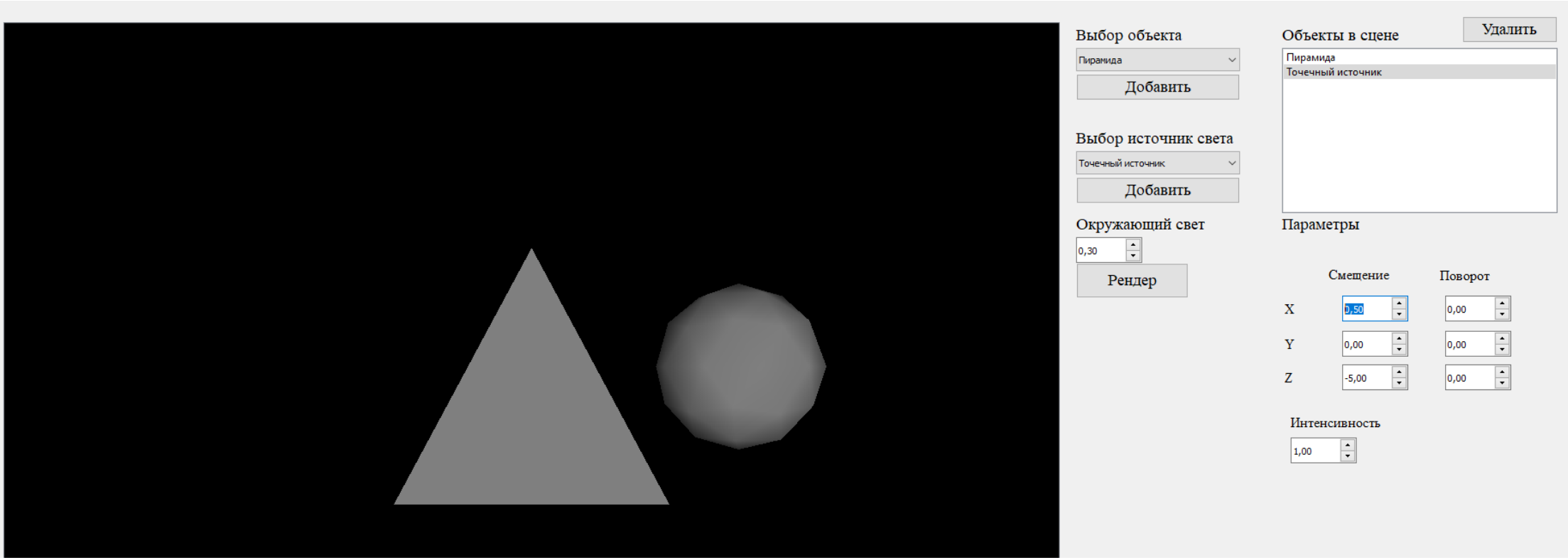
Интерфейс программы



1. Выбор объекта - добавление объектов в сцену
2. Выбор источника света
3. Изменение параметра окружающего света
4. Просмотр объектов сцены
5. Изменение положения объекта
6. Добавление текстуры и изменение цвета объекта
7. Изменение параметров отражения, блеска и прозрачности

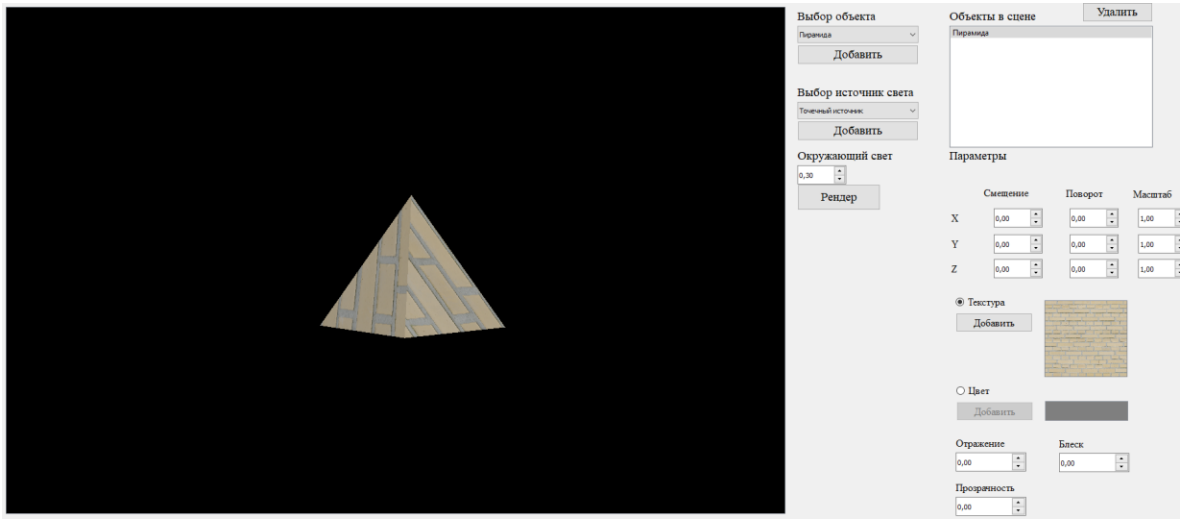
Фролов Евгений ИУ7-55Б

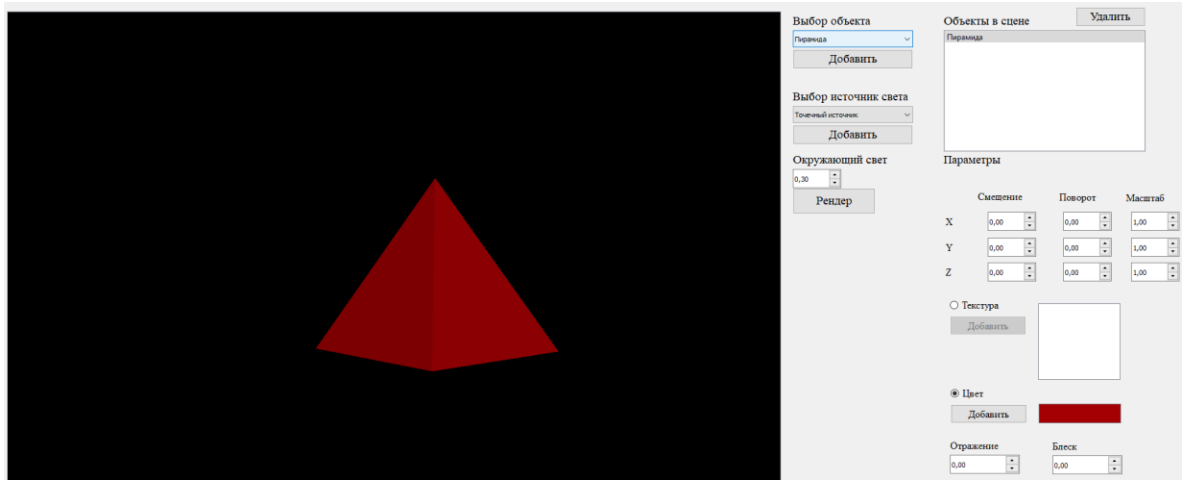
На сцену добавили пирамиду и источник света



Фролов Евгений ИУ7-55Б

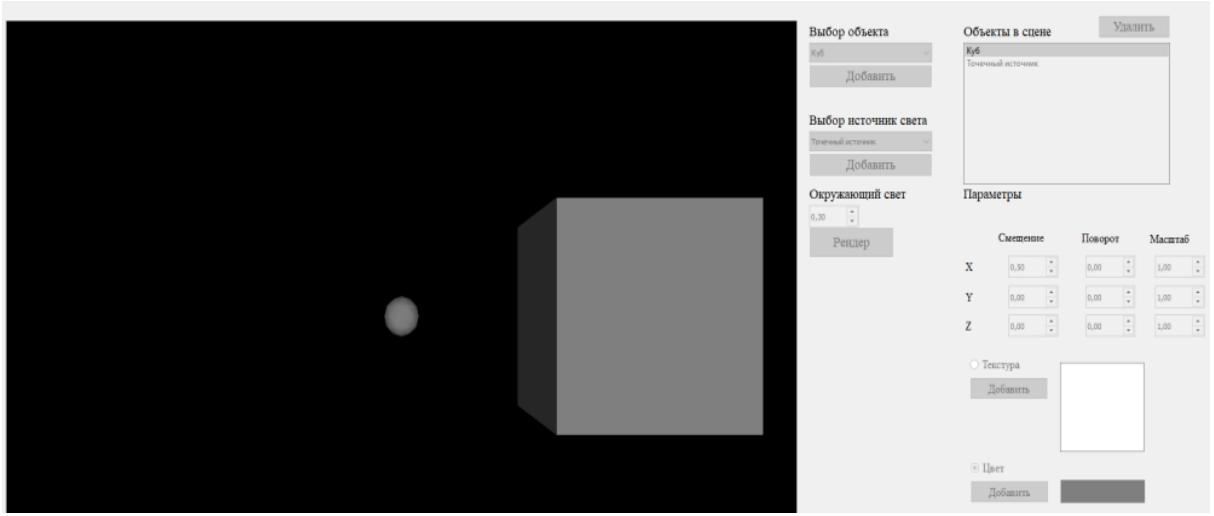
Наложение текстуры и цвета

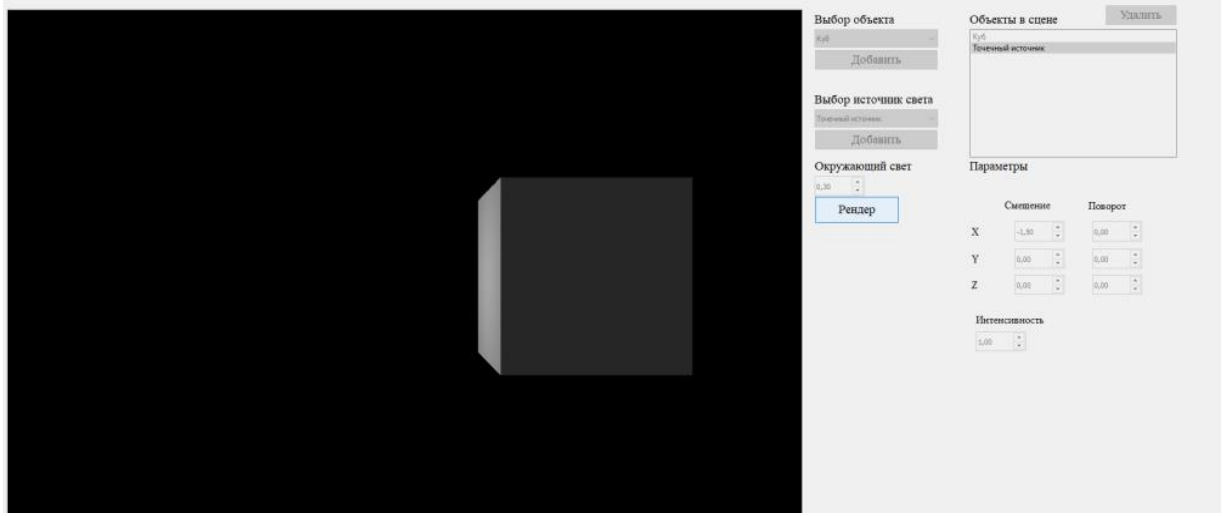




Фролов Евгений ИУ7-55Б

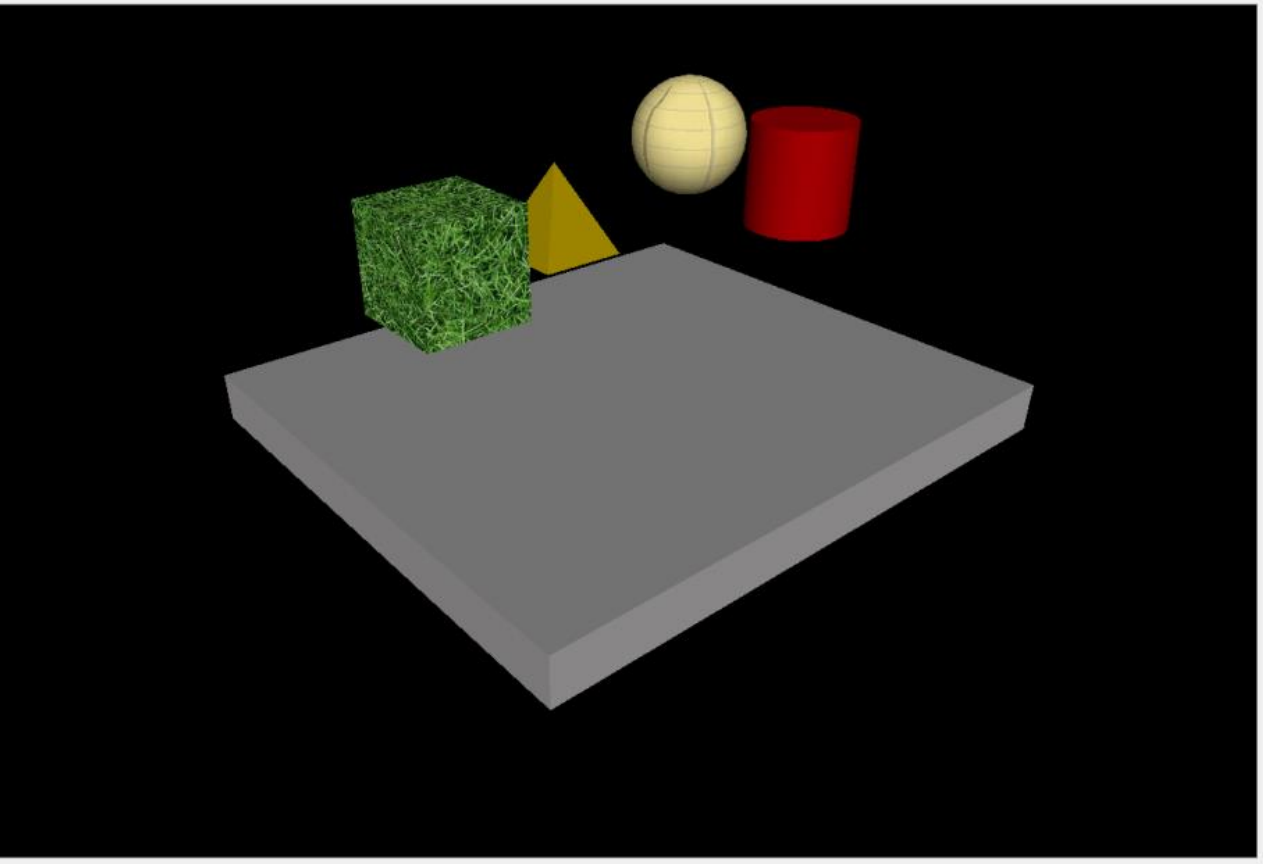
Результат рендеринга

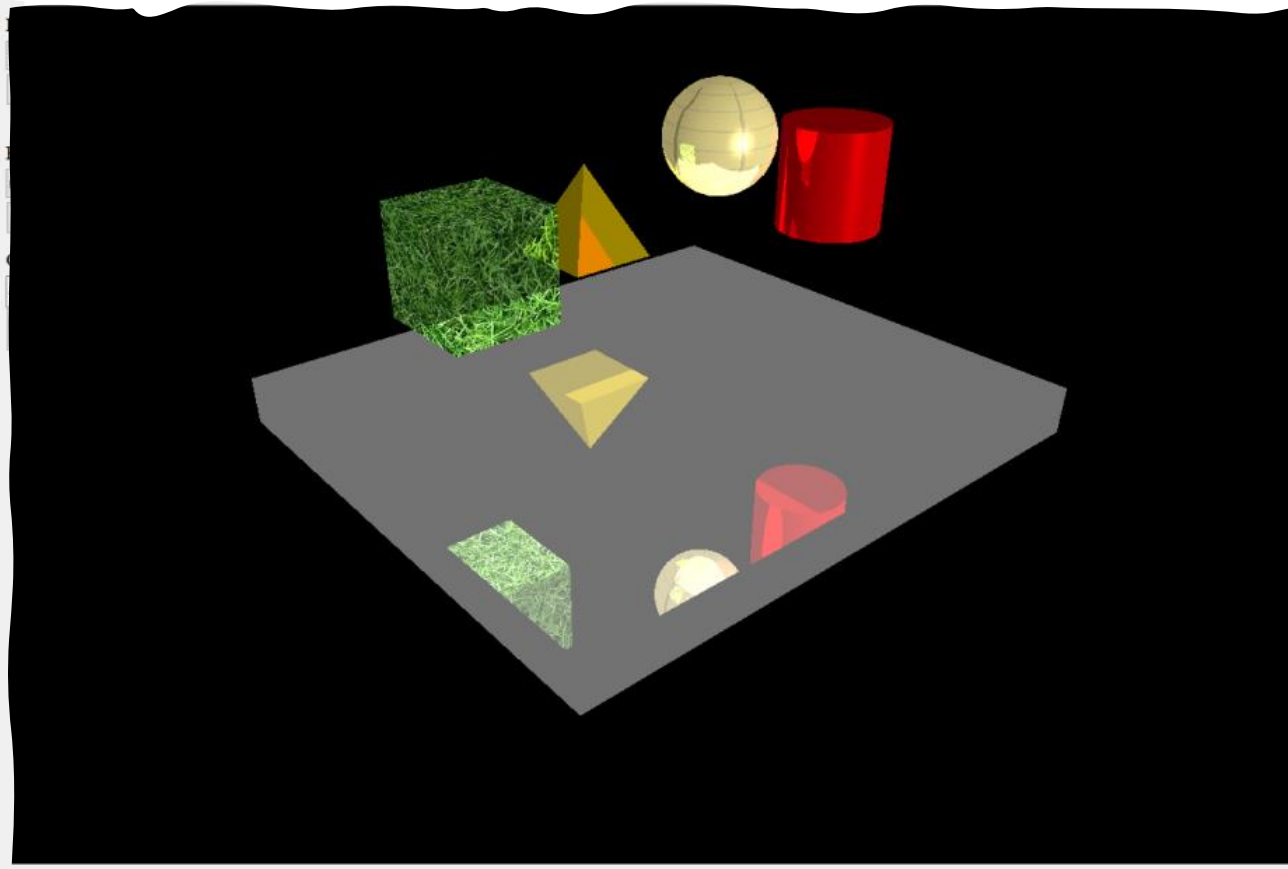




Фролов Евгений ИУ7-55Б

Построение реалистического изображения

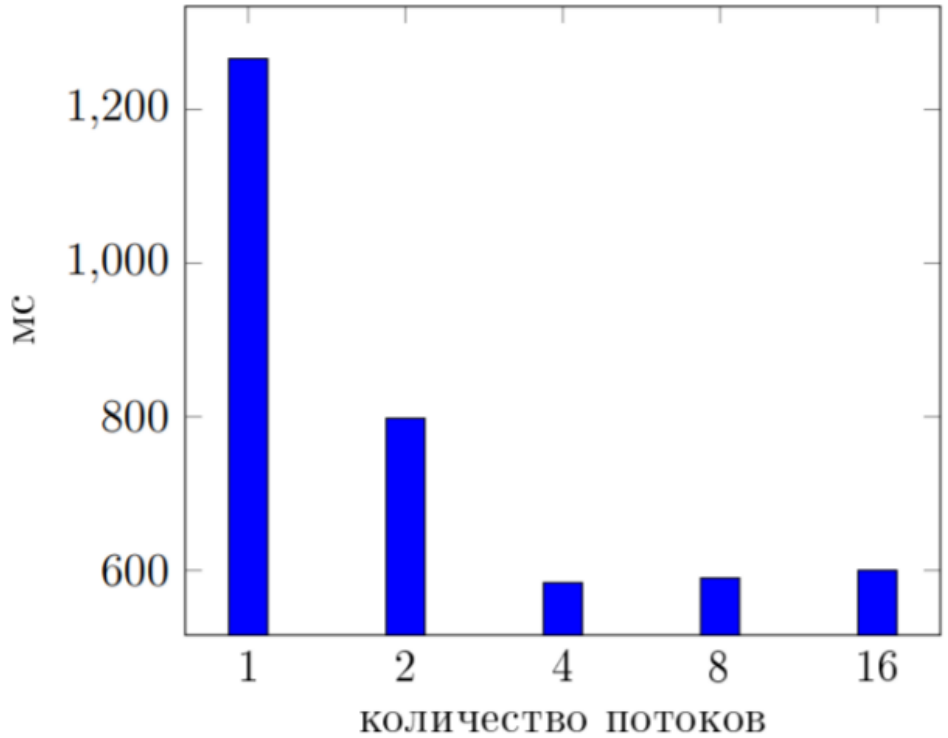




Фролов Евгений ИУ7-55Б

Эксперимент

Для проведения замеров времени была использована формула



$$t = \frac{T_n}{N}$$

где

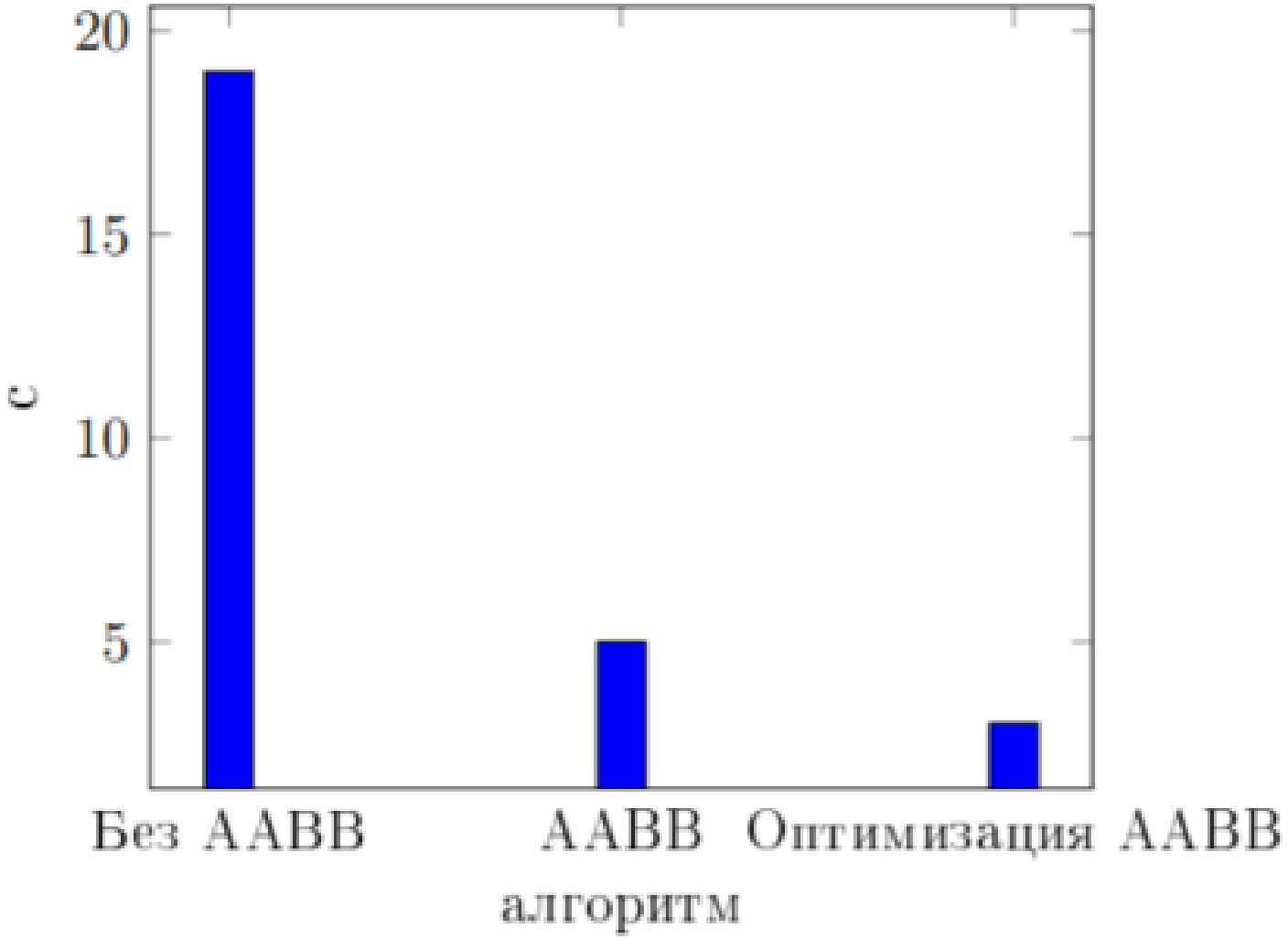
- N - количество потоков;
- t - время выполнения реализации алгоритма;
- T_n - время выполнения N замеров

Начиная с четырех, увеличение числа потоков не дало сильного прироста в скорости

Фролов Евгений ИУ7-55Б

Идея: ограничить каждый объект параллелепипедом, который параллелен координатным осям (AABB axis-aligned bounding box)

Использование AABB существенно ускоряет время работы программы, а применени ряда оптимизаций сокращает время отрисовки в среднем в 1.5 раза.



Фролов Евгений ИУ7-55Б

Спасибо за внимание

Фролов Евгений ИУ7-55Б

Москва 2021