Содержание

Введение

Кубик Рубика является очень простой головоломкой с точки зрения своего устройства. Но стоящая за ней математика проделала довольно долгий путь. Существуют нерешённые задачи, связанные с этой головоломкой.

Сам по себе кубик Рубика, может выглядеть очень по-разному. В рамках данного курсового проекта будет рассмотрена одна из вариаций данной головоломки, а именно — зеркальный кубик Рубика.

Целью курсового проекта является разработка программы построения и визуализации трёхмерной модели зеркального кубика Рубика на основе данных о вращении его граней. Для достижения поставленной цели необходимо решить следующие задачи:

- разработка способа формального представления кубика Рубика;
- анализ существующих алгоритмов удаления невидимых граней;
- реализация описанного алгоритма
- проектирование архитектуры программы
- разработка интерфейса программы
- разработка спроектированного ПО

1 Аналитический раздел

1.1 Кубик Рубика

Кубик Рубика (в честь Эрнё Рубика) — механическая головоломка, представляющая из себя куб, каждая грань которого имеет уникальный цвет и делится малыми кубами на 9 частей (26 в сумме со всех сторон) [1]. Механизм, располагающийся в центре кубика, позволяет вращать каждую грань относительно кубика Рубика. Таким образом, набор случайных поворотов граней приводит к «разборке» кубика рубика. Обратная «сборка» заключается в последовательности поворотов грани так, чтобы каждая грань кубика рубика снова стала одноцветной.

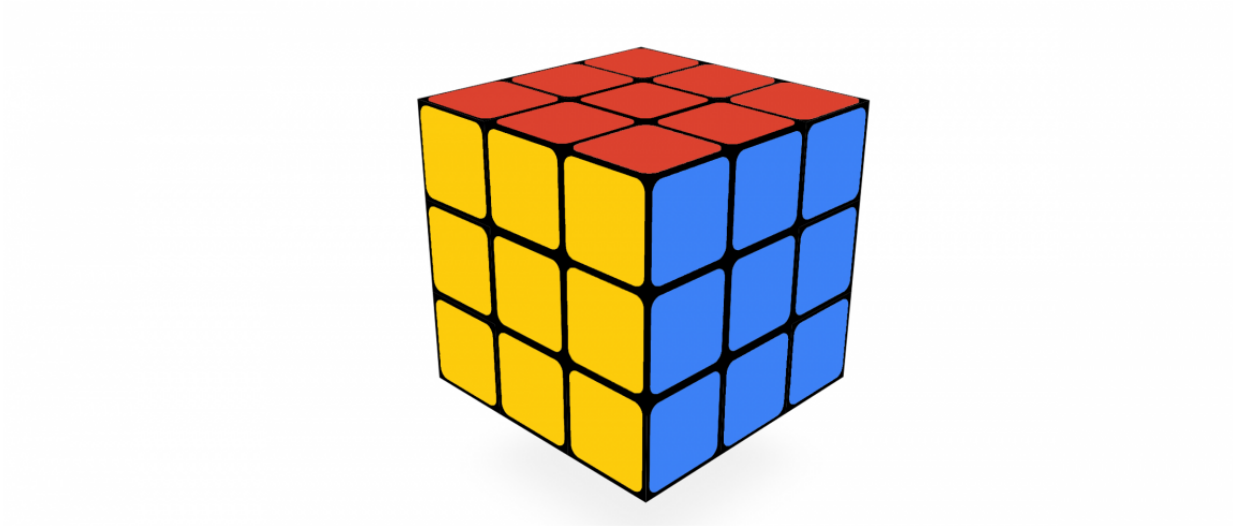

Рисунок 1 – Кубик Рубика

1.1.1 Формализация действий над кубиком Рубика

В дальнейшем, будем использовать обозначения поворотов граней кубика Рубика, введённые «World Cube Association» [2]. Пусть куб находится напротив наблюдателя. Тогда:

- F (front) — фронтальная грань. Находится непосредственно напротив наблюдателя.
- B (back) — задняя грань. Противоположная фронтальной.
- L (left) — левая грань. Находится по левую руку от наблюдателя.

– R (right) — правая грань. Находится по правую руку от наблюдателя.

– U (up) — верхняя грань.

– D (down) — нижняя грань.

Операции поворота грани по часовой стрелке обозначаются именем грани. Так, например, повернуть дважды фронтальную грань, затем повернуть по часовой стрелке верхнюю грань будет обозначаться так: FFU.

Предусмотрено обозначение для поворота грани против часовой стрелки, для этого после символа обозначения грани добавляется апостроф. Так, например, поворот правой грани по часовой стрелке, затем поворот нижней грани против часовой стрелки, затем поворот правой грани по часовой стрелке обозначается так: RD'R.

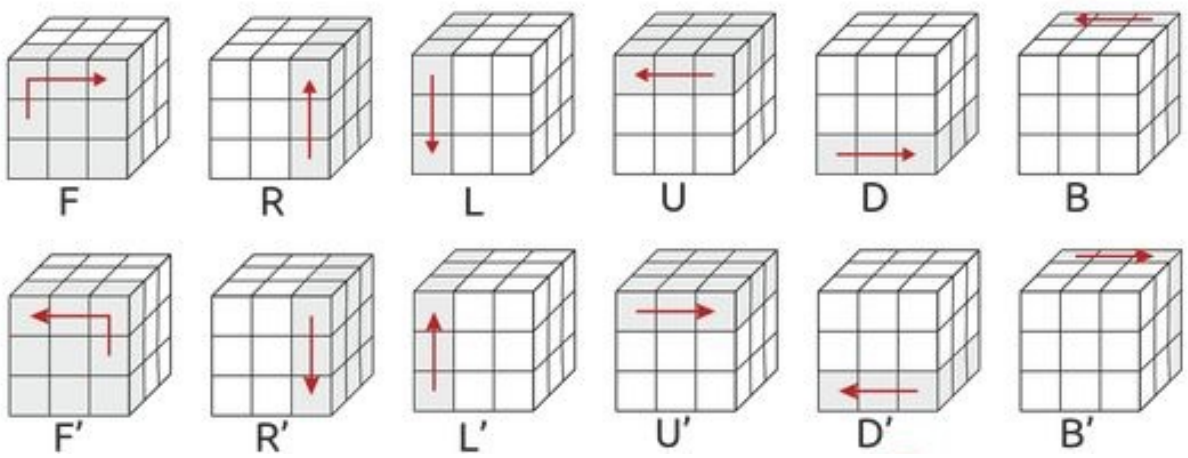

Рисунок 2 – Способы поворота граней кубика Рубика

1.1.2 Сборка кубика Рубика

Для сборки кубика Рубика не обязательно знать всю последовательность действий, применённую к нему, достаточно информации о цветах граней. Для сборки кубика Рубика можно применить алгоритм Джессики Фридрих [3].

Сборка кубика Рубика по методу Джессики Фридрих, делится на четыре этапа:

1) Сборка «креста»

2) Сборка первых двух слоёв (F2L)

3) Ориентация кубиков верхнего слоя (OLL)

4) Расстановка кубиков верхнего слоя (PLL)

Каждый из этих этапов имеет конечное, относительно малое количество вариантов расположения цветов граней. Решение очередного этапа кубика Рубика состоит в определении конкретной ситуации (цветов граней), и применение соответствующей последовательности действий. Например, во время сборки OLL может возникнуть ситуация, проиллюстрированная на рис. 3. Её решением является следующая последовательность действий: R'U'R'FRF'UR

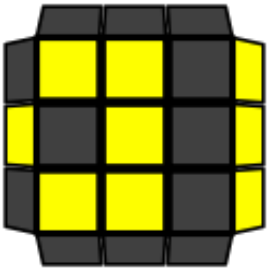

Рисунок 3 – Ситуация, которая может возникнуть во время сборки OLL

1.1.3 Зеркальный кубик Рубика

Цвет — не единственный способ обозначать правильно собранные грани. Одной из модификаций классического кубика Рубика является зеркальный кубик Рубика. Он отличается от классической версии тем, что грани отличаются не по цвету, а по высоте каждого параллелограмма относительно центра кубика. Грань собранного кубика Рубика состоит из граней параллелограммов разного размера, но остаётся в форме квадрата; при разборке — форма меняется. Несмотря на эти различия, зеркальная и классическая версии кубика Рубика имеют общие принципы сборки.

1.1.4 Пропорции зеркального кубика Рубика

Введём пропорции зеркального кубика Рубика (рисунок 5). Пусть:

– наблюдатель смотрит по направлению оси X (нормаль к грани B),

– строго налево от наблюдателя направлена ось Y (нормаль к грани L),

Рисунок 4 – Внешний вид зеркального кубика Рубика в собранном (слева) и в разрбранном (справа) состояниях

– сторо наверх от наблюдателя направлена ось Z (нормаль к грани U)

Тогда вдоль каждой оси последовательно отложим отрезки трёх длин:

– для оси X: 17, 19, 21
– для оси Y: 13, 19, 25
– для оси Z: 9, 19, 29

Плоскости, проведённые перпендикулярно соответствующей оси на границе отрезка, отрезают трёхмерный куб размером 57x57x57, разделённый на 27 параллелограммов.

1.2 Отрисовка сцены

Одной из основных задач разрабатываемого програмного продукта является отрисовка зеркального кубика Рубика. Рассмотрим существующие алгоритмы удаления невидимых граней:

– Алгоритм плавающего горизонта
– Алгоритм Варнока
– Алгоритм Робертса
– Алгоритм Вейлера-Азертона

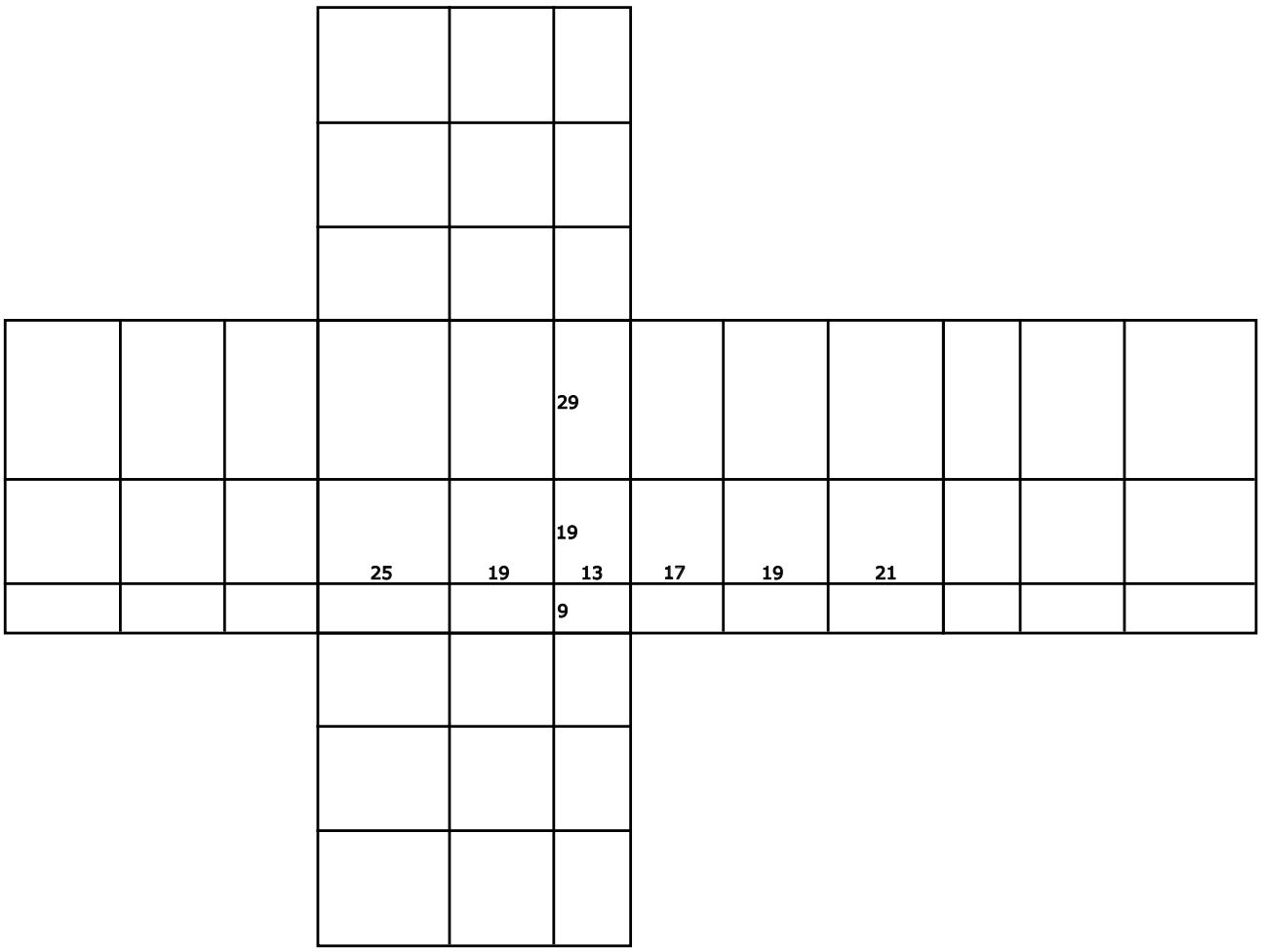

Рисунок 5 – Пропорции зеркального кубика Рубика

– Алгоритм, Z-буфера
– Алгоритм, использующий список приоритетов
– Алгоритмы построчного сканирования

Все вышеперечисленные алгоритмы не могут привести к корректному результату, если среди объектов сцены присутствует зеркальные поверхности. Таким образом, для отрисовки зеркальных поверхностей необходимо использовать алгоритм трассировки лучей [4].

1.2.1 Алгоритм прямой трассировки лучей

Основная идея алгоритма — промоделировать поведение света для получения реалистичного изображения. В реальном мире, свет исходит от источника света, отражается от отражающих поверхностей и рассеивается рассеивающими поверхностями. Некоторые лучи света попадают на сетчатку

глаза, которая посылает сигнал об интенсивности и цвете пришедшего луча. Алгоритм, реализующий такую модель, называется алгоритмом прямой трассировки лучей.

1.2.2 Алгоритм обратной трассировки лучей

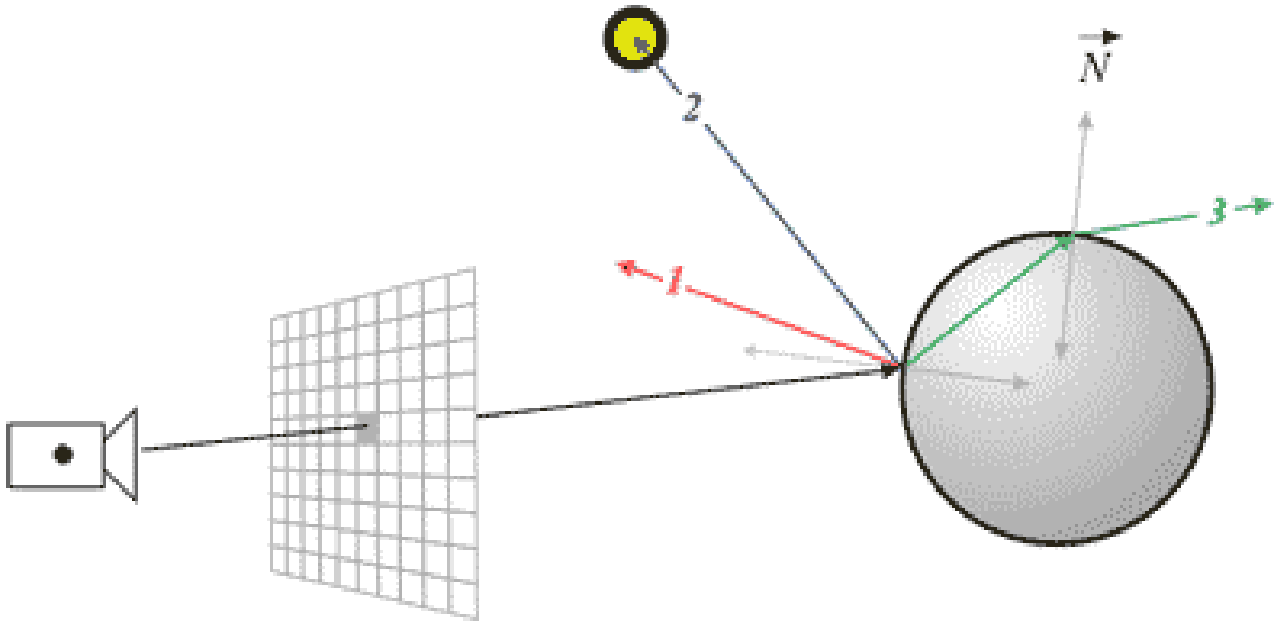

Рисунок 6 – Обратная трассировка лучей

Заметим, что нас интересуют лишь те лучи света, которые попали на сетчатку глаза. На основе этого, при моделировании подобной системы, лучи можно пускать в обратную сторону, то есть начиная от сетчатки. Теперь цвет луча при столкновении будет рассчитываться в зависимости от освещённости очередной точки попадания луча и цвета поверхности, на которую попал луч.

В качестве допущения, будем считать, что все поверхности на сцене являются непрозрачными.

1.2.3 Описание алгоритма обратной трассировки лучей

Алгоритму поступает следующие данные:

- местоположение наблюдателя;
- расстояние до проецирующей плоскости;
- местоположение и геометрия всех объектов на сцене.

На основе этих данных, алгоритм должен окрасить сетку пикселей $n \times m$ так, чтобы при отображении её на экране, можно было распознать реалистичное изображение.

Для каждого пикселя, из камеры пускается луч в соответствующем направлении. При попадании луча в плоскость, порождается дополнительный луч, отражённый. Вектор отражения $\bar{V}_M$, при исходном луче $\bar{V}$, относительно нормали $\bar{n}$, при $|\bar{n}| = 1$ считается по следующей формуле:

$$\bar{V}_M = \bar{V} - 2 \cdot \bar{V} \cdot \bar{n} \cdot \bar{n} \tag{1}$$

Цвет данной точки рассчитывается как смешение цвета поверхности и цвета, рассчитаного при попадании луча $\bar{V}_M$ на поверхность.

С целью упростить задачу поиска пересечения луча и поверхности, будем считать, что каждый объект сцены можно представить в виде набора из конечного числа треугольников.

1.3 Вывод по разделу

В аналитическом разделе был описан кубик Рубика, обозначения граней и их поворотов. Была объяснена разница между классическим кубиком Рубика и его вариацией — зеркальным кубиком Рубика.

Так же, описана задача отрисовки трёхмерной сцены. Обозначена причина, по которой был выбран алгоритм обратной трассировки лучей в качестве способа отрисовки трёхмерной сцены.

Были установлены ограничения по содержанию сцены, а именно:

- Все объекты на сцене являются непрозрачными
- Каждый объект сцены можно представить в виде конечного набора треугольников

2 Конструкторский раздел

Во время разработки программы, должна быть учтена возможность её модификации. Объектно-ориентированный подход обеспечивает гибкость в вопросе изменения уже существующего кода [5].

В программы будет высоконагруженная секция — алгоритм отрисовки сцены. При разработке алгоритма, необходимо как можно больше операций перенести на другие части программы, с целью получения максимального выигрыша в производительности.

2.1 Сцена, объекты сцены

Определим ключевые сущности, тем или иным образом связанные с понятием сцены:

- Сцена. Должна агрегировать все объекты, которые могут быть включены в неё.
- Полигональная модель. Будет содержать множество точек в пространстве и ими задаваемые треугольники.
- Камера. Представляет собой точку в пространстве, имеющую направление и область обзора.
- Кубик Рубика. Агрегирует 27 моделей малых параллелограммов, формирующих зеркальный кубик Рубика.

Добавив несколько вспомогательных классов, получим UML-диаграмму (см. рис. 7). Полученая диаграмма описывает некий домен сцены. Рассмотрим основные математические структуры:

2.2 Математические структуры

Для описания положения объекта в пространстве, необходима следующая информация:

- позиция объекта,
- поворот объекта,
- размер объекта.

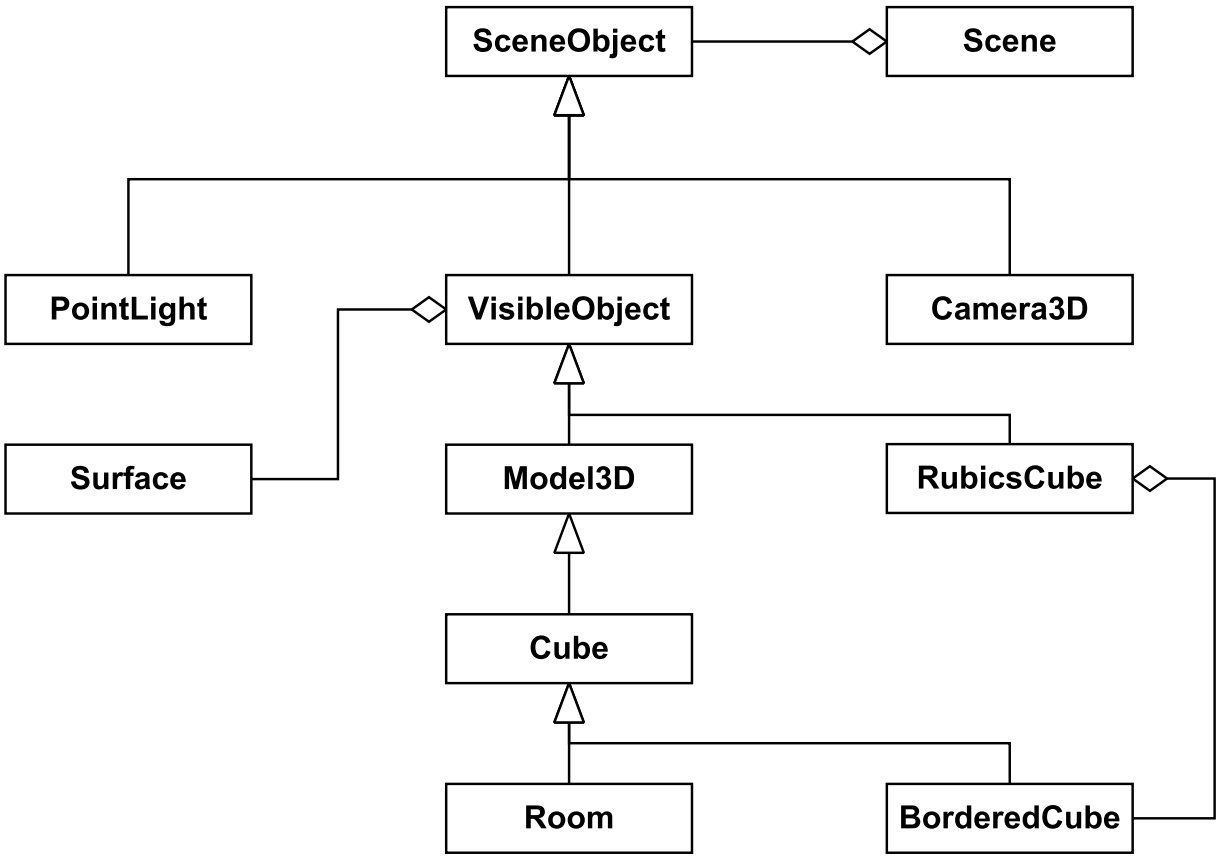

Рисунок 7 – UML диаграмма классов сцены, без описания полей и методов

2.2.1 Трёхмерный вектор

Трёхмерный вектор является тройкой действительных чисел (x, y, z), каждая компонента которого является величиной проекции на соответствующую ось [6].

Позиция объекта может быть описана радиус-вектором, описывающим смещение всех точек объекта относительно начала координат. Таким образом, достаточно прибавить данный радиус-вектор к каждой точке трёхмерной модели, чтобы получить её истинное положение.

Используя точно такую же структуру, будем обозначить размер объекта. Каждый компонент вектора является коэффициентом масштабирования для каждой оси.

2.2.2 Кватернион

Использование тройки углов, описывающих последовательный поворот объекта вдоль всех осей, имеет свои недостатки [7]:

- неоднозначность. Для любой последовательности вращений A, существует как минимум одна другая последовательность, приводящая объект к тому-же положению, что A;
- некорректный результат при последовательном применении двух и более троек углов поворота.

Вместо так называемых Эйлеровых углов, предпочтительнее использовать кватернионы [8].

Кватернион представляет собой кортеж из четырёх чисел (q_0, q_1, q_2, q_3). Для каждого кватерниона q, существует обратный ему $q^{-1} \neq q$:

$$q^{-1} = (q_0, -q_1, -q_2, -q_3), \tag{2}$$

исключением является единичный кватернион $(1, 0, 0, 0)$, так как он обратен самому себе.

Для кватернионов определена операция умножения. Пусть:

$$(t_0, t_1, t_2, t_3) = (r_0, r_1, r_2, r_3) \times (s_0, s_1, s_2, s_3). \tag{3}$$

Тогда:

$$t_0 = (r_0 \cdot s_0 - r_1 \cdot s_1 - r_2 \cdot s_2 - r_3 \cdot s_3) \tag{4a}$$

$$t_1 = (r_0 \cdot s_1 + r_1 \cdot s_0 - r_2 \cdot s_3 + r_3 \cdot s_2) \tag{4b}$$

$$t_2 = (r_0 \cdot s_2 + r_1 \cdot s_3 + r_2 \cdot s_0 - r_3 \cdot s_1) \tag{4c}$$

$$t_3 = (r_0 \cdot s_3 - r_1 \cdot s_2 + r_2 \cdot s_1 + r_3 \cdot s_0) \tag{4d}$$

С помощью кватерниона, можно повернуть точку в пространстве, заданную тремя координатами. Для этого нужно задать кватернион p, как:

$$p = (p_0, p_1, p_2, p_3) = (0, x, y, z), \tag{5}$$

где (x, y, z) - точка в пространстве. Получим p' следующим образом:

$$p' = q^{-1} \cdot p \cdot q, \tag{6}$$

где q — кватернион, описывающий вращение. Искомая точка является тройкой (p'_1, p'_2, p'_3).

2.2.3 Цвет

Структура цвета необходима для описания поверхностей объектов. Представляет собой тройку чисел (r, g, b). Значение компонент определяет интенсивность цветов: красного, зелёного и синего соответственно. Сочетания этих трёх цветов разной интенсивности позволяют получить любой другой цвет.

Определим операцию умножения цвета на скаляр:

$$C \cdot k = (r, g, b) \cdot k = (r \cdot k, g \cdot k, b \cdot k), \tag{7}$$

где C — исходный цвет, k — скаляр.

Определим операцию сложения цветов:

$$C_1 + C_2 = (r_1, g_1, b_1) + (r_2, g_2, b_2) = (r_1 + r_2, g_1 + g_2, b_1 + b_2), \tag{8}$$

где C_1 и C_1 — исходные цвета

2.2.4 Список математических структур

В конечном итоге, выделяем следующие структуры:

- Vector3D — трёхмерный вектор
- Quaternion — кватернион
- Transform — положение объекта в пространстве
- Color — цвет

2.3 Алгоритм обратной трассировки лучей

Основная идея алгоритма трассировки лучей заключается в том, чтобы проследить путь, который проходит свет от источника до наблюдателя. Делается этот процесс в обратном направлении, то есть из точки наблюдателя

выпускаются лучи, которые, несколько раз отражаются от встреченных поверхностей. Данный процесс продолжается до тех пор, пока либо количество дополнительно выпущеных лучей не превысит определённое значение, либо очередная поверхность не окажется неотражающей.

2.3.1 Алгоритм определения цвета пикселя

Алгоритм, определяющий цвет пикселя, принимает на вход два трёхмерных вектора: местонахождение наблюдателя (rs) и направление бросания луча (rd). Схема алгоритма представлена на рис. 8

Переменная hits[] является массивом из структур, содержащих информацию о попадании. Они должны хранить следующие данные:

- было ли попадание (hit) — логическое;
- точке попадания луча (point) — трёхмерный вектор;
- направление отскока (bounce) — трёхмерный вектор;
- поверхность, с которой произошло пересечение (surface) — поверхность;

Структура поверхности содержит в себе информацию о треугольнике в пространстве и его свойствах, а именно:

- три точки, задающие треугольник (points) — тройка трёхмерных векторов;
- коэффициент рассеивания (diffuse) — вещественное. Является обратным к коэффициенту отражения;
- цвет поверхности (color) – цвет;
- нормаль к поверхности (normal) — трёхмерный вектор;
- объект, частью которого является поверхность (owner) — объект сцены;

Функция Blend реализует смешение двух цветов по формуле:

$$C = C_1 \cdot k + C_2 \cdot (1 - k), \tag{9}$$

где C, C_1, C_2 — цвета, $k \in [0, 1]$ — скаляр.

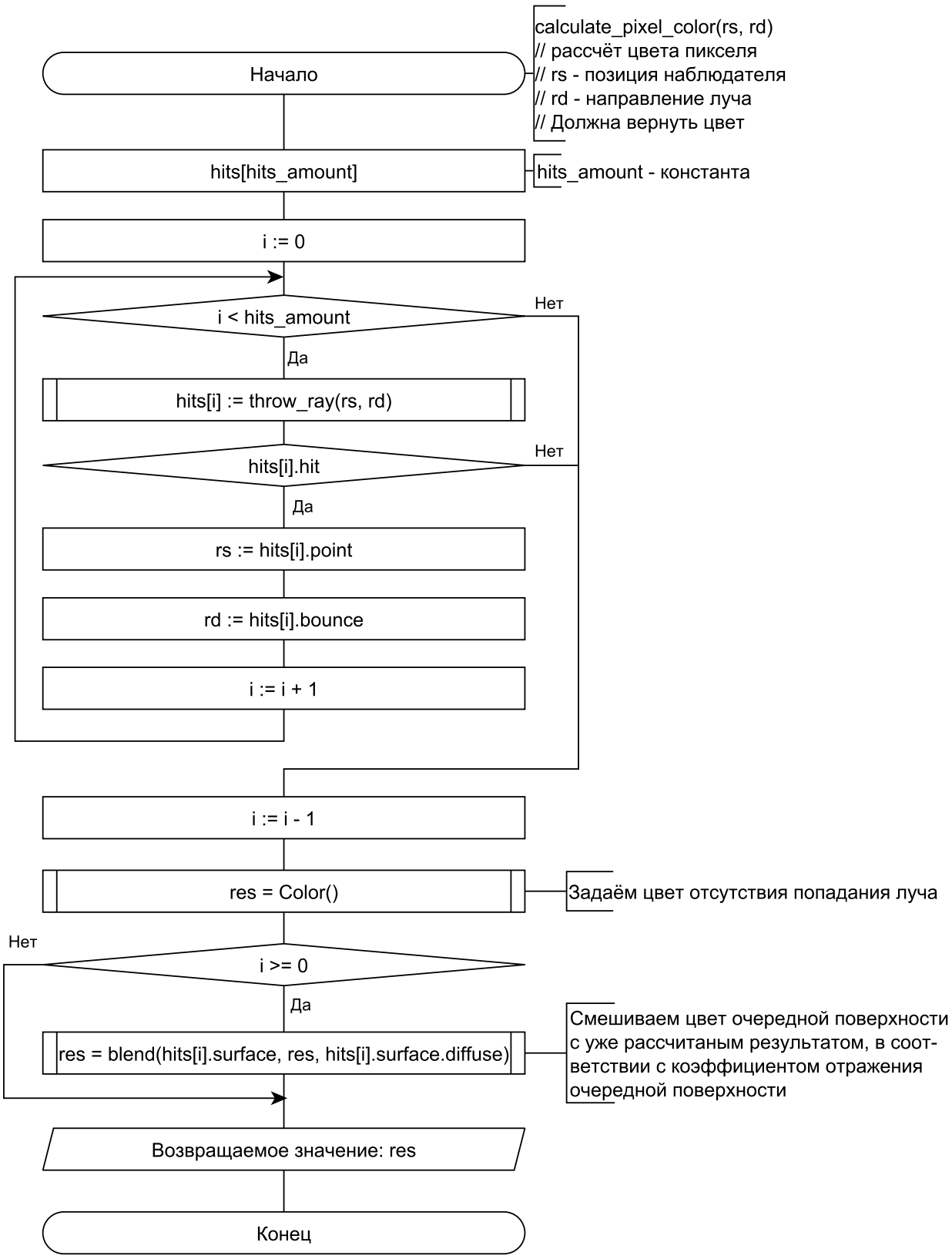

Рисунок 8 – Схема алгоритма определения цвета пикселя

2.3.2 Алгоритм бросания луча

Алгоритм бросания луча принимает на вход два трёхмерных вектора: местонахождение наблюдателя (rs) и направление бросания луча (rd). Схема алгоритма представлена на рис. 9.

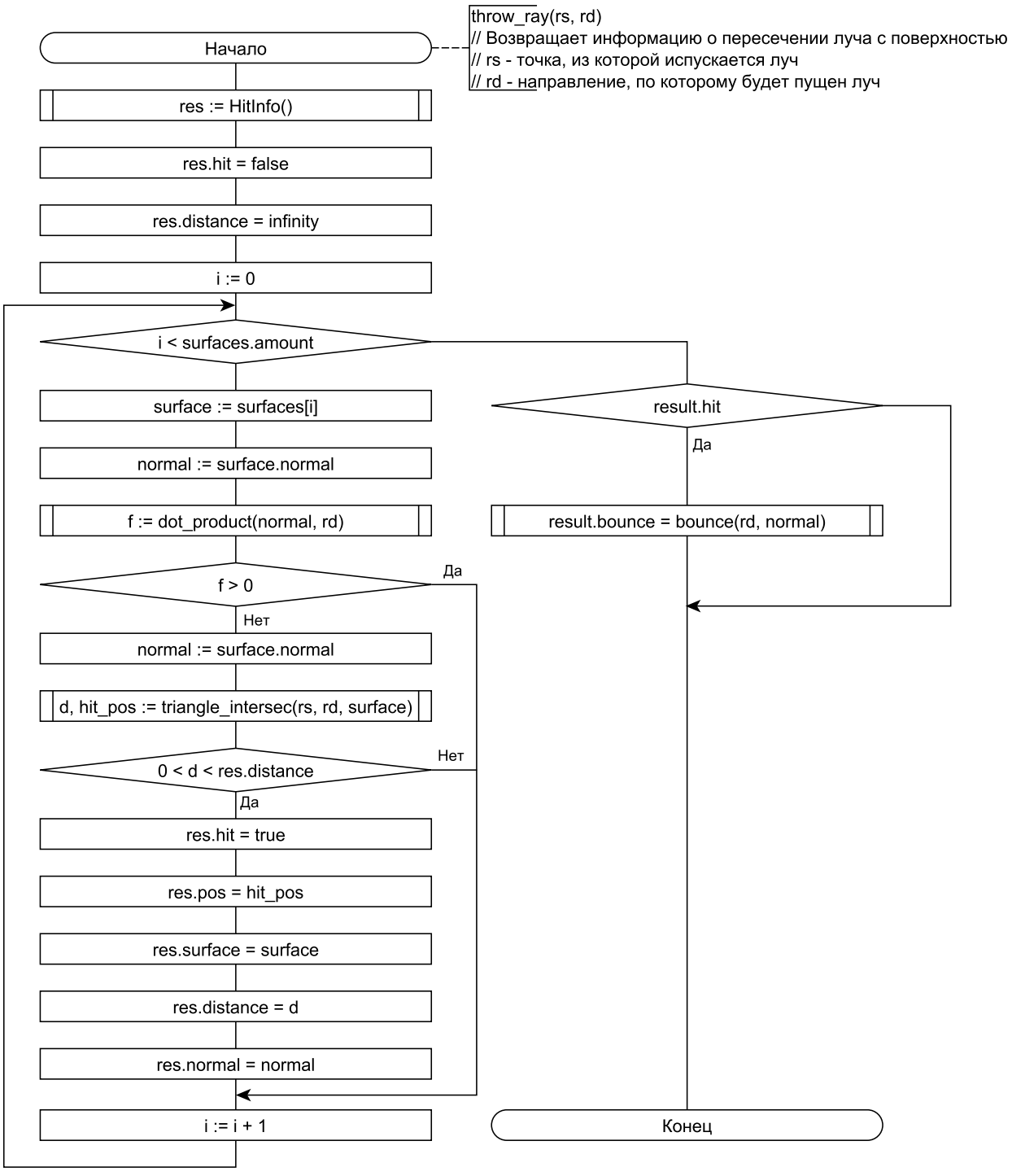

Рисунок 9 – Схема алгоритма бросания луча

Функция bounce выполняет отражение вектора $\bar{V}$ относительно нормали $\bar{n}$ в соответствии с формулой 1

2.3.3 Алгоритм поиска пересечения луча с треугольником

В качестве алгоритма поиска пересечения луча с треугольником используется алгоритм Моллера-Трумбора[9]. Схема алгоритма представлена на рис. 10

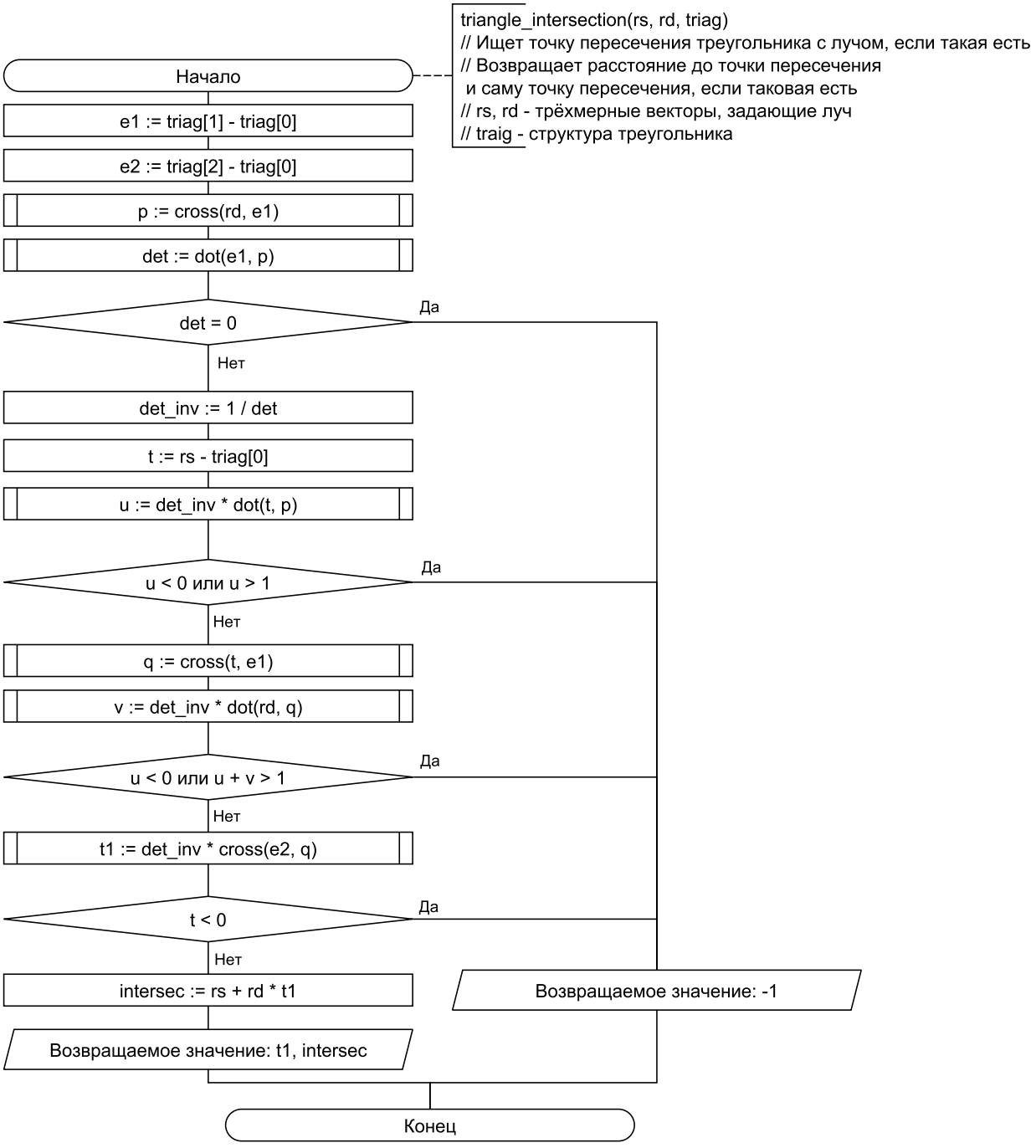

Рисунок 10 – Схема алгоритма поиска пересечения луча с треугольником

Функция cross выполняет векторное произведение двух векторов по формуле:

$$(x_1, y_1, z_1) \times (x_2, y_2, z_2) = (y_1 \cdot z_2 - z_1 \cdot y_2,\ z_1 \cdot x_2 - x_1 \cdot z_2,\ x_1 \cdot y_2 - y_1 \cdot x_2) \quad (10)$$

Функция dot - вычисляет скалярное произведение двух векторов по формуле:

$$(x_1, y_1, z_1) \cdot (x_2, y_2, z_2) = x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2 \quad (11)$$

Идея алгоритма заключается в том, чтобы отыскать точку пересечения луча с треугольником в барицентрических координатах. Вместо того, чтобы искать точку пересечения луча и плоскости, а потом определять лежит ли точка внутри треугольника, получив барицентрические координаты по их значению можно сразу понять, лежит ли точка внутри треугольника или нет

2.4 Зеркальный кубик Рубика

2.4.1 Создание зеркального кубика Рубика

Создавать зеркальный кубик Рубика необходимо в соответствии с пропорциями, указанными на рис. 5. Схема алгоритма создания параллелограммов представлена на рис 11.

Данная схема не иллюстрирует использование пропорций, и отражает лишь порядок в котором должны создаваться параллелограммы.

2.4.2 Вращение зеркального кубика Рубика

Вращение грани производится в три этапа (порядок произвольный):

1) вращение моделей,
2) смещение граней в массиве,
3) смещение углов в массиве.

Под углами и гранями подразумеваются угловые и граневые параллелограммы кубика Рубика.

Смещение граней и углов в массиве выполняется как циклический сдвиг его элементов с соответствующей четвёркой индексов. В таблице 1 поставлены в соответствие какой циклический сдвиг нужно сделать при повороте со-

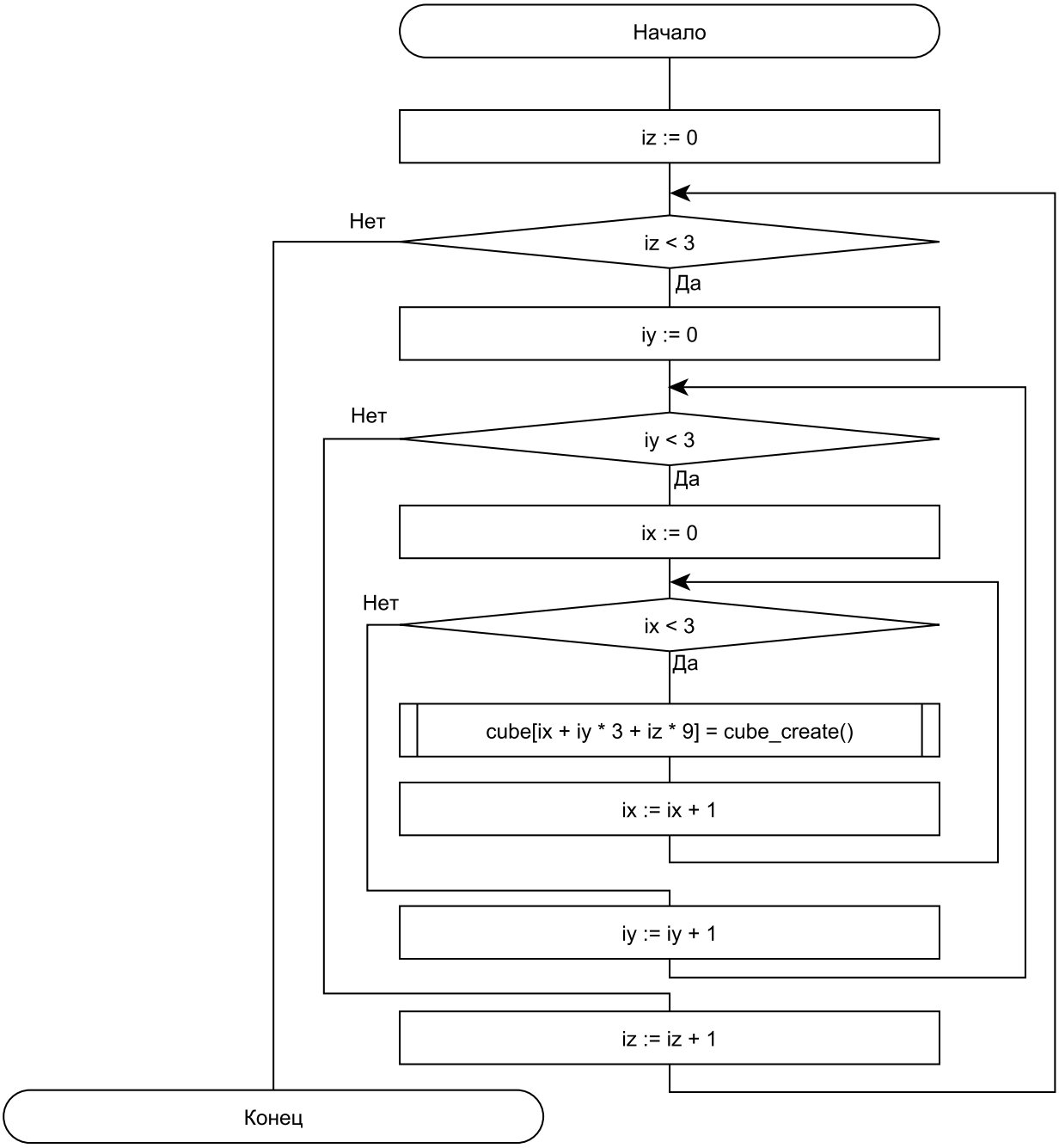

Рисунок 11 – Схема алгоритма создания кубика рубика

ответствующей грани. Если вращение выполняется в противоположную сторону, достаточно развернуть последовательность индексов в другую сторону.

2.5 Вывод по разделу

В конструкторском разделе была описана архитектура создаваемой программы; выделены структуры, используемые в программе; ключевые алгоритмы, в числе которых отрисовка сцены методом трассировки лучей, поиск

Таблица 1 – грани кубика Рубика и соответствующие им параллелограммы

Сторона	Индексы углов	Индексы граней
F	0, 6, 24, 18	3, 15, 21, 9
B	8, 2, 20, 26	5, 11, 23, 17
L	6, 8, 26, 24	7, 17, 25, 15
R	2, 0, 18, 20	1, 9, 19, 11
U	18, 24, 26, 20	21, 25, 23, 19
D	0, 2, 8, 6	1, 5, 7, 3

точки пересечения луча и треугольника, вращение кубика Рубика.

3 Технологический раздел

3.1 Выбор используемых технологий

3.1.1 Выбор ЯП

Основные два требования, которые предъявляются к языку программирования в рамках поставленной задачи:

1) высокая производительность;
2) поддержка ООП.

Всем вышеперечисленым требованиям отвечает язык «C++» стандарта 20-го год [10].

3.1.2 Выбор фреймворка

Одно из ограничений курсовой работы — запрет на использование уже существующих решений для отрисовки трёхмерной сцены. Так как алгоритм будет разрабатываться самостоятельно, производительность не является основным критерием для выбора фреймворка. От него требуется возможность отрисовать на экране сетку пикселей. Так же фреймворк должен предоставлять средства разработки графического интерфейса.

Всем вышеперечисленым требованиям удовлетворяет пакет программ «Qt», он и будет использоваться для создания интерфейса и отрисовки сетки пикселей на экране [11].

3.1.3 Выбор среды программирования

Qt creator — кроссплатформенная среда разработки, разработана специально для работы с фреймворком Qt. Несмотря на это, в качестве среды программирования (далее IDE) был использован Microsoft Visual Studio 2020. Таков выбор объясняется тем, что данная IDE предоставляет [12]:

– систему решений, позволяющих разделять программуна множество связанных между собой проектов
– средства профилирования, графически выделяющие строчки кода, занимающие большую долю процессорного времени

– средства упрощённой работы с git

Таким образом, в качестве IDE используется Microsoft Visual Studio 2020

3.2 Интерфейс пользователя

Интерфейс программы представлен на рисунке 12.

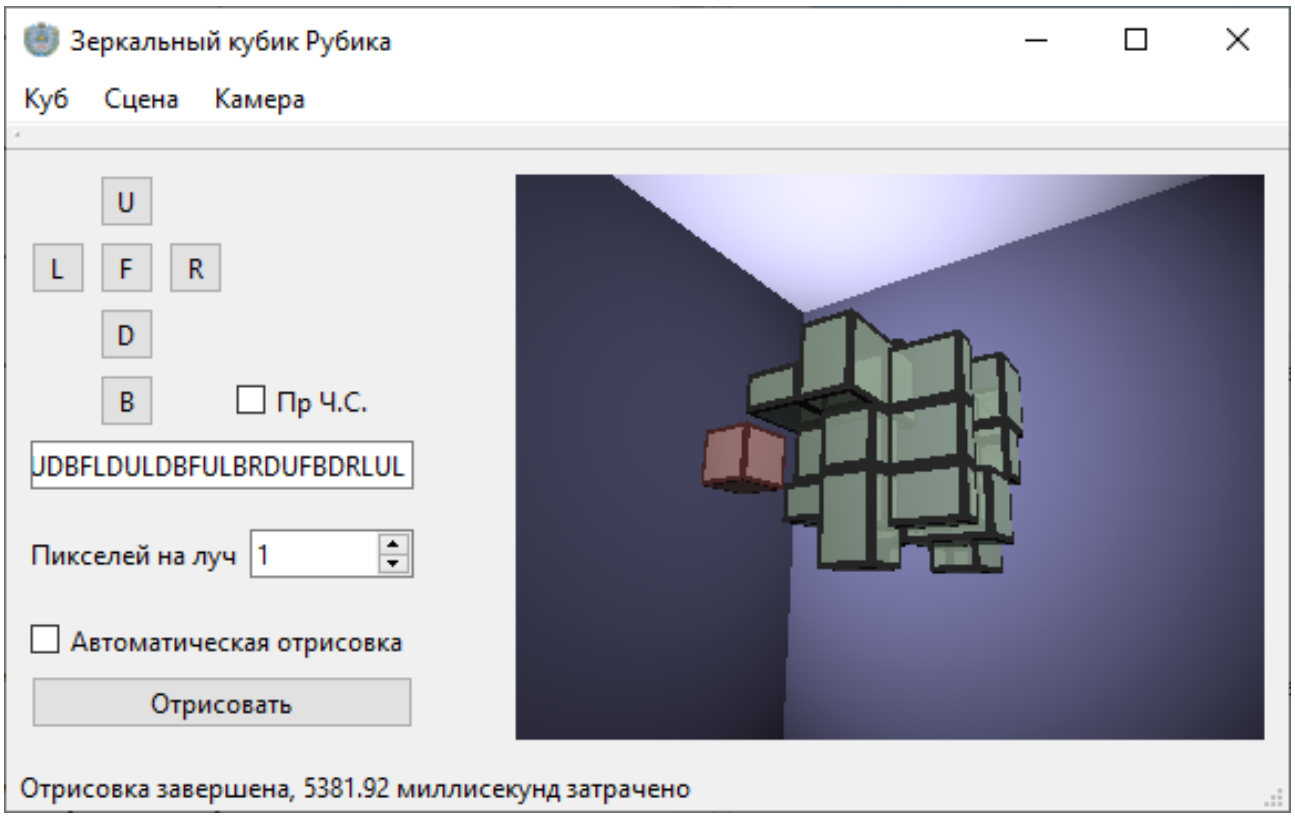

Рисунок 12 – Интерфейс программы

Справа находится окно просмотра, в котором отображается сцена после отрисовки. Основные элементы управления программой находятся слева.

3.2.1 Кнопки, флаги, поля

Кнопки U, L, F, R, D, B выполняют соответствующий поворот зеркального кубика Рубика, по отношению к изначальной позиции камеры. Флаг Пр. Ч. С. переключает кнопки U, L, F, R, D, B на U’, L’, F’, R’, D’, B’, функция которых меняется в соответствии с названием.

Список поворотов — поле только для чтения, в которое записывается каждый совершённый поворот.

Пикселей на луч — числовое поле, позволяющее установить габариты квадрата из пикселей, для которого будет пущен луч. Например, при значении 3, на каждый луч будет приходиться 3×3 пикселей.

Автоматическая отрисовка - флаг, при включении которого любое действие меняющее сцену или область отрисовки будет инициировать автоматическую перерисовку.

Отрисовать — кнопка, инициирующая рендер сцены.

3.2.2 Меню

Меню Куб предоставляет действия:

1) Отменить поворот. Переводит куб в состояние, в котором он был до последней операции поворота грани
2) В исходное состояние. Переводит куб с состояние, в котором он был при запуске программы

Меню Сцена предоставляет действие Простая отрисовка. Переключает программу между режимами отрисовки сцены с использованием алгоритмов трассировки лучей и проецирования каркасной модели без удаления невидимых граней.

Меню Камера предоставляею опцию В исходное состояние, которая возвращает камеру на позицию, где она была в момент запуска программы.

Кроме того, в меню камера содержится подменю Повернуть... в котором перечислены 4 действия, смещающие камеру:

1) Влево
2) Вправо
3) Вверх
4) Вниз

3.3 Тестирование

Для тестирования методов, работающих с математическими структурами, используется система автоматического тестирования Microsoft Visual Studio CPP Unit Test Framework. На листинге 1 представлен пример тести-

рующего класса.

Листинг 1 – Класс, реализующий тестирование методов для работы с трёхмерным вектором

```
#include "pch.h"
#include "CppUnitTest.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace unittest {
TEST_CLASS(TestVector3D) {
public:
  TEST_METHOD(DotProduct) {
    Assert::AreEqual(Vector3D(4,-6,2)*Vector3D(2,3,3),-4.);
    Assert::AreEqual(Vector3D(2,3,3)*Vector3D(4,-6,2),-4.);
    Assert::AreEqual(
      Vector3D::dot_product(Vector3D(4,-6,2),Vector3D(0,0,0)),0.);
    Assert::AreEqual(
      Vector3D::dot_product(Vector3D(2,3,3),Vector3D(4,-6,2)),0.);
  }
  TEST_METHOD(CrossProduct) {
    auto a=Vector3D(4,-6,2);
    auto b=Vector3D(2,3,3);
    Vector3D cross=Vector3D::cross_product(a,b);
    Assert::AreEqual(Vector3D(-24,-8,24).to_string(),cross.to_string());

    b=Vector3D(0,0,0);
    cross=Vector3D::cross_product(a,b);
    for(int i=0; i<3; i++)
      Assert::AreEqual(0.,cross[i]);
  }
};
}
```

3.3.1 Тестовые данные

Далее будут приведены данные, использованные для тестирования функций, работающих с основными структурами.

1) real Angle::to_radians(real) — функция, принимающая на вход значение угла в градусах, возвращающая значение угла в радианах. Тестовые

Таблица 2 – Тестовые данные для функций real Angle::to_radians(real) и real Angle::to_degrees(real)

Угол в градусах	Угол в радианах
180	π
90	$\frac{\pi}{2}$
45	$\frac{\pi}{4}$
-90	$-\frac{\pi}{2}$
0	0

Таблица 3 – Тестовые данные для функции real Angle::optimize_radians(real)

Вход	Выход
$\frac{5\pi}{2}$	$\frac{\pi}{2}$
$-\frac{\pi}{2}$	$\frac{3\pi}{2}$
$\frac{\pi}{2}$	$\frac{\pi}{2}$
0	0
2π	0

данные приведены в таблице 2.

2) real Angle::to_degrees(real) — функция, принимающая на вход значение угла в радианах, возвращающая значение угла в градусах. Тестовые данные приведены в таблице 2.
3) real Angle::optimize_degrees(real) — функция, приводящий произвольный угол $\alpha \in \mathbb{R}$, заданный в градусах к углу $\beta \in [0; 360)$, заданному в градусах. Тестовые данные для этой функции приведены в таблице 3
4) real Angle::optimize_degrees(real) — функция, приводящий произвольный угол $\alpha \in \mathbb{R}$, заданный в радианах к углу $\beta \in [0; 2\pi)$, заданному в радианах. Тестовые данные для этой функции приведены в таблице 4
5) Color Color::blend(Color, Color, real) — функция, выполняющая смеше-

Таблица 4 – Тестовые данные для функции real Angle::optimize_degrees(real)

Вход	Выход
540	180
-90	270
90	90
0	0
360	0

Таблица 5 – Тестовые данные для функции Color Color::blend(Color, Color, real)

C_1	C_2	k	C
(250, 120, 250)	(100, 100, 100)	0.5	(175, 110, 175)
(250, 120, 250)	(100, 100, 100)	0.25	(137, 105, 137)

ние цветов в соответствии с формулой 9. Тестовые данные для этой функции приведены в таблице 5

Здесь TEST_CLASS — группа тестов, соответствующая классу программы; TEST_METHOD - метод, в котором прописаны тесты для соответствующего тестируемого метода класса.

Visual Studio предоставляет инструмент для прогонки тестов, в виде окна Test Explorer. Его внешний вид представлен на рис 13.

3.3.2 Окно просмотра

Оценка правильности полученного изображения была проведена с помощью визуального анализа. В качестве вспомогательного инструмента, была реализована функция отображения сцены в каркасном виде без удаления граней. Внешний вид окна просмотра при работе этой функции представлен на рис. 14

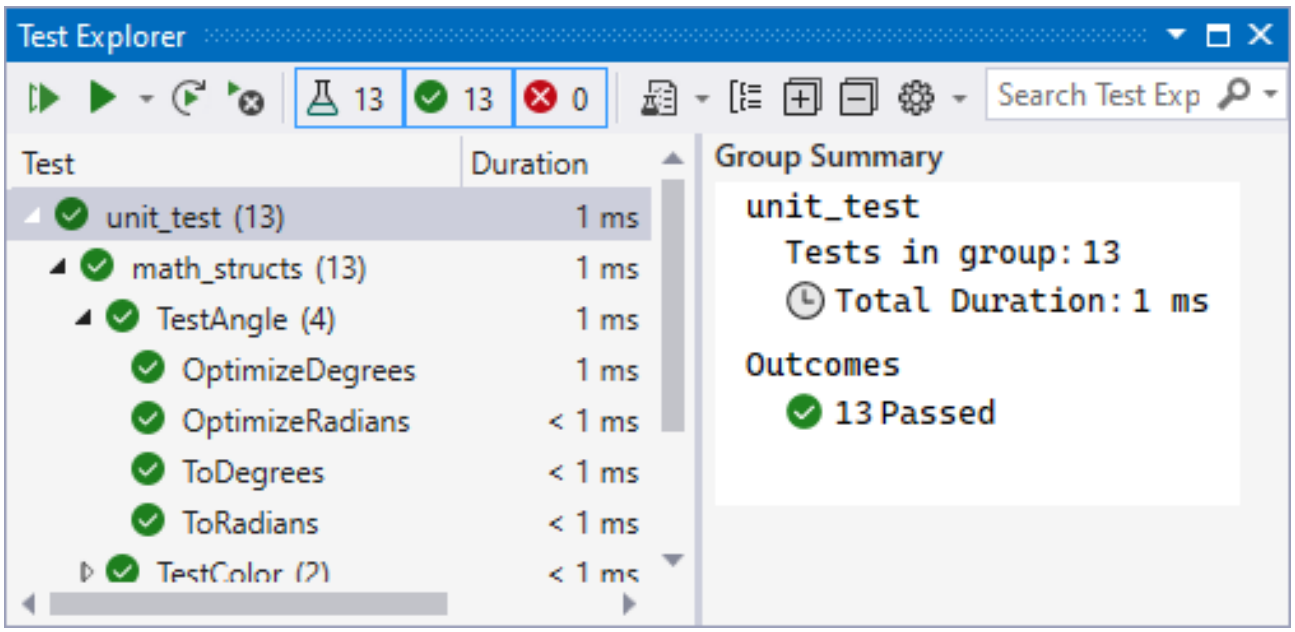

Рисунок 13 – Окно Test Explorer

3.4 Вывод по разделу

В технологическом разделе представлены все использованные программные средства с обоснованием их выбора, а именно:

– C++ стандарта 20-го года;

– Qt фреймворк;

– Microsoft Visual Studio 2020;

Представлены методы тестирования программы, тестовые данные и описан интерфейс пользователя.

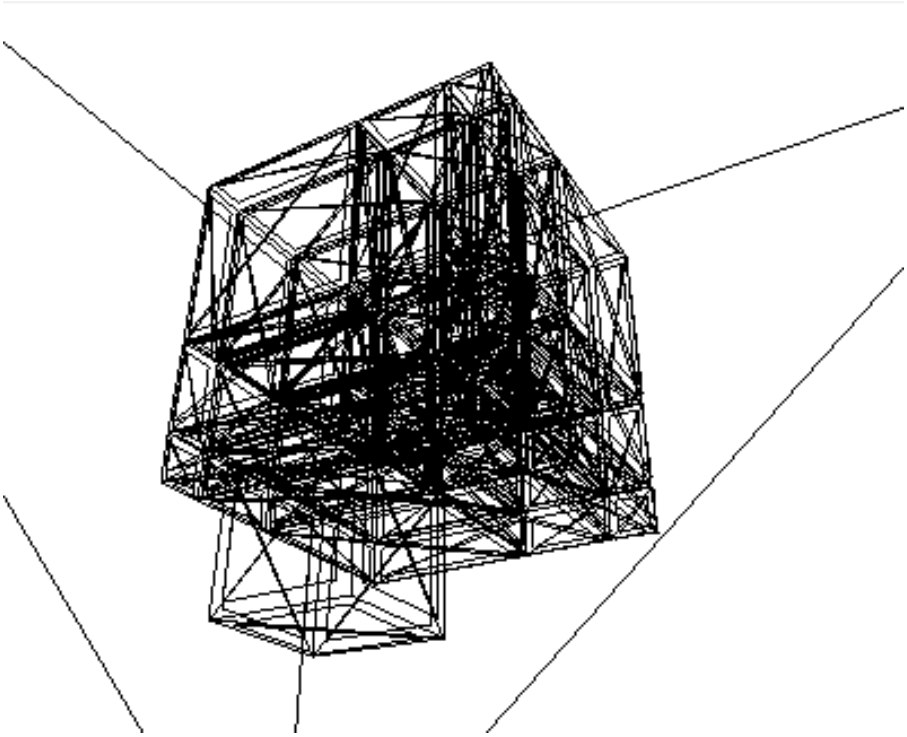

Рисунок 14 – Окно просмотра в режиме простой отрисовки

Заключение

Цель достигнута. В рамках данной курсовой работы были проанализированы существующие алгоритмы отрисовки сцены, среди которых был выбран алгоритм обратной трассировки лучей, из-за возможности отрисовки зеркальных поверхностей.

Была составлена архитектура программы, а именно:

- представлена сокращённая UML-диаграмма классов сцены (рисунок 7),
- выделены основные используемые программой структуры данных,
- формализованы алгоритмы, необходимые для отрисовки сцены.

Была реализована программа, причём:

- был определён набор инструментов для реализации программы,
- представлен внешний вид интерфейса (рисунок 12),
- составлены тесты для функций, использующих математические структуры данных.

Источники

1. Зайцев, А. А. Механика кубика Рубика / А. А. Зайцев, М. Л. Сайфуллина. — Текст : непосредственный // Юный ученый. — 2019. — № 4 (24). — С. 56-61. — URL: https://moluch.ru/young/archive/24/1422/ (дата обращения: 28.09.2022).

2. WCA Regulations. 2022. С. 9–10. Режим доступа: https://www.worldcubeassociation.org/regulations/wca-regulations-and-guidelines.pdf (дата обращения: 28.09.2022).

3. J. Fridrich. My system for solving Rubik's cube. Режим доступа: http://www.ws.binghamton.edu/fridrich/system.html (дата обращения: 28.09.2022).

4. Никулин, Е. А. Компьютерная графика. Модели и алгоритмы : учебное пособие / Е. А. Никулин. — 2-е изд., стер. — Санкт-Петербург : Лань, 2022. — ISBN 978-5-8114-2505-1. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/213038 (дата обращения: 19.09.2022). — Режим доступа: для авториз. пользователей. — С. 217.

5. Барков, И. А. Объектно-ориентированное программирование : учебник / И. А. Барков. — Санкт-Петербург : Лань, 2022. — ISBN 978-5-8114-3586-9. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/206699 (дата обращения: 28.09.2022). — Режим доступа: для авториз. пользователей. — С. 155.

6. Канатников, А. Н. Аналитическая геометрия : учебник / А. Н. Канатников, А. П. Крищенко ; под редакцией В. С. Зарубина, А. П. Крищенко. — 9-е изд. — Москва : МГТУ им. Баумана, 2019. — ISBN 978-5-7038-4904-0. — Текст : электронный // Лань : электронно-библиотечная система. —

URL: https://e.lanbook.com/book/205082 (дата обращения: 28.09.2022). — Режим доступа: для авториз. пользователей. — С. 13-41.

7. D. Rose. Rotations in Three-Dimensions: Euler Angles and Rotation Matrices. 2015. Режим доступа: https://danceswithcode.net/engineeringnotes/rotations_in_3d/rotations_in_3d_part1.html (дата обращения: 28.09.2022).

8. D. Rose. Rotation Quaternions, and How to Use Them. 2015. Режим доступа: https://danceswithcode.net/engineeringnotes/quaternions/quaternions.html (дата обращения: 28.09.2022).

9. Möller T. Trumbore B. Fast, Minumum Storage Ray/Triangle Intersection. Режим доступа: https://cadxfem.org/inf/Fast MinimumStorage RayTriangle Intersection.pdf (дата обращения: 28.09.2022).

10. О'Двайр, А. Осваиваем C++17 STL / А. О'Двайр. — Москва : ДМК Пресс, 2018. — ISBN 978-5-97060-663-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/116126 (дата обращения: 28.09.2022). — Режим доступа: для авториз. пользователей. — С. 17.

11. Qt Modules and Tools for Designers and Developers. Режим доступа: https://www.qt.io/product/features (дата обращения: 28.09.2022).

12. What can you do with Visual Studio? Режим доступа: https://visualstudio.microsoft.com/vs/features/ (дата обращения: 28.09.2022).