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 программного обеспечения, визуализирующего «Кубик Рубика», «Пирамидку Мефферта»

Студент ИУ7-52Б _____ **И.С.Климов**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ **А.В.Шикуть**
(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 ИУ7
(Индекс)
_____ И.В. Рудаков
(И.О.Фамилия)
« _____ » _____ 2021 г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине Компьютерная графика

Студент группы ИУ7-52Б

Климов Илья Сергеевич
(Фамилия, имя, отчество)

Тема курсовой работы Разработка программного обеспечения, визуализирующего «Кубик Рубика», «Пирамидку Мефферта»

Направленность КР (учебная, исследовательская, практическая, производственная, др.)

учебная

Источник тематики (кафедра, предприятие, НИР) кафедра

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

Задание разработать программное обеспечение, визуализирующее «Кубик Рубика», «Пирамидку Мефферта». Предоставить возможность поворота граней, вращения каждой головоломки вокруг осей. Создать интерфейс, позволяющий пользователю выбирать одну из двух головоломок, задавать параметры (ее размер, цветовую гамму), начальное положение, управлять объектом, масштабировать, задавать до двух источников света.

Оформление курсовой работы:
2.1. Расчетно-пояснительная записка на 25-30 листах формата А4.
Расчетно-пояснительная записка должна содержать постановку введение, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение, список литературы, приложения.

2.2. Перечень графического материала (плакаты, схемы, чертежи 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 курсовой работы _____ А.В. Шикуть
(Подпись, дата) (И.О.Фамилия)

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

Содержание

Введение

В 1974 году венгерский скульптор Эрнё Рубик изобрел головоломку, которая спустя время стала одной из самых популярных в мире. Наверное, нет такого человека, который бы не слышал про неё. Головоломка носит название в честь её изобретателя – кубик Рубика и представляет собой пластмассовый куб с подвижными деталями. На рисунке 1 изображен кубик в собранном состоянии.

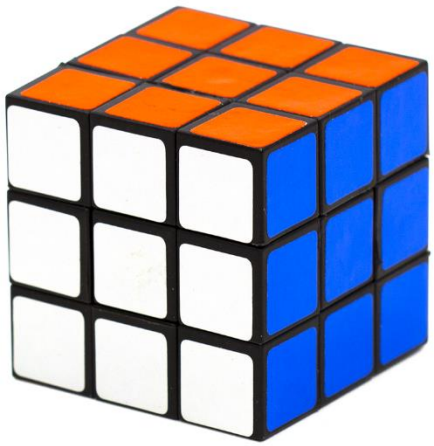

Рисунок 1 – Кубик Рубика в собранном состоянии

Изначально размерность составляла только 3x3x3 – в таком случае кубик состоит из крестовины с 6-ю центральными элементами, 8 угловых и 12 реберных элементов. В дальнейшем начали появляться различные модификации, включающие в себя не только другую размерность, но и разную форму с отличающимся внутренним строением. На рисунке 2 представлена лишь малая часть всего разнообразия существующих головоломок.

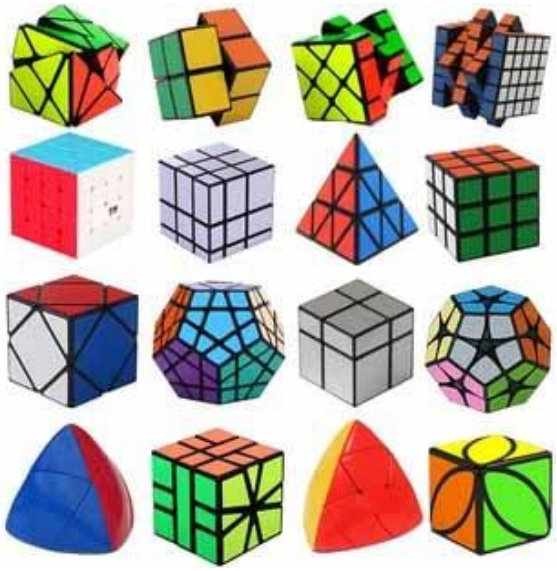

Рисунок 2 – Разновидности головоломок

Одна из самых популярных является пирамидка Мефферта (рисунок 3), которая представляет из себя правильный тетраэдр. Она имеет схожий принцип с кубиком, но разный механизм и отличное количество деталей.

Рисунок 3 – Пирамидка Мефферта в собранном состоянии

Задача решения кубика и пирамидки состоит в их сборке в исходное положение путем поворота граней. Сложность обуславливается тем, что число всех достижимых состояний довольно велико. Для кубика Рубика оно составляет приблизительно 43 квинтиллиона ($43 * 10^{18}$). Интересно, что из любого положения можно привести к единственному собранному всего за 20

ходов. На сегодняшний день проводится множество соревнований по различным дисциплинам. Например, классический кубик Рубика 3 на 3 был собран буквально за 3 секунды. [1]

Однако не все хотят тратиться на покупку головоломки, и гораздо удобней воспользоваться виртуальным аналогом. Благо, что с развитием технологий начали появляться приложения с возможностью сборки различных механических головоломок.

Цель данной работы – создать программный продукт, визуализирующий «Кубик Рубика», «Пирамидку Мефферта» и позволяющий собирать их. Для достижения цели были выделены следующие **задачи**:

1) определить требования для приложения на основе анализа аналогичных;
2) провести анализ алгоритмов удаления невидимых поверхностей и выбрать подходящий;
3) определить способ поворота грани;
4) выбрать наиболее подходящий метод окрашивания граней;
5) составить диаграммы классов, схемы алгоритмов;
6) разработать выбранные алгоритмы;
7) провести сравнительный анализ скорости работы приложения на разных режимах.

1 Аналитическая часть

В данном разделе рассматриваются существующие программы по заданной теме, анализируются алгоритмы удаления невидимых поверхностей, закрашивания, поворота точек вокруг произвольной оси. Исходя из анализа выбирается наиболее подходящий вариант.

1.1 Существующие программные обеспечения

В качестве аналогов для проекта были взяты приложения из Microsoft Store[1]. По соответствующему запросу было найдено три продукта, каждый из которых имеет как плюсы, так и минусы.

1.1.1 Приложение «Кубик Рубика»

На рисунке 5 изображен интерфейс рассматриваемого приложения.

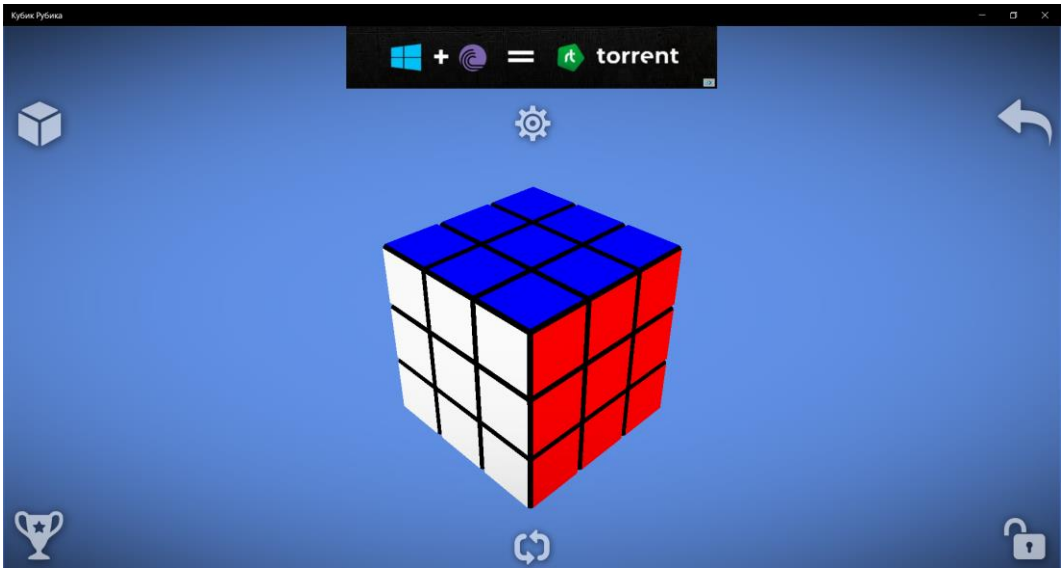

Рисунок 5 – Экран приложения «Кубик Рубика»

[1] Все программные продукты, которые будут упоминаться далее, рассмотрены на операционной системе Windows 10

Достоинства:

- возможность выбора головоломки и её размерности;
- возможность масштабирования;
- возможность вращения головоломки в разных плоскостях;
- возможность вернуть предыдущее состояние.

Недостатки:

- отсутствие возможности кастомизации;
- отсутствие кнопок для управления головоломкой.

1.1.2 Приложение «Free Super Cube»

На рисунке 6 представлен интерфейс приложения «Free Super Cube».

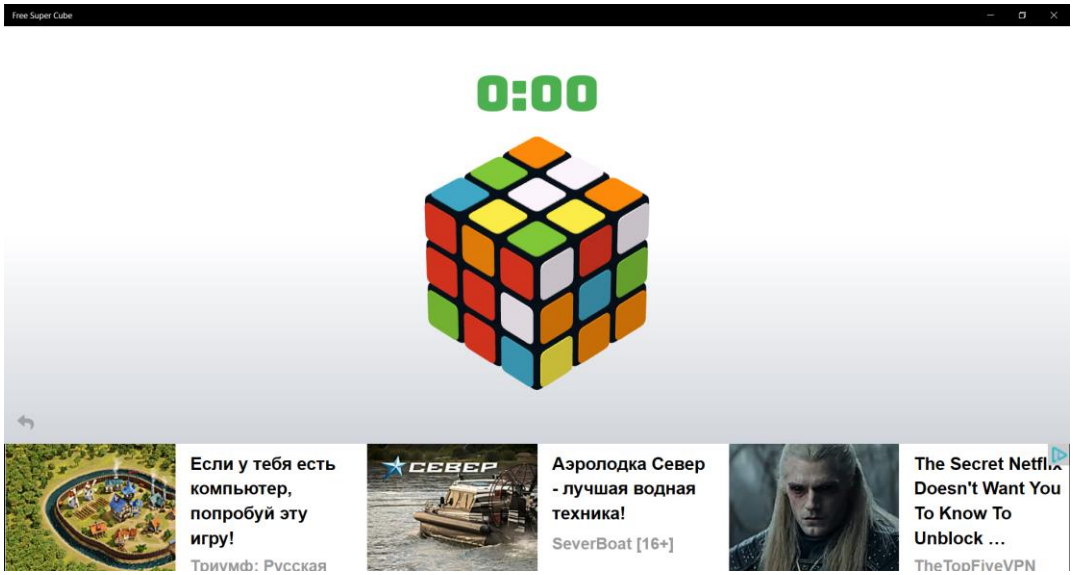

Рисунок 6 – Экран приложения «Free Super Cube»

Достоинства:

- плавный поворот граней мышкой, возможность контроля;
- наличие настроек головоломки;
- возможность выбора размерности головоломки.

Недостатки:

- поворот вокруг осей четко фиксирован;
- отсутствие выбора головоломок;
- отсутствие кнопок для управления головоломкой.

1.1.3 Приложение «Cube Genius»

На рисунке 7 приведен экран приложения «Cube Genius».

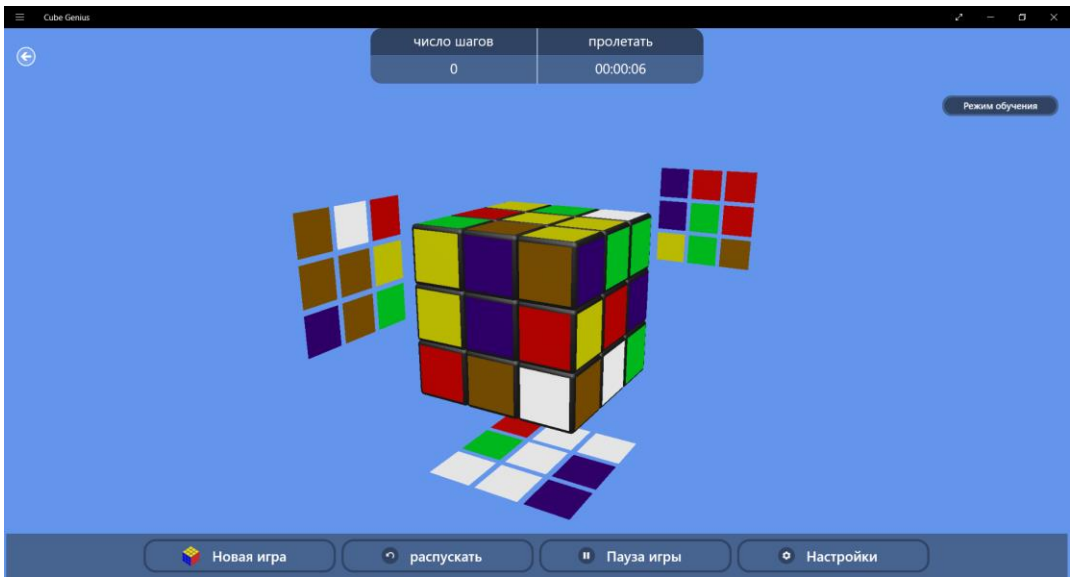

Рисунок 7 – Экран приложения «Cube Genius»

Достоинства:

- видны цвета нелицевых граней;
- отсутствие рекламы.

Недостатки:

- грани поворачиваются быстро, без контроля;
- отсутствие возможности кастомизации;
- поворот вокруг осей четко фиксирован;
- отсутствие выбора головоломок;
- отсутствие кнопок для управления головоломкой.

Вывод: анализ аналогичных приложений, выявление их плюсов и минусов показали, какие требования следует предъявить для будущего приложения.

- возможность выбора головоломки (кубик Рубика, пирамидка Мефферта) и её размерности;
- возможность вращения головоломки в разных плоскостях;
- возможность масштабирования;
- наличие настроек головоломки;
- возможность поворота граней с помощью кнопок и клавиш;
- возможность добавления до двух источников освещения.

1.2 Алгоритмы удаления невидимых поверхностей.

Задача удаления невидимых поверхностей является одной из самых важных и сложных в компьютерной графике. Выделяют три класса данных алгоритмов:

1) работающие в объектном пространстве – работают с физической системой координат, в которой описаны объекты. Получаются весьма точные результаты, объем вычислений растет, как квадрат числа объектов;
2) работающие в пространстве изображения – работают с системой координат экрана, на котором визуализируются объекты. Точность вычислений ограничена разрешением экрана, объем вычислений растет, как произведение количества объектов на число пикселей;
3) формирующие списки приоритетов – работают попеременно с физической и системой координат экрана. [2]

Так как взаимодействие между пользователем и программой происходит в режиме реального времени, то одной из главных характеристик будет время, затрачиваемое на алгоритм.

1.2.1 Алгоритм Робертса

Алгоритм Робертса является первым известным решением задачи удаления невидимых линий. Он позволяет определить, какие рёбра или часть рёбер видимы, а какие заслонены другими гранями. [3] Относится к алгоритмам, работающим в объектном пространстве. Алгоритм Робертса проходит в два этапа.

На первом этапе необходимо определить нелицевые грани для каждого объекта. Плоскость, которая содержит некоторую грань многогранника, разделяет пространство на два подпространства. Положительным называют то, в которое направлена внешняя нормаль к грани, и если точка наблюдения в нем, то грань лицевая, иначе – нелицевая. Для того, чтобы определить это, используется проверка знака скалярного произведения двух векторов: l – радиус-вектор, направленный к наблюдателю, n – вектор внешней нормали грани. Если знак положительный, то грань лицевая (угол между векторами острый), если отрицательный, то угол тупой, то есть грань является нелицевой. Для этого любое выпуклое тело представляют в виде матрицы тела, состоящей из коэффициентов уравнения плоскостей. Затем происходит скалярное умножение этой матрицы на точку (вектор точки), которая соответствует точке наблюдения. Положительное скалярное произведение дает только такая плоскость, относительно которой точка лежит снаружи.

На втором этапе происходит определение и удаление невидимых ребер. Каждое из ребер последовательно сравнивается со всеми остальными телами. В классической версии количество вычислений растет теоретически как квадрат числа объектов. [4]

Достоинства:

- относительная простота.

Недостатки:

- объем вычислений растет, как квадрат числа объектов;
- ориентирован на работу с выпуклыми многогранниками;

- невозможность работы с прозрачными объектами.

1.2.2 Алгоритм художника

Алгоритм художника позволяет определить, какие грани видимы, а какие заслонены, выводит каждую из них целиком по мере приближения к наблюдателю, то есть подобно тому, как художник наносит изображением на холст слой за слоем. Изначально все объекты необходимо отсортировать от заднего плана к переднему. [5] Данный алгоритм относится к алгоритмам, работающим в пространстве изображения.

Достоинства:

- при тривиальной сортировке обладает высоким быстродействием;
- простота.

Недостатки:

- сортировки по оси Z может быть нетривиальной задачей;
- при некотором расположении граней не может дать верный результат;
- происходит отрисовка всех граней объектов, из-за чего время работы может сильно увеличиться.

1.2.3 Алгоритм Z-буфера

Алгоритм Z-буфера позволяет определить, какие пиксели граней видны, а какие заслонены другими. Является одним из простейших алгоритмов удаления невидимых поверхностей, работает в пространстве изображения. [3] Можно сказать, что это обобщение идеи о буфере кадра. Он используется для запоминания цвета каждого пикселя. Z-буфер же является двухмерным массивом, размеры которого равны размерам окна, и в каждой ячейке хранится значение глубины пикселя. В процессе работы z-координата каждого пикселя сравнивается с соответствующим значением в Z-буфере. Если она ближе, то

значения в двух буферах обновляются. [6] Производительность можно оценить, как *O(CN)*, где *N* – количество граней, *C* – количество пикселей [3]

Достоинства:

- отсутствие предварительной сортировки по глубине;
- делает тривиальной визуализацию пересекающихся поверхностей, сцены могут быть любой сложности;
- простота;
- высокая производительность.

Недостатки:

- требуется большой объем памяти;
- высокая стоимость устранения лестничного эффекта, невозможность реализовать эффекты прозрачности и просвечивания.

1.2.4 Вывод

На основе анализа алгоритмов удаления невидимых поверхностей можно осуществить выбор наиболее подходящего. Одним из основных факторов является время выполнения работы, при этом желательно, чтобы оно не зависело от количества объектов (в данном случае – деталей). Поэтому алгоритм Робертса можно отбросить, так как время возрастает, как квадрат числа объектов. Однако можно заметить, что первый этап может подойти для отрисовки головоломки и её вращении, при этом это не будет затратным по времени. Этого будет достаточно, так как она в таком случае головоломка представляет собой единый предмет без разделения на элементы. При повороте грани такой подход не сможет дать верный результат, так как одни элементы будут перекрывать другие. Для отрисовки поворачиваемой грани предлагается использовать либо алгоритм Z-буфера, либо алгоритм художника. Так как для использования первого необходимо создавать буфер кадра и Z-буфер для всего экрана, то при небольших размерах головоломки это будет неэффективно (также учитывая тот факт, что алгоритм будет применяться для одной грани).

Сортировку же по глубине возможно сделать тривиальной. Для этого на каждой стороны всех деталей грани, которая поворачивается, определяется среднее арифметическое значение координаты Z, и происходит сортировка в порядке возрастания. На основе полученного порядка отрисовываются детали.

Стоит обратить внимание, что невозможна ситуация, когда грани перекрывают друг друга одновременно. При этом количество объектов не будет слишком большим. То есть минусы алгоритма художника ликвидируются, и можно остановить выбор на нем. Таким образом, был составлен алгоритм из двух этапов:

1) определение нелицевых граней всего объекта;
2) в случае поворота грани применение для неё алгоритма художника.

1.3 Поворот грани

Задача решения головоломки состоит в её переводе в изначальное положение путем поворота граней. Для поворота необходимо решить проблему поворота точки вокруг произвольной оси.

Известно, как следует проводить поворот вокруг координатной оси, поэтому основная идея заключается в том, чтобы совместить произвольную ось вращения с одной из координатных. То есть если произвольная ось проходит через точку (x_0, y_0, z_0) с единичным направляющим вектором (c_x, c_y, c_z), то поворот на угол δ осуществляется в следующие этапы:

- выполнить перенос так, чтобы (x_0, y_0, z_0) лежала в начале координат;
- выполнить повороты так, чтобы ось вращения совпала с осью **y** (либо любой другой);
- выполнить поворот на угол **δ** вокруг оси **y**:
- выполнить повороты, обратные тем, которые позволили совместить ось вращения с осью **y**;
- выполнить обратный перенос.

В общем случае для совмещения произвольной оси с координатной, необходимо выполнить два последовательных поворота вокруг двух других. Чтобы совместить с осью **y**, следует сначала выполнить вокруг оси **z**, затем – вокруг **x**.

Для определения угла **α** вокруг оси **z**, используемый для перевода оси в плоскость **yz**, направляющий вектор этой оси сначала проецируется на плоскость **xy**. Компоненты **x** и **y** спроецированного вектора равны $\mathbf{c_x}$ и $\mathbf{c_y}$ соответственно (компонентам единичного направляющего вектора оси вращения). Его длину можно определить по формуле 1.

$$d = \sqrt{c_x^2 + c_y^2} \tag{1}$$

А косинус и синус угла поворота рассчитать по формулам 2 и 3.

$$\cos(\alpha) = \frac{c_y}{d} \tag{2}$$

$$\sin(\alpha) = \frac{c_x}{d} \tag{3}$$

После перевода в плоскость **xy** необходимо выполнить поворот х. **z**-компонента единичного вектора равна **d**, а **x**-компонента равна $\mathbf{c_x}$. Длина единичного вектора равна 1. Тогда синус и косинус угла поворота определяются по формулам 4 и 5.

$$\cos(\beta) = d \tag{4}$$

$$\sin(\beta) = c_x \tag{5}$$

На практике углы не вычисляются, а можно оставить в виде тригонометрических функций. Так как компоненты единичного направляющего

вектора неизвестны, то можно нормализовать вектор, соединяющий первую и вторую точку. [8]

Вывод: в результате анализа алгоритма поворота точки вокруг произвольной оси была математически решена проблема поворота каждой грани головоломки.

1.4 Метод закрашивания

Так как в сцене предполагается наличие источников освещения, необходимо применять некоторый метод закрашивания для затенения полигонов. Уровень освещенности (интенсивность излучаемого света) в каждой точке можно найти по закону Ламберта (формула (6)).

$$I = I_0 \cos(\alpha) \tag{6}$$

где I_0 – интенсивность излучения в направлении нормали к поверхности, то есть максимальный уровень освещенности;
α – угол между вектором нормали поверхности и вектором, который направлен от источника освещения к рассматриваемой точке. [9]

Так как в работе происходит обработка многогранников, имеющих диффузное отражение, то следует использовать простую закраску, которая даст результат, удовлетворяющий требованиям приложения. В таком случае каждая грань имеет уровень интенсивности, вычисляющийся по закону Ламберта. То есть все поверхности закрашиваются однотонно. [7]

Так как алгоритм обладает относительной простотой, то имеет довольно высокое быстродействие. Его недостаток в виде отсутствия работы с фигурами вращения (будут видны ребра) не важен ввиду их отсутствия в разрабатываемой сцене.

Вывод: в результате был сделан выбор в пользу простого метода окрашивания, и вследствие чего решена проблема закраски граней головоломки с учетом наличия источника освещения.

1.5 Вывод

В ходе анализа аналогичных приложений, алгоритмов удаления невидимых поверхностей, алгоритма поворота вокруг произвольной оси, методов закрашивания сделан выбор, благодаря которому:

1) определены требования будущей программы:
   - возможность выбора кубика или пирамидки и их размерности;
   - возможность вращения головоломки в разных плоскостях;
   - возможность масштабирования;
   - наличие настроек головоломки;
   - возможность поворота граней с помощью кнопок и клавиш;
   - возможность добавления до двух источников освещения;
2) выбран алгоритм удаления невидимых поверхностей, который будет состоять из двух этапов:
   - определение нелицевых граней всего объекта;
   - в случае поворота грани применение для неё алгоритма художника.
3) решена проблема поворота грани вокруг оси;
4) решена проблема закраски граней – выбран простой метод закрашивания.

2 Конструкторская часть

В данном разделе приведены диаграммы классов, схемы выбранных алгоритмов, на основе которых в дальнейшем будет реализована программа. В ходе разработки будет использоваться парадигма объектно-ориентированного программирования, что позволит создать код, который станет легко модифицируемым и с возможным расширением (например, в виде добавления новых функций и головоломок).

2.1 Диаграммы классов

Для описания классов используется унифицированный язык моделирования – UML, на котором можно довольно просто описать создаваемые классы.

2.1.1 Связи между классами

На рисунке 8 приведено описание связей между классами. Для наглядности методы и атрибуты всех классов удалены.

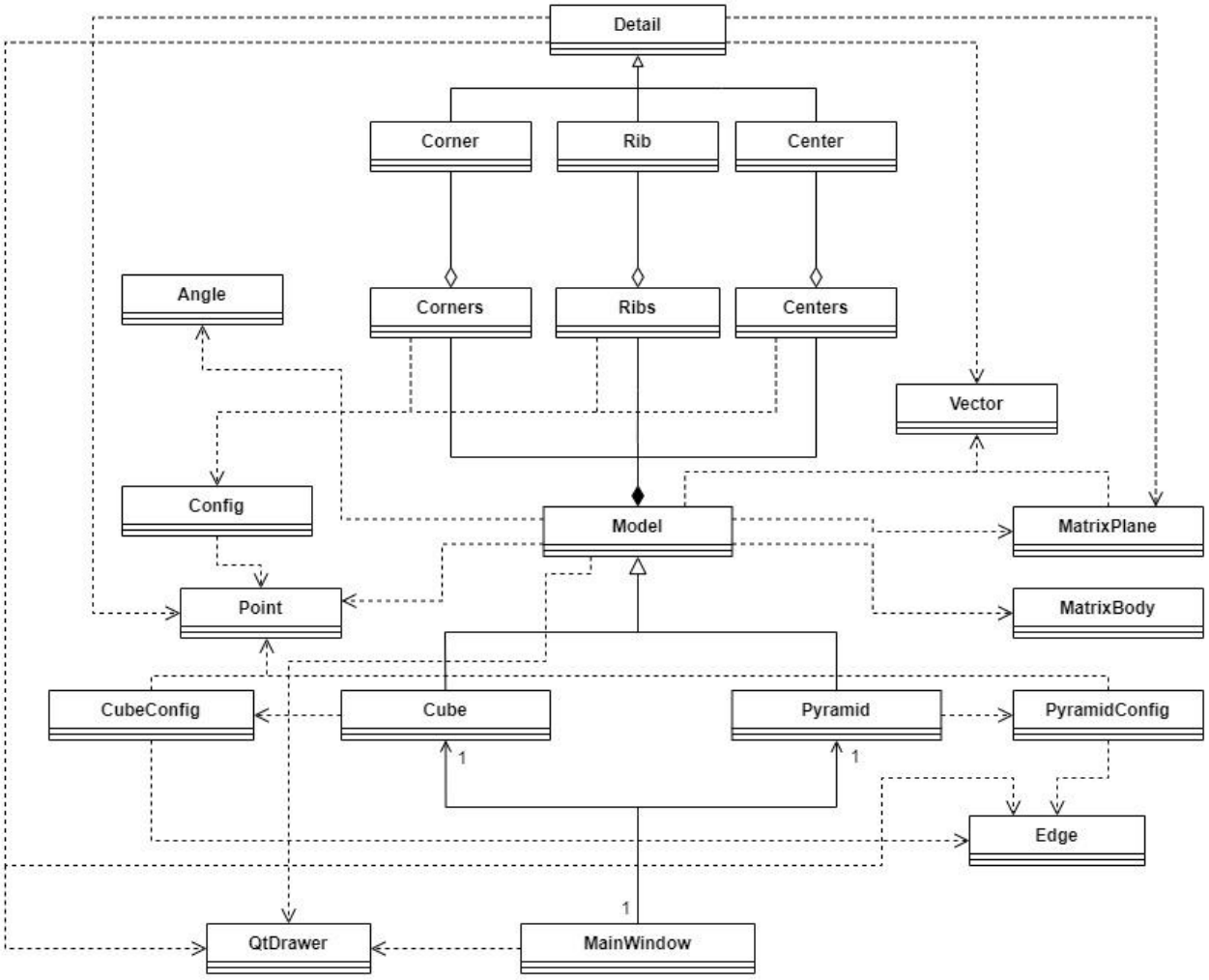

Рисунок 8 – Связи между классами

Для создания объекта проведена декомпозиция на центральные, угловые и реберные элементы. Для удаления нелицевых граней будут созданы классы матриц плоскости и тела. Также присутствуют классы конфигурационных данных, что в общем случае позволит подменить один класс на другой, чтобы изменить выбранную головоломку. Это позволит гибко управлять моделью и задавать ей желаемое поведение.

2.1.2 Конфигурационные классы

На рисунке 9 представлены диаграммы классов Config (в котором хранятся общие данные), CubeConfig (данные для кубика) и PyramidConfig (данные для пирамидки).

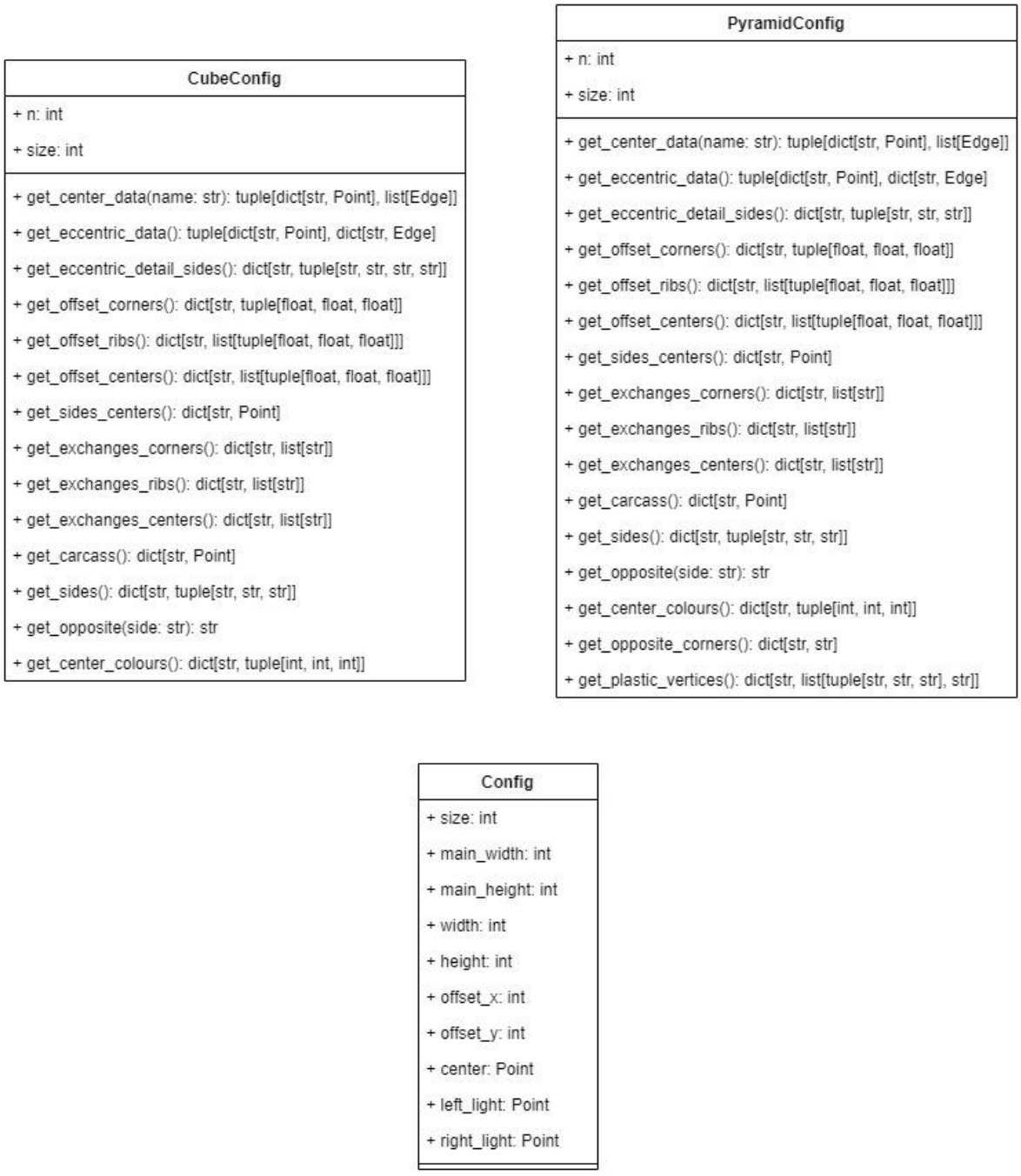

Рисунок 9 – Диаграммы конфигурационных классов

2.1.3 Вспомогательные классы

На рисунке 10 изображены диаграммы вспомогательных классов.

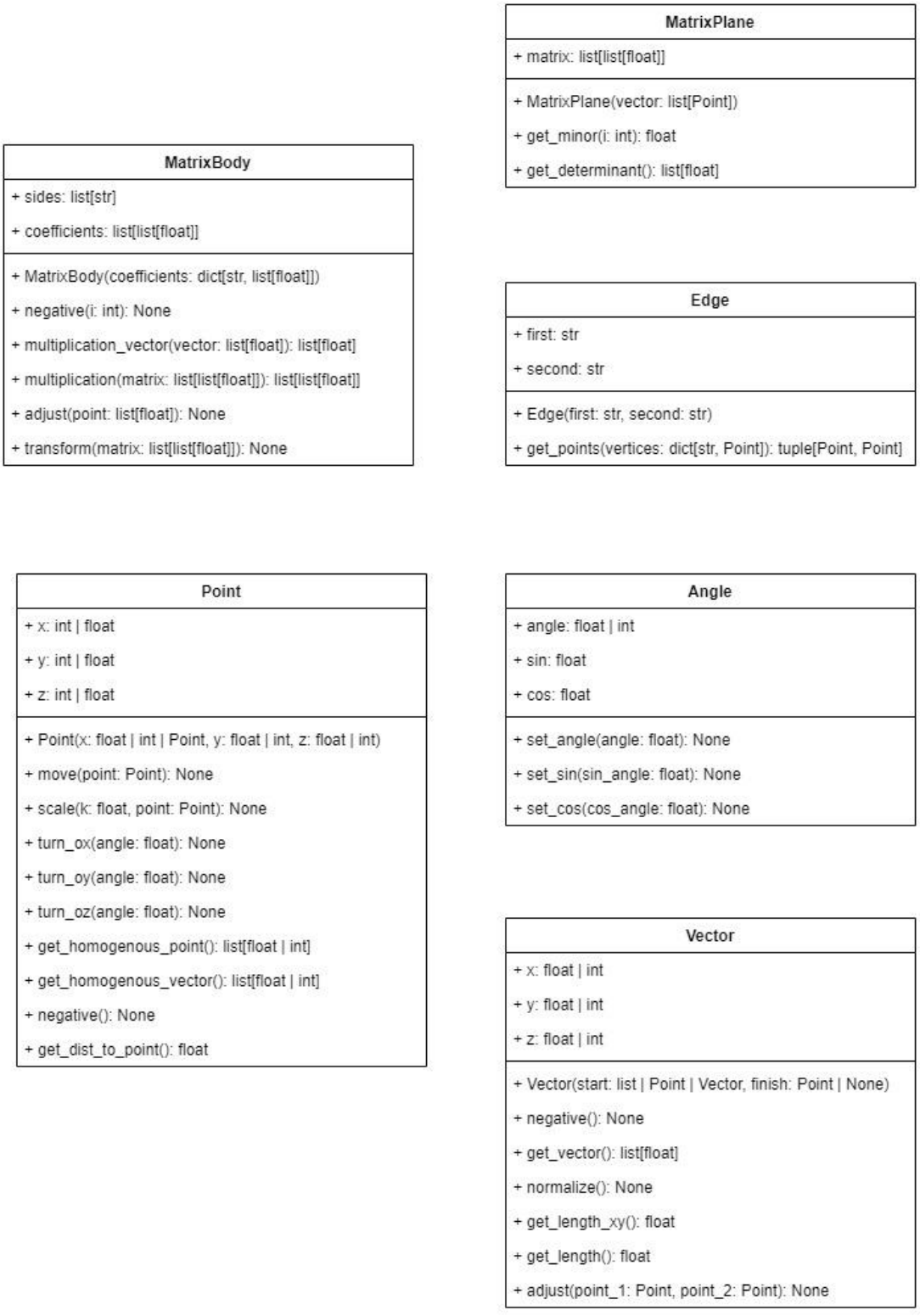

Рисунок 10 – Диаграммы вспомогательных классов

Классы Point и Edge позволяют удобно производить работу со всеми ребрами и вершинами, MatrixPlane и MatrixBody – задавать матрицы для дальнейших вычислений, Angle и Vector – для обработки углов и векторов.

2.1.4 Классы деталей

Все детали головоломки поделены на три вида: угловые, реберные и центральные. Каждая из них обладает своими свойствами. Однако очевидно, что независимо от вида у них имеются общие методы и атрибута. Поэтому при проектировании это следует учитывать (в виде наследования). На рисунке 11 представлены диаграммы классов детали и трёх её видов.

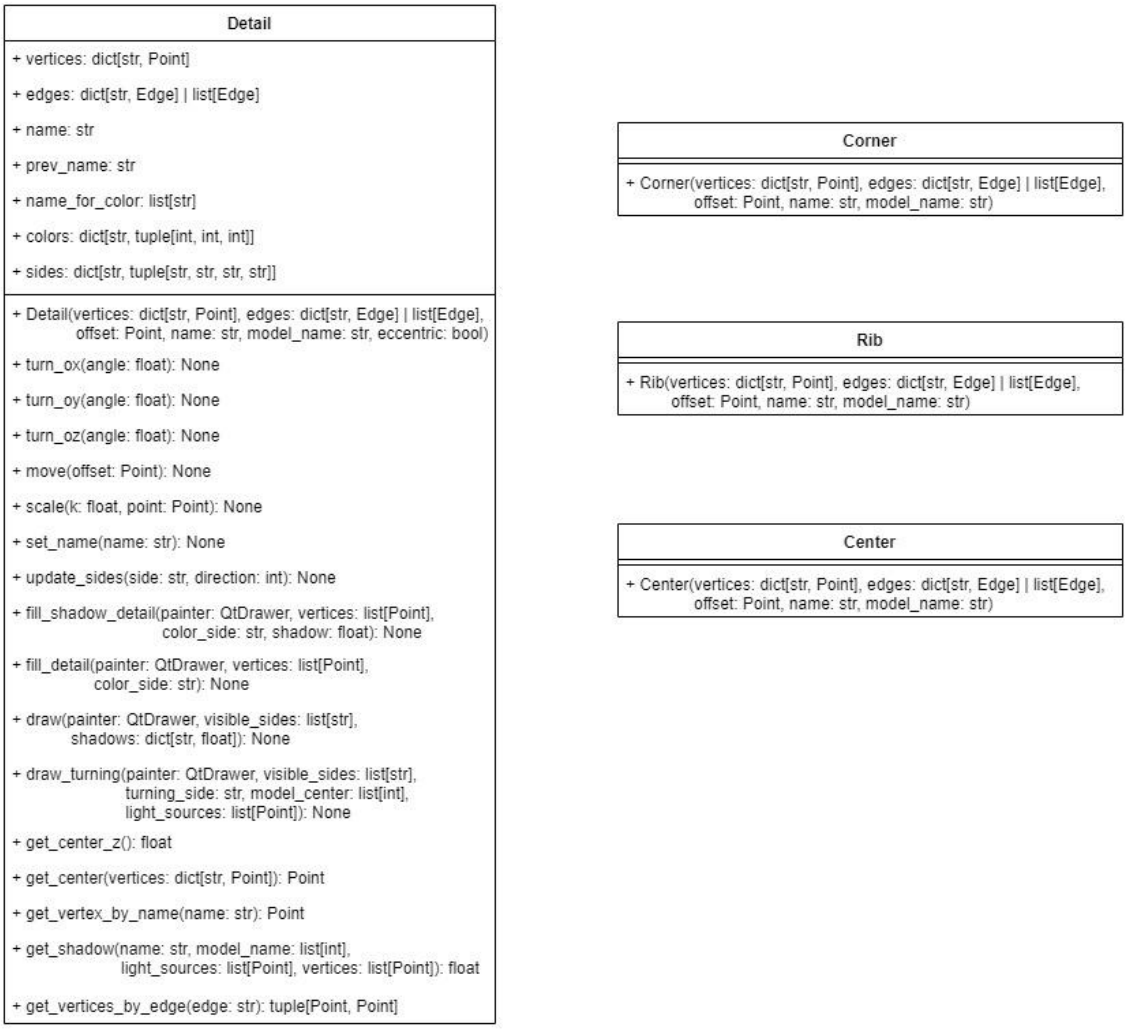

Рисунок 11 – Диаграммы классов деталей

Но у головоломки не одна деталь, поэтому следуют создать классы, которые будут позволять работы с деталями в совокупности. На рисунке 12 показаны диаграммы данных классов.

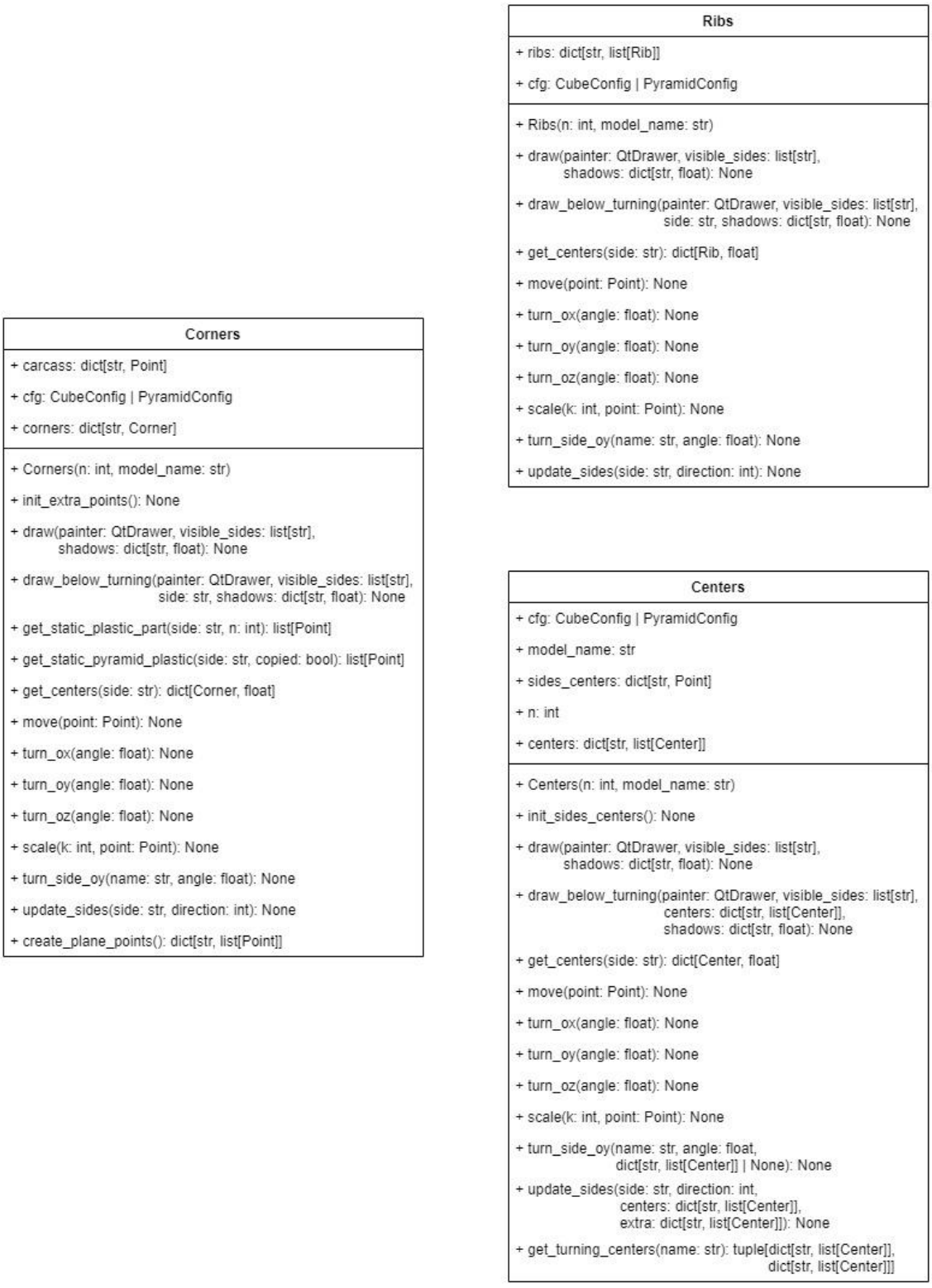

Рисунок 12 – Диаграммы классов деталей в совокупности

2.1.5 Классы моделей

Кубик Рубика и пирамидка Мефферта относятся к механическим головоломкам и обладают одинаковыми методами (за небольшим исключением). Некоторые реализации в зависимости от выбранного вида могут меняться. На рисунке 13 представлены диаграммы классов моделей.

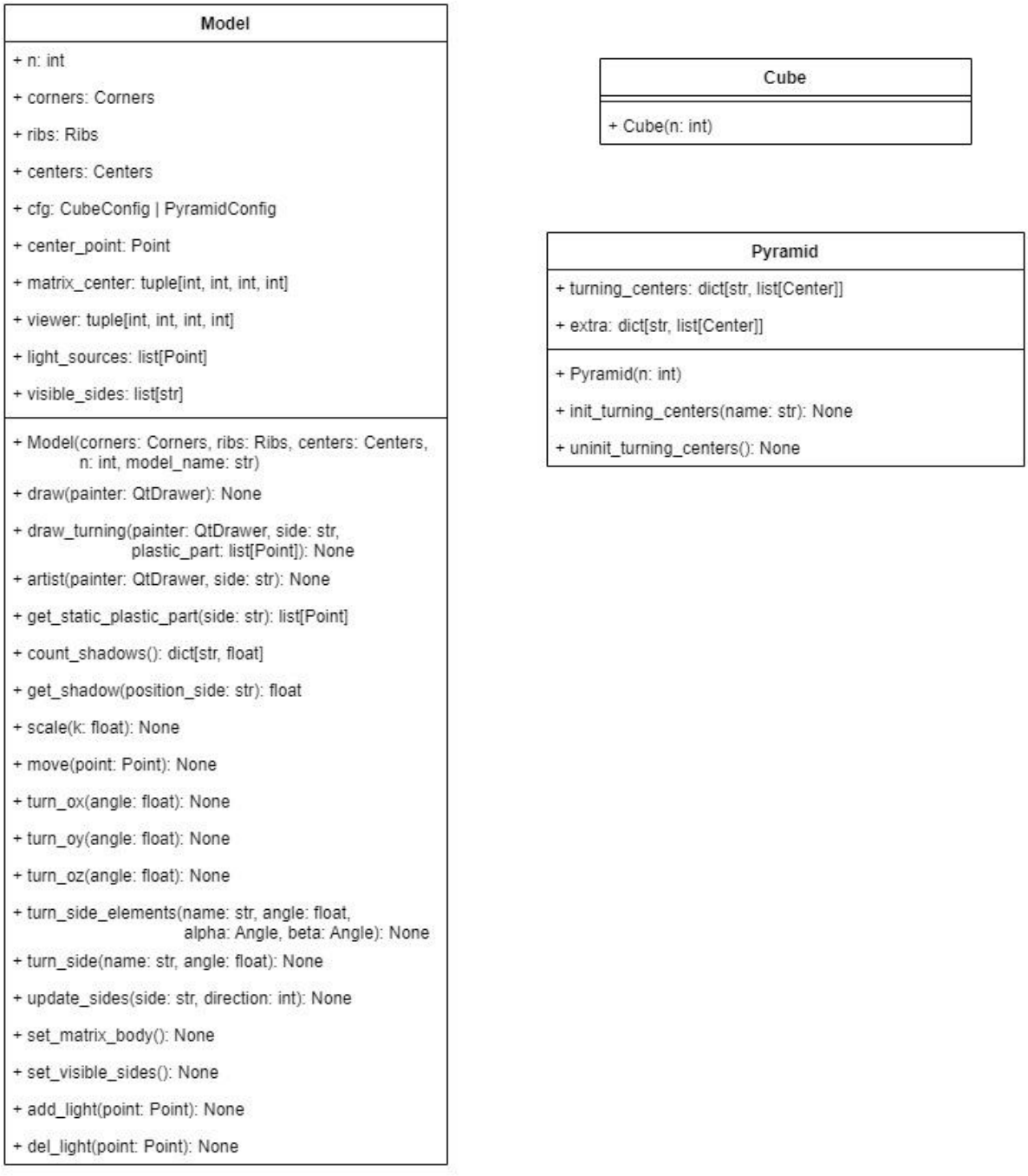

Рисунок 13 – Диаграммы классов моделей

2.1.6 Классы интерфейса

Взаимодействие пользователя с приложением происходит через интерфейс, из которого возможен вызов всех соответствующих методов классов, находящихся ниже по иерархии. Также необходим класс для отрисовки моделей. На рисунке 14 изображены эти два класса.

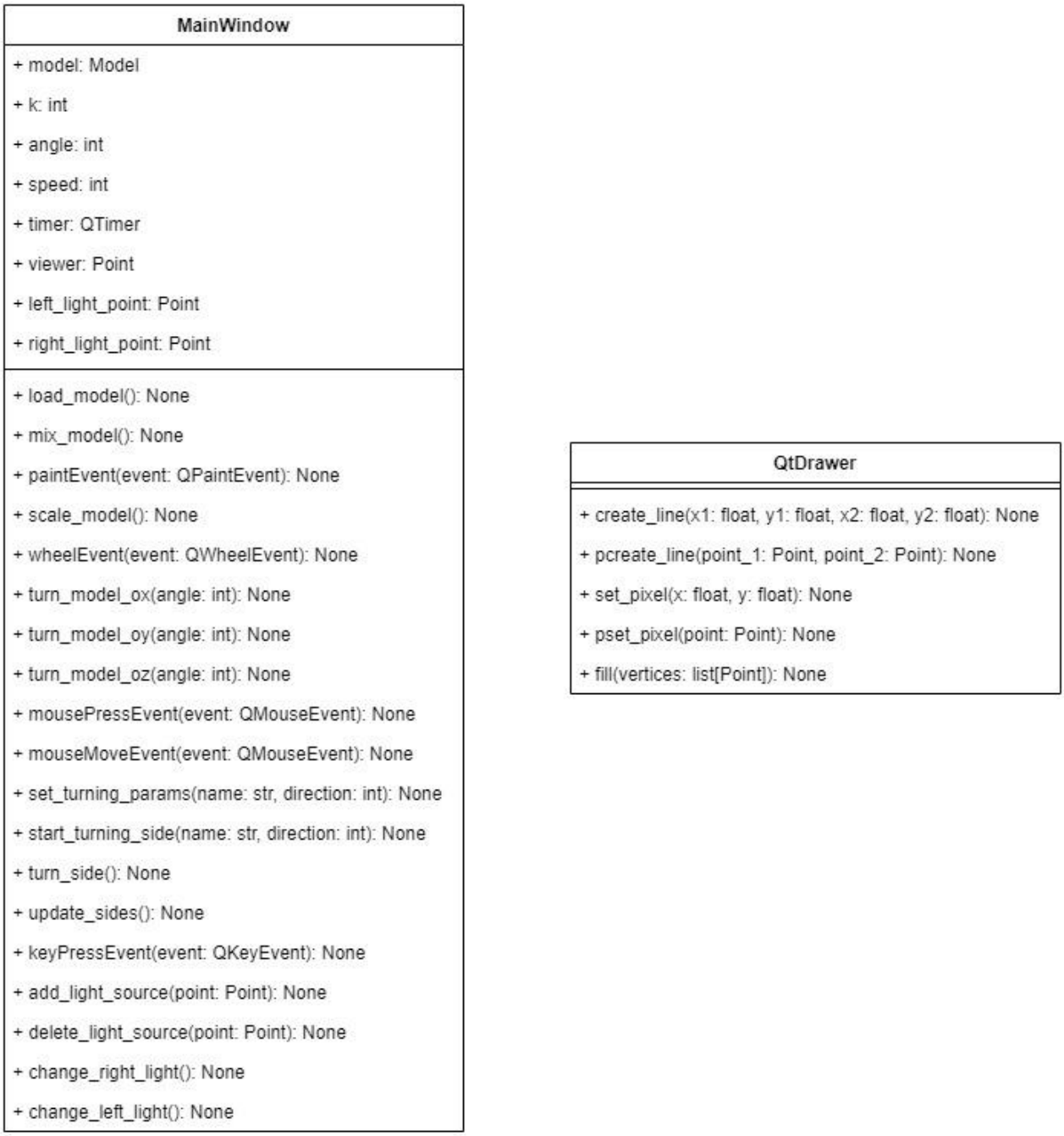

Рисунок 14 – Диаграммы классов интерфейса

2.2 Схемы алгоритмов

Перед реализацией алгоритмов необходимо составить их схемы, чтобы понять основные этапы и структурировать теоретические знания о них.

2.2.1 Алгоритм удаления невидимых поверхностей

В ходе анализа алгоритмов удаления невидимых поверхностей был составлен двухэтапный алгоритм, содержащий удаление нелицевых граней и алгоритма художника (в случае поворота грани).

На рисунках 15-16 представлен алгоритм определения граней, которые видны наблюдателю. На основе полученного результата можно убрать нелицевые грани. Данный алгоритм применим в случае поворота модели вокруг своих осей.

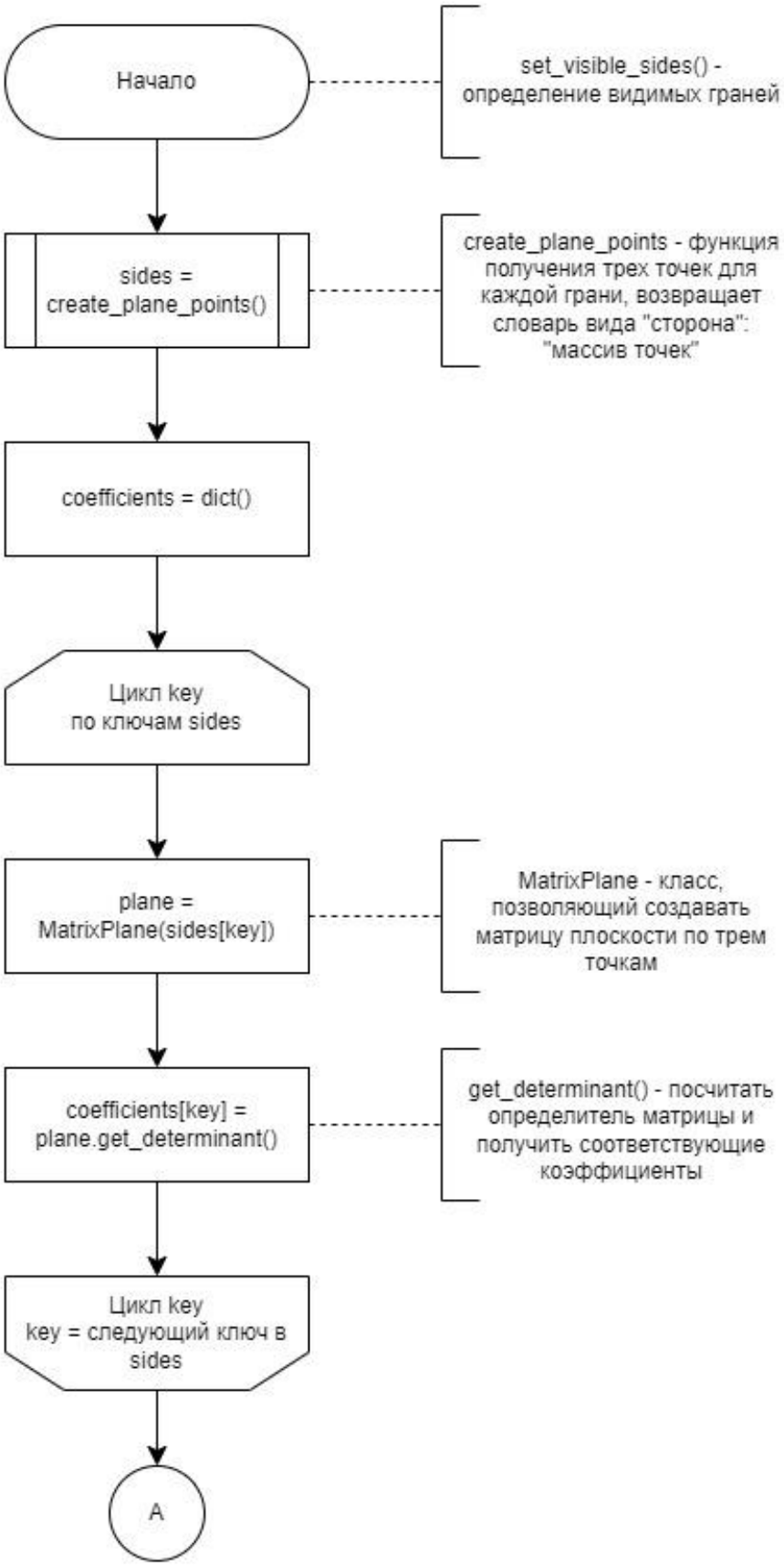

Рисунок 15 – Схема алгоритма определения лицевых граней (часть 1)

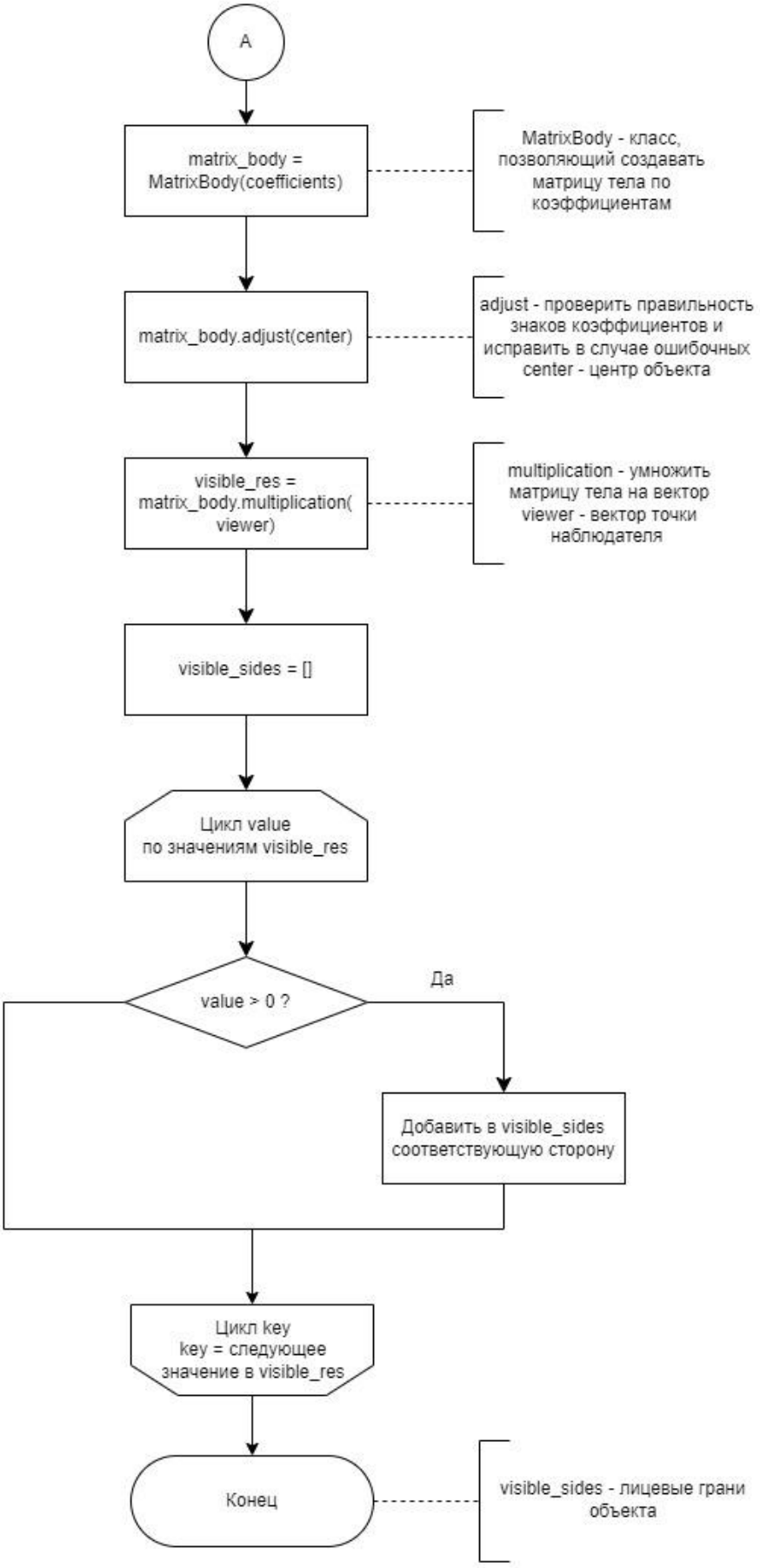

Рисунок 16 – Схема алгоритма определения лицевых граней (часть 2)

При повороте грани невозможно воспользоваться предыдущим алгоритмом, так как нарушается целостность общего вида куба или пирамиды. Применение алгоритма художника позволит получить нужный результат. Детали поворачиваемой грани сортируется по среднему значению координаты Z. Это позволит эффективно решить поставленную задачу. На рисунке 17 представлен алгоритм художника, на рисунке 18 – алгоритм отрисовки модели во время поворота грани.

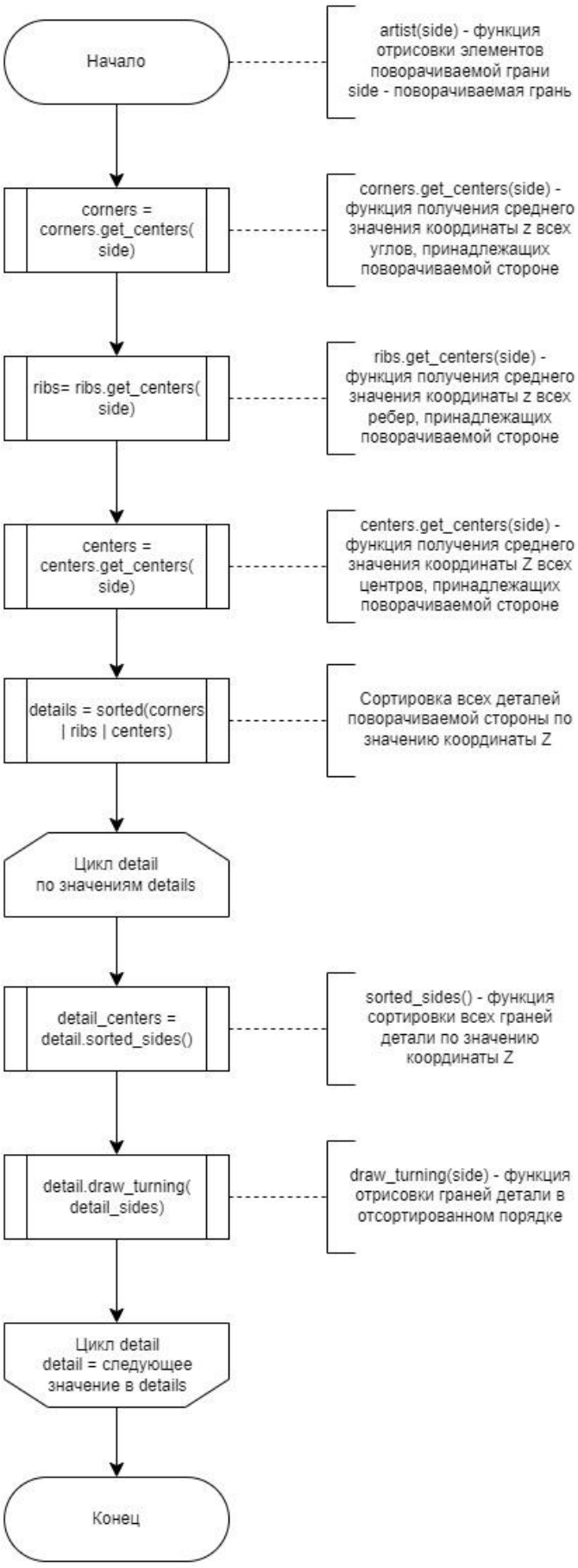

Рисунок 17 – Схема алгоритма художника

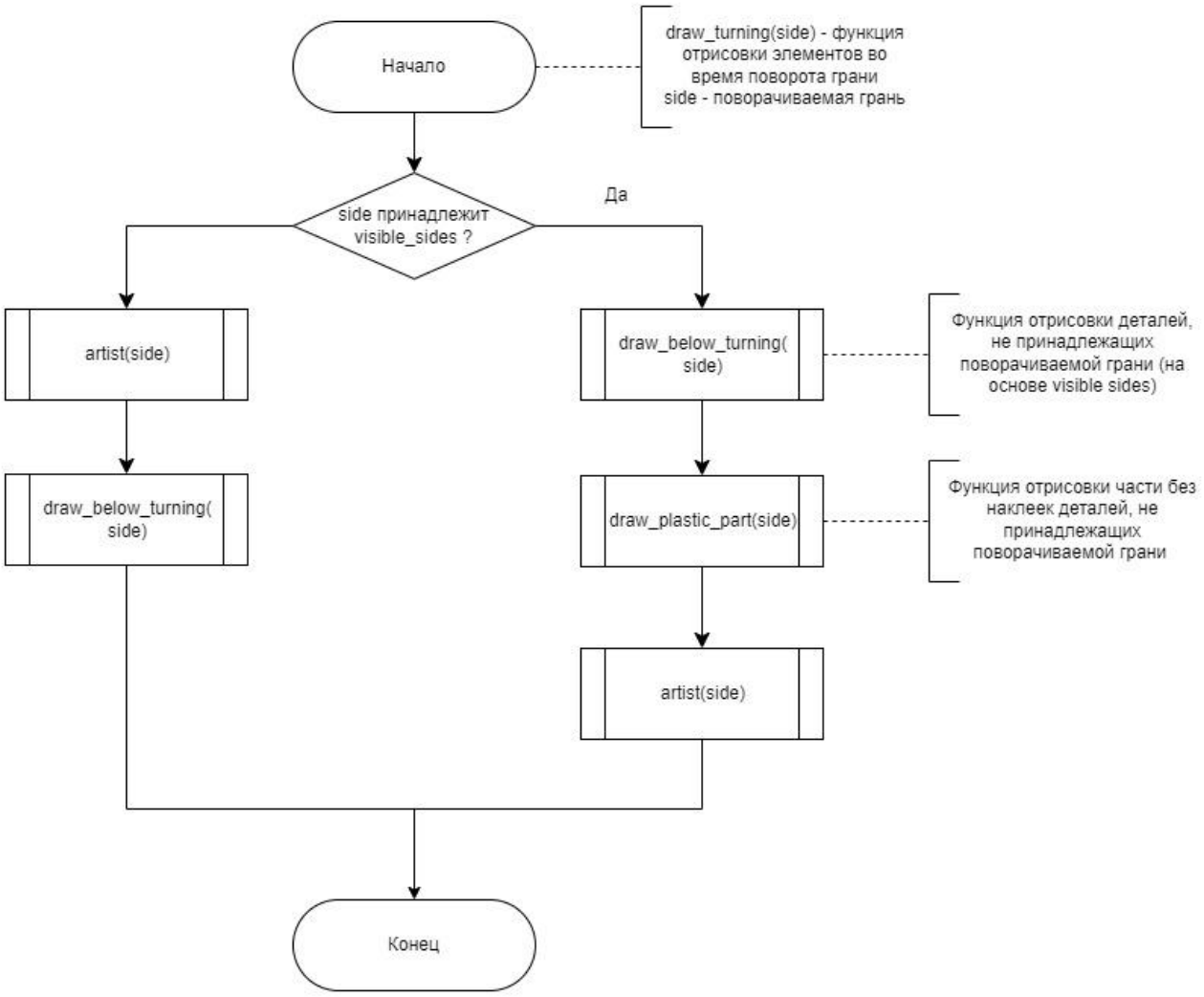

Рисунок 18 – Схема алгоритма отрисовки поворачиваемой грани

2.2.2 Алгоритм поворота грани

Для поворота грани необходимо совершить поворот всех необходимых точек вокруг произвольной оси. Задача не является тривиальной, так как нет алгоритмов, позволяющих сделать это сразу. Поэтому сначала необходимо повернуть объект, чтобы совместить ось, вокруг которой совершается поворот, например, с осью ординат. Только после этого можно будет выполнить сам поворот. На рисунках 19-20 представлены схема алгоритма поворота грани.

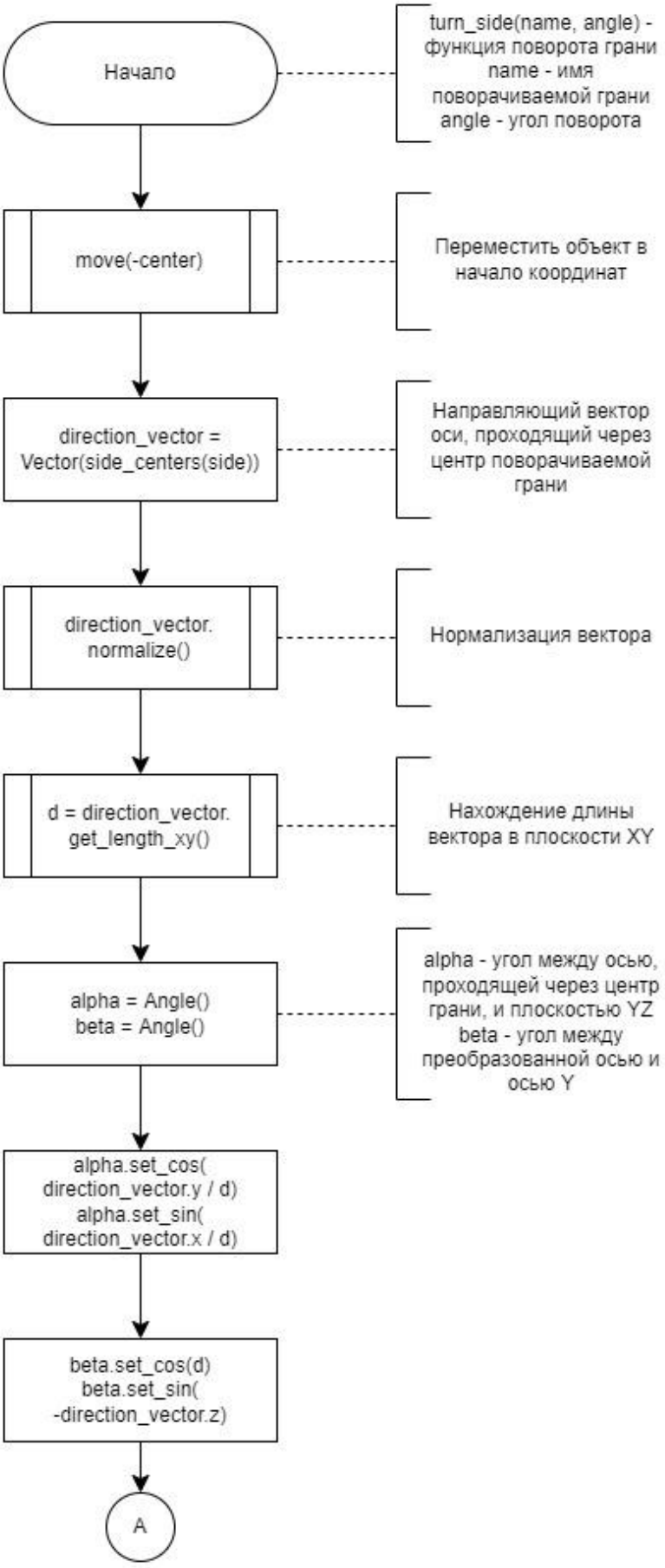

Рисунок 19 – Схема алгоритма поворота грани (часть 1)

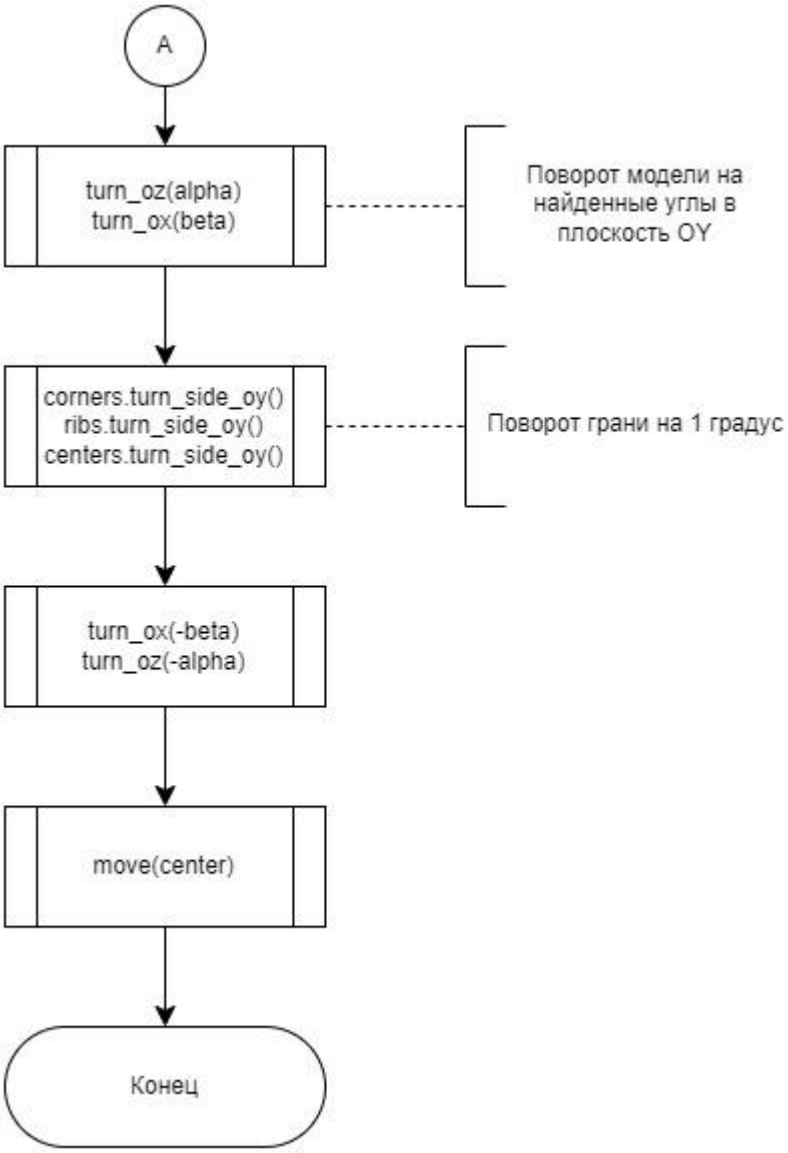

Рисунок 20 – Схема алгоритма поворота грани (часть 2)

2.2.3 Алгоритм закрашивания

Для закрашивания элементов был выбран простой метод, который позволяет рассчитать интенсивность света для каждой грани на основе косинуса угла между нормалью и вектором, соединяющим центр стороны и точку источника света. При повороте грани интенсивность рассчитывается для каждой детали. На рисунках 21-22 представлена схема алгоритма закрашивания.

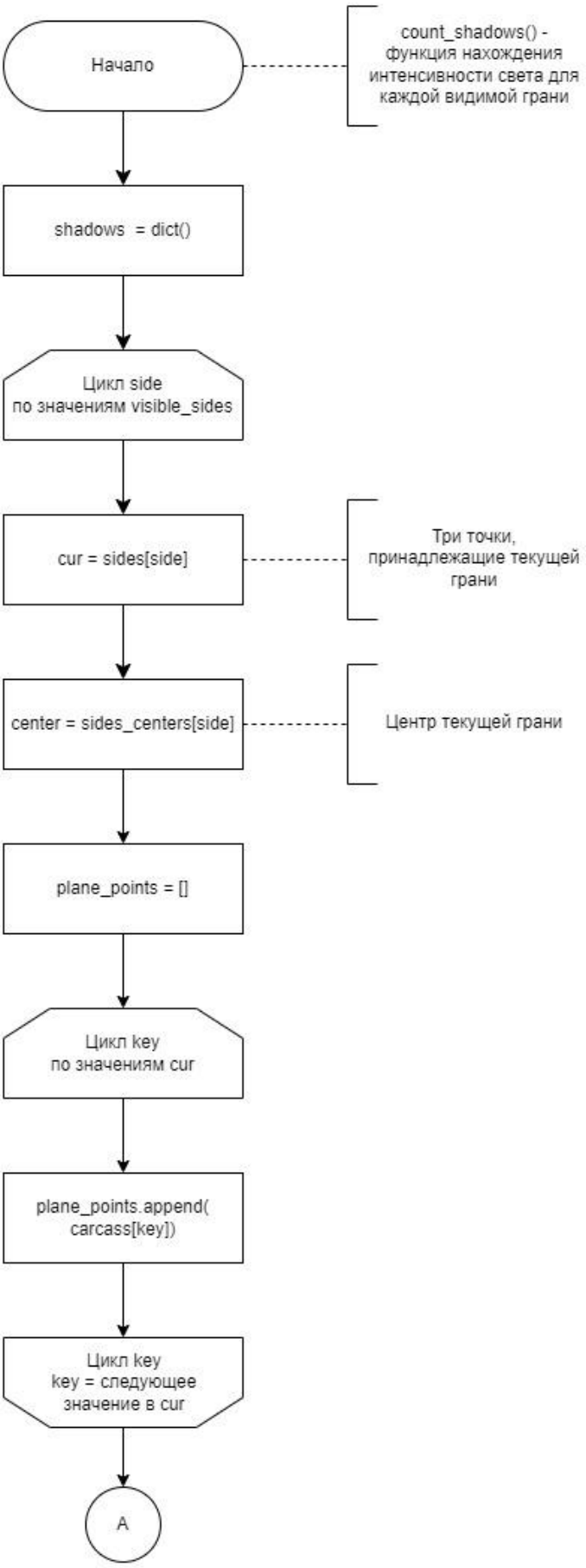

Рисунок 21 – Схема алгоритма закрашивания (часть 1)

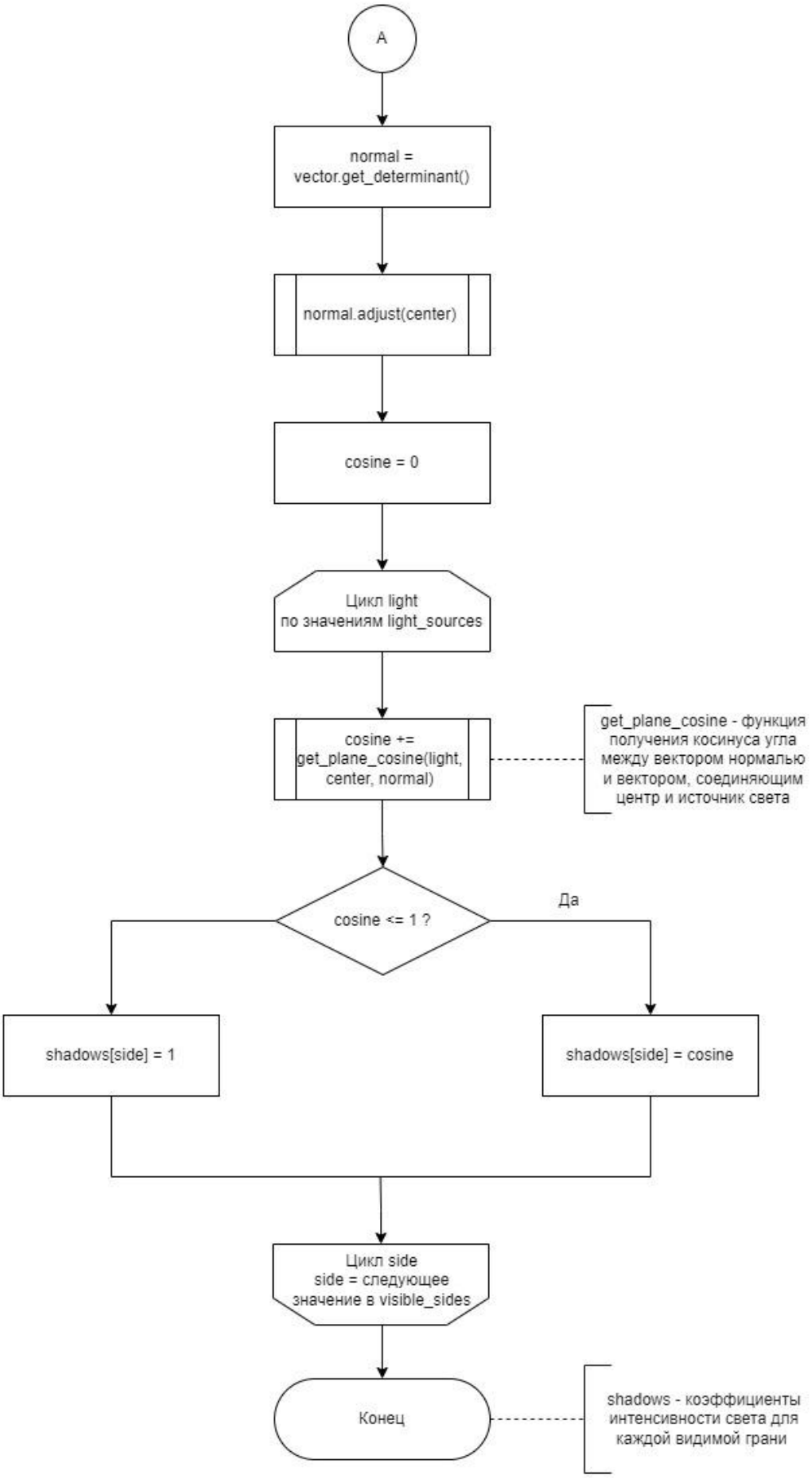

Рисунок 22 – Схема алгоритма закрашивания (часть 2)

2.3 Вывод

В результате разработки составлены схемы алгоритмов, на основе которых можно сделать их реализацию. Диаграммы классов позволили понять, какую структуру должны иметь объекты в программе и какая связь между ними.

3 Технологическая часть

В данном разделе обосновывается выбор используемого языка программирования, приводятся листинги реализации алгоритмов, интерфейс и тестирование разработанного приложения.

3.1 Инструменты для реализации и исследования

Для реализации разрабатываемого программного продукта выбран язык программирования Python. Благодаря своей простоте и гибкости он может быть использован во многих сферах, и компьютерная графика – не исключение. Python предоставляет возможность быстрой разработки. Также поддерживается как структурный стиль программирования, так и объектно-ориентированный, что позволяет сочетать их, делая программу более эффективной. К минусам можно отнести довольно невысокую производительность за счет динамической типизации и ряда других причин. Однако грамотная реализация поставленной задачи может помочь избежать данную проблему.

В языке имеется огромное количество библиотек, в том числе и для графического интерфейса. В качестве такой выбрана библиотека PyQt5, которая предоставляет удобные возможности для разработки.

Для замера времени используется функция process_time() из библиотеки time, которая возвращает процессорное время текущего процесса.

3.2 Реализация алгоритмов

В ходе работы реализованы алгоритмы определения лицевых граней (листинги 1-2), отрисовки поворачиваемой грани (листинг 3), поворота граней (листинги 4-5) и нахождения коэффициентов интенсивности света для каждой грани (листинг 6).

Листинг 1 – Алгоритм определения лицевых граней (часть 1)

```
class MatrixPlane:
    def __init__(self, vector):
        self.matrix = []
        point_1, point_2, general = vector[0], vector[1], vector[2]

        self.matrix.append(Vector(general, Point()).get_vector())
        self.matrix.append(Vector(general, point_1).get_vector())
        self.matrix.append(Vector(general, point_2).get_vector())

    def get_minor(self, i):
        minor_matrix = [self.matrix[j][:i] + self.matrix[j][i+1:] for j in
range(1, 3)]
        return minor_matrix[0][0] * minor_matrix[1][1] - minor_matrix[1][0] *
minor_matrix[0][1]

    def get_determinant(self):
        result = []
        d = 0

        for i in range(3):
            minor = self.get_minor(i)
            if i == 1:
                minor *= -1
            result.append(minor)
            tmp = self.matrix[0][i] * minor
            d += tmp

        result.append(d)
        return result


class MatrixBody:
    def __init__(self, coefficients):
        self.sides = coefficients.keys()
        self.coefficients = list(coefficients.values())
        self.size = len(self.coefficients)

    def __str__(self):
        result = ''
        for i in range(len(self.coefficients[0])):
            for j in range(self.size):
                result += f'{self.coefficients[j][i]:^14.1f}'
            result += '\n'

        return result

    def negative(self, i):
        self.coefficients[i] = [-coefficient for coefficient in
self.coefficients[i]]
```

Листинг 2 – Алгоритм определения лицевых граней (часть 2)

```
    def multiplication_vector(self, vector):
        result = []

        for i in range(self.size):
            result.append(0)
            for j in range(len(vector)):
                result[i] += vector[j] * self.coefficients[i][j]

        return result

    def multiplication(self, matrix):
        result = [[] for _ in range(self.size)]
        size = len(matrix[0])

        for i in range(len(matrix)):
            for j in range(self.size):
                result[j].append(0)
                for k in range(size):
                    result[j][i] += matrix[i][k] * self.coefficients[j][k]

        return result

    def adjust(self, point):
        result = self.multiplication_vector(point)
        for i in range(len(result)):
            if result[i] > 0:
                self.negative(i)

    def transform(self, matrix):
        self.coefficients = self.multiplication(matrix)


def set_matrix_body(self):
    sides = self.corners.create_plane_points()
    coefficients = {}

    for key, value in sides.items():
        plane = MatrixPlane(value)
        coefficients[key] = plane.get_determinant()

    self.matrix_body = MatrixBody(coefficients)
    self.matrix_body.adjust(self.matrix_center)


def set_visible_sides(self):
    self.set_matrix_body()
    result = self.matrix_body.multiplication_vector(self.viewer)
    sides = self.matrix_body.sides
    self.visible_sides = [side for side, value in zip(sides, result) if value >
EPS]
```

Листинг 3 – Алгоритм отрисовки поворачиваемой грани

```
def draw_turning(self, painter: QtDrawer, side: str, plastic_part: list[Point])
-> None:
    def draw_below_turning(shadows_=None):
        self.ribs.draw_below_turning(painter, self.visible_sides, side,
shadows_)
        self.centers.draw(painter, self.visible_sides, shadows_)
        self.corners.draw_below_turning(painter, self.visible_sides, side,
shadows_)

    def draw_static_plastic_part():
        painter.setBrush(QBrush(QColor('black'), Qt.SolidPattern))
        painter.fill(plastic_part)

    pen = QPen(Qt.black, 6)
    painter.setPen(pen)
    shadows = None if not self.light_sources else self.count_shadows()

    if side in self.visible_sides:
        draw_below_turning(shadows)
        draw_static_plastic_part()
        self.artist(painter, side)
    else:
        self.artist(painter, side)
        draw_below_turning(shadows)

def artist(self, painter: QtDrawer, side: str) -> None:
    corners = self.corners.get_centers(side)
    ribs = self.ribs.get_centers(side)
    centers = self.centers.get_centers(side)

    eccentric = corners | ribs | centers
    details = sorted(eccentric, key=eccentric.get)

    for detail in details:
        detail.draw_turning(painter, self.visible_sides, side,
self.matrix_center[:-1], self.light_sources)
```

Листинг 4 – Алгоритм поворота грани (часть 1)

```
def turn_side_elements(self, name, angle, alpha, beta):
    self.turn_oz_funcs(alpha.sin, alpha.cos)
    self.turn_ox_funcs(beta.sin, beta.cos)

    self.corners.turn_side_oy(name, angle)
    self.ribs.turn_side_oy(name, angle)
    self.centers.turn_side_oy(name, angle)

    self.turn_ox_funcs(-beta.sin, beta.cos)
    self.turn_oz_funcs(-alpha.sin, alpha.cos)
```

Листинг 5 – Алгоритм поворота грани (часть 2)

```
def turn_side(self, name, angle):
    self.move(-self.center_point)

    direction_vector = Vector(Point(0, 0, 0), self.centers.sides_centers[name])
    direction_vector.normalize()
    d = direction_vector.get_length_xy()

    alpha = Angle()  # to yz plane
    beta = Angle()  # to y
    try:
        alpha.set_cos(direction_vector.y / d)
        alpha.set_sin(direction_vector.x / d)
    except ZeroDivisionError:
        alpha.set_cos(1)
        alpha.set_sin(0)
    beta.set_cos(d)
    beta.set_sin(-direction_vector.z)

    self.turn_side_elements(name, angle, alpha, beta)

    self.move(self.center_point)
```

Листинг 6 – Алгоритм нахождения интенсивности света

```
def count_shadows(self) -> dict[str, float | None]:
    shadows = {}
    for side in self.visible_sides:
        shadows[side] = self.get_shadow(side)

    return shadows

def get_shadow(self, position_side: str) -> float | None:
    if not self.light_sources:
        return None

    side = self.cfg.get_sides()[position_side]
    center = self.centers.sides_centers[position_side]

    plane_points = [self.corners.carcass[key] for key in side]
    normal = Vector(MatrixPlane(plane_points).get_determinant()[:-1])
    normal.adjust(center, Point(*self.matrix_center[:-1]))

    cosine = 0
    for light in self.light_sources:
        cosine += get_plane_cosine(light, center, normal)

    if cosine <= 1:
        return cosine
    else:
        return 1
```

3.3 Тестирование программного продукта

На рисунках 23-24 представлен интерфейс разработанного программного обеспечения с кубиком и пирамидкой соответственно.

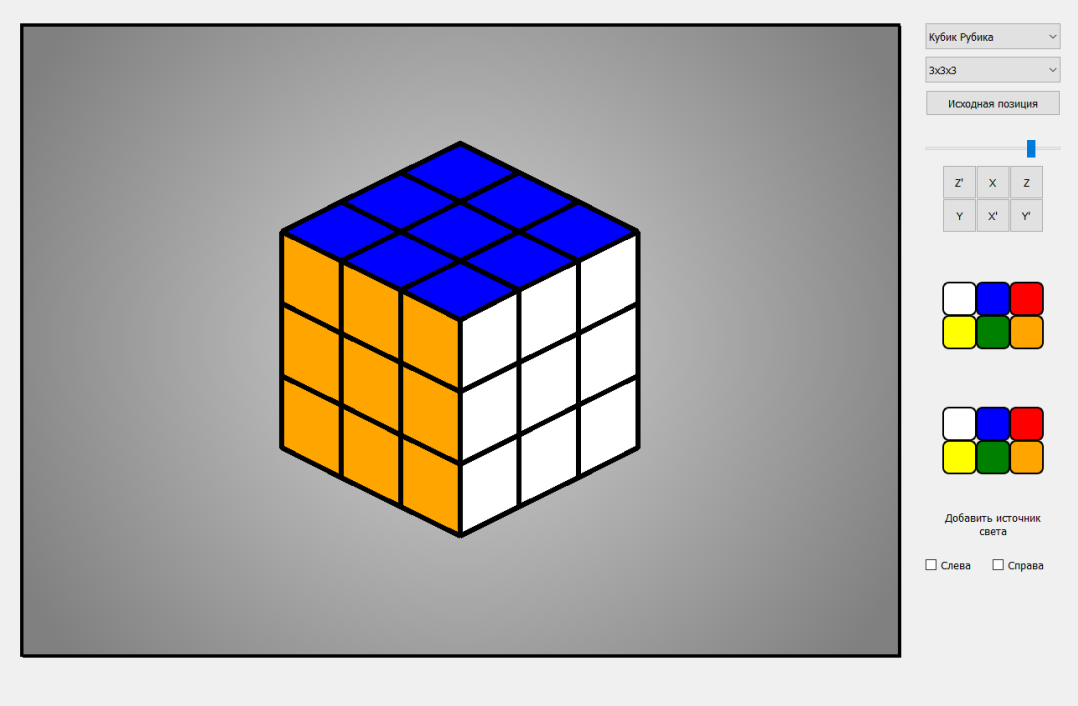

Рисунок 23 – Интерфейс разработанного приложения (кубик Рубика)

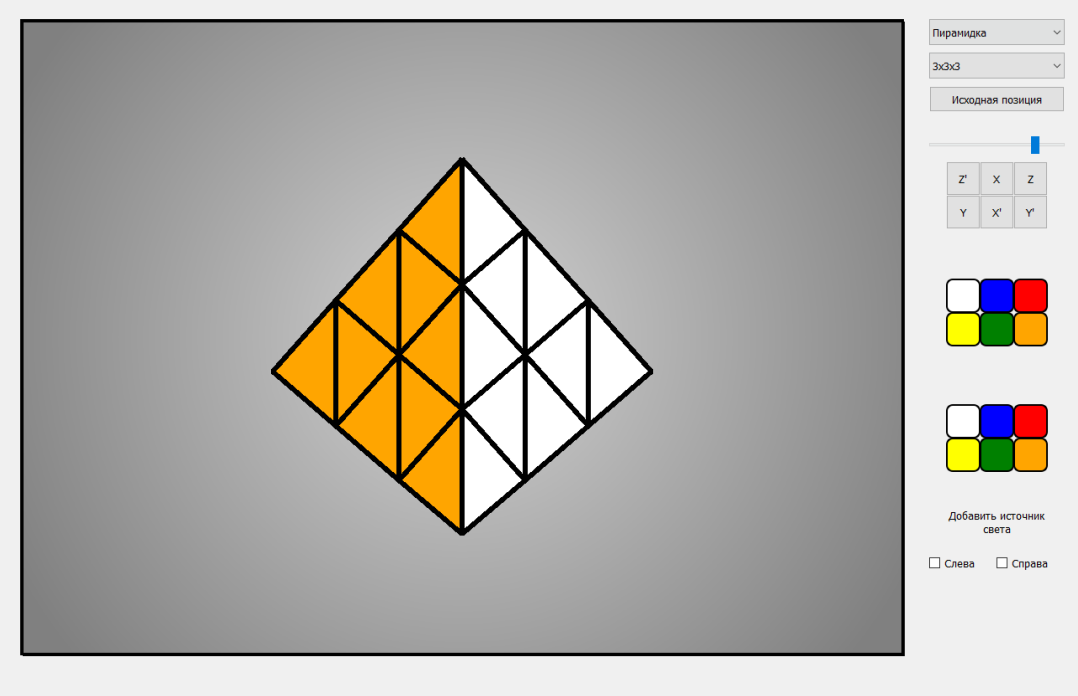

Рисунок 24 – Интерфейс разработанного приложения (пирамидка Мефферта)

Пользователь может выбирать головоломку, в случае кубика Рубика – задавать ее размер, масштабировать, поворачивать модель кнопками, мышкой, поворачивать грани при помощи кнопок, добавлять источники света с левой и правой стороны.

Для определения корректности работы предложенных функций необходимо провести тестирование. Так как и кубик, и пирамидка наследуются от одного класса, то тестирование можно провести только для одной головоломки, например, для кубика Рубика.

1. На рисунках 25-27 представлены повороты вокруг осей абсцисс, ординат и аппликат соответственно.

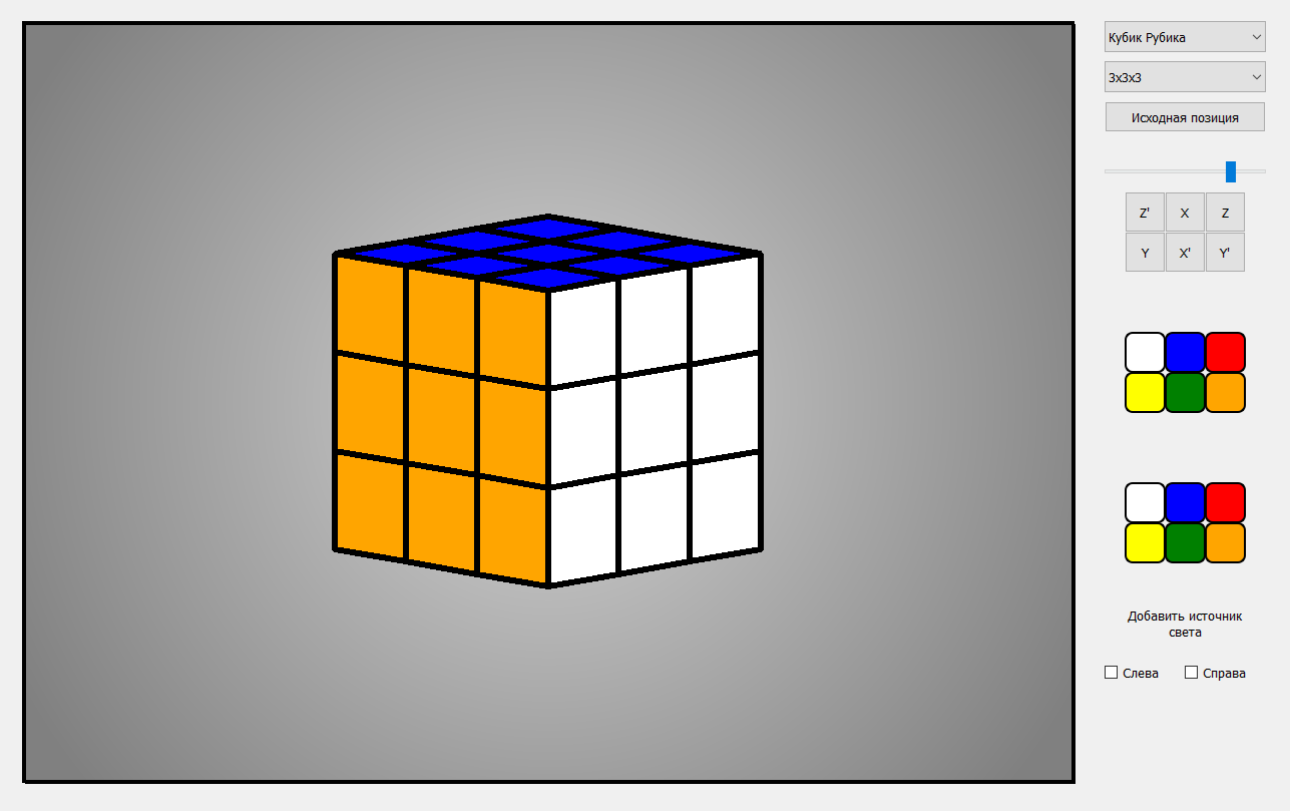

Рисунок 25 – Поворот вокруг оси абсцисс

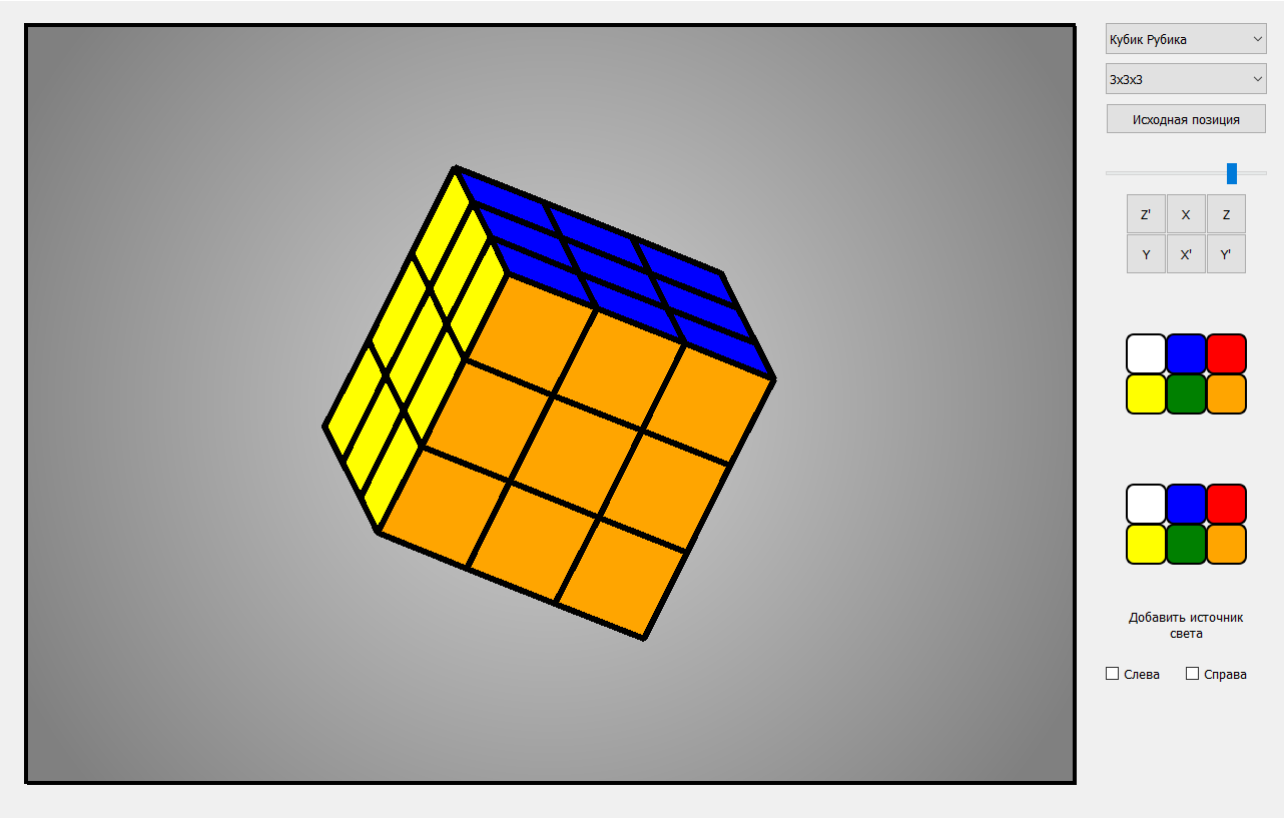

Рисунок 26 – Поворот вокруг оси ординат

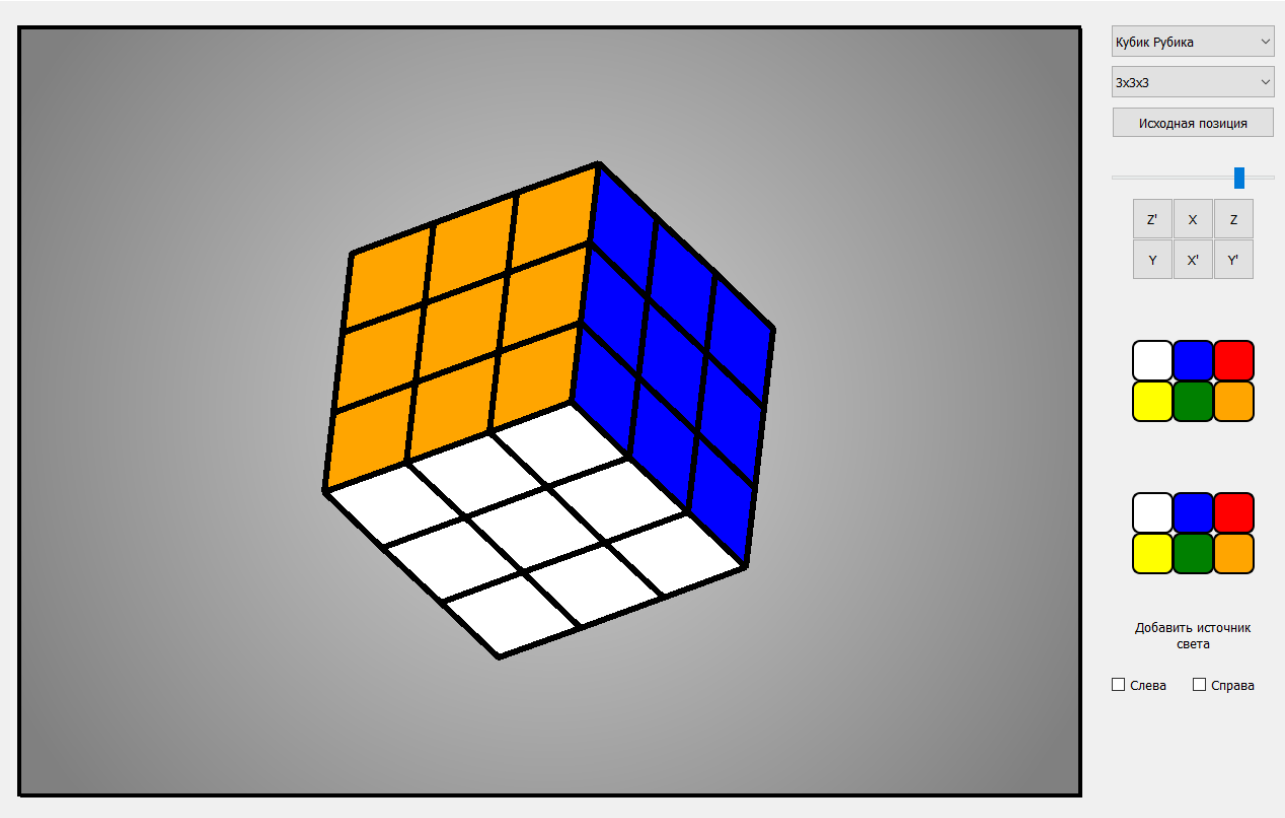

Рисунок 27 – Поворот вокруг оси аппликат

2. На рисунке 28 показан результат масштабирования.

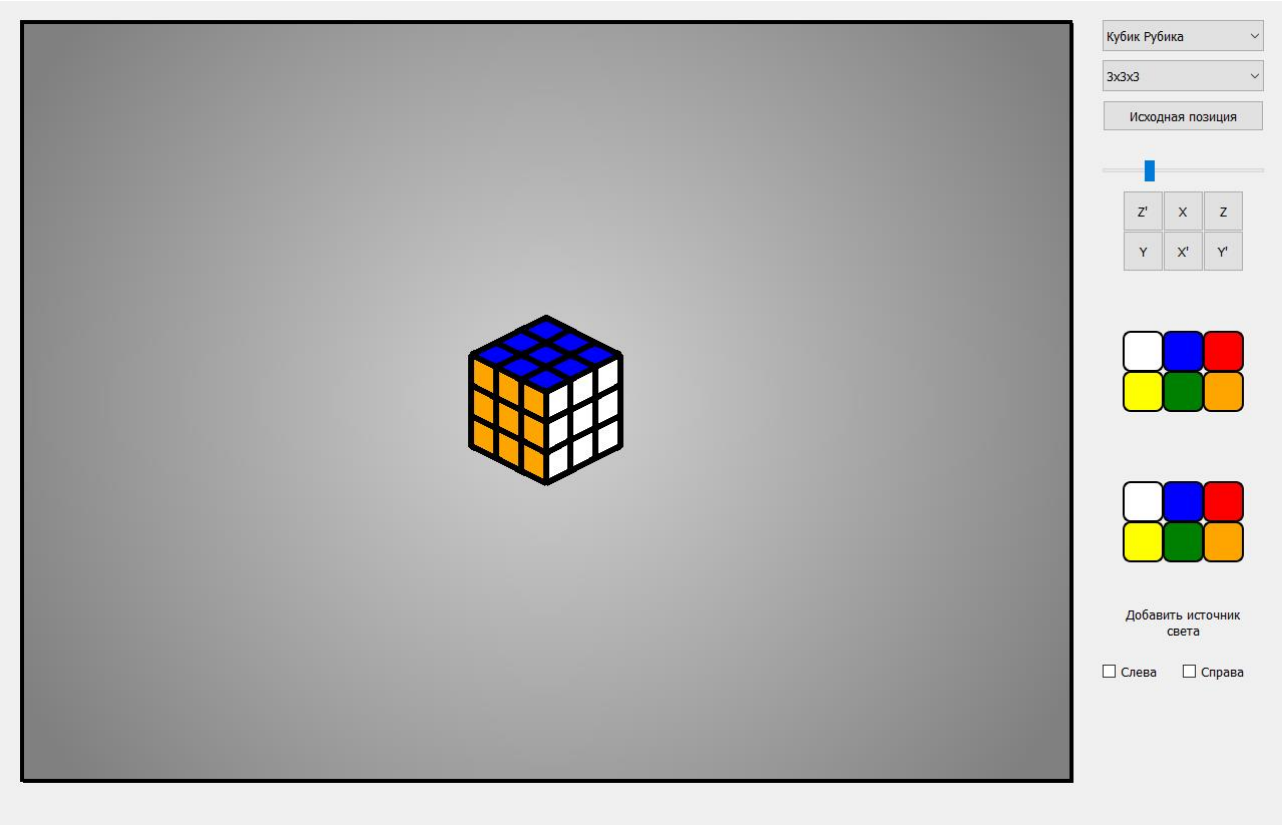

Рисунок 28 – Масштабирование

3. На рисунке 29 показан результат поворота верхней грани.

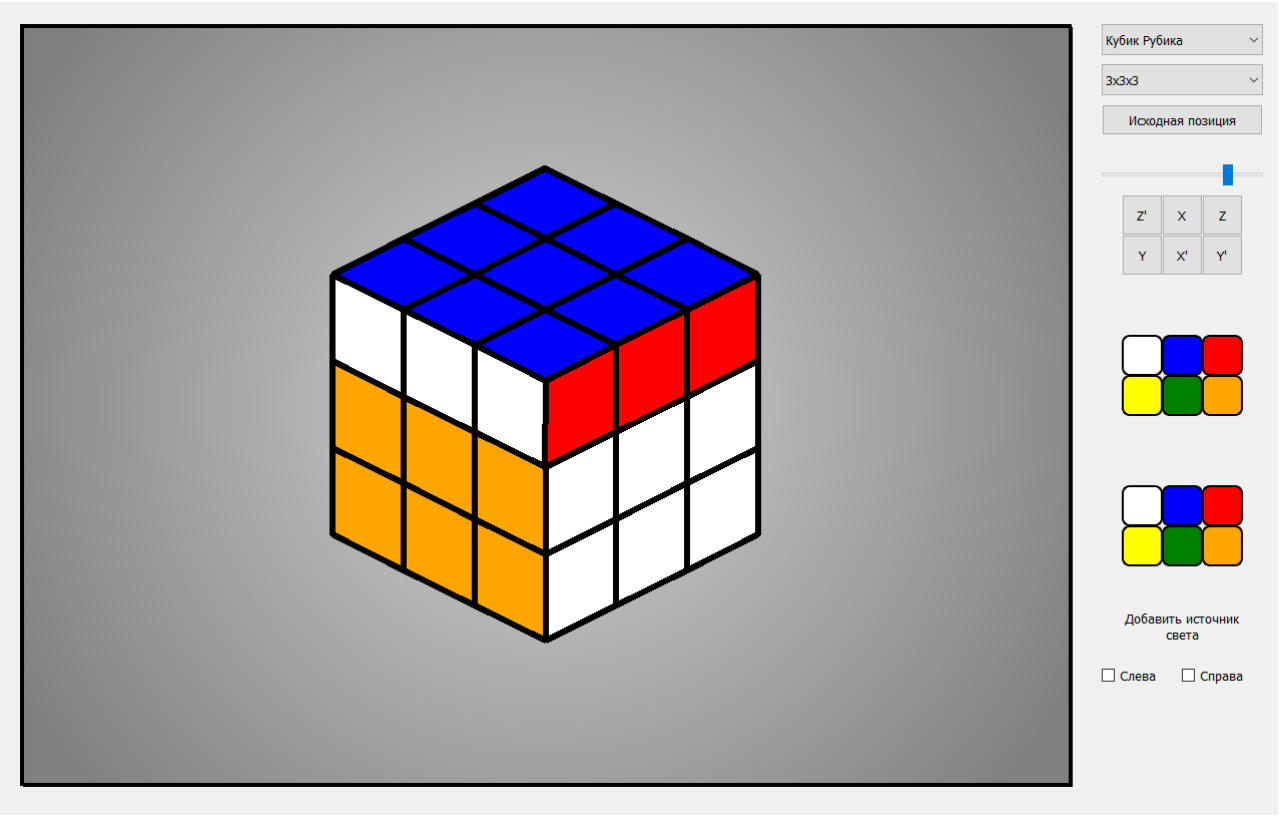

Рисунок 29 – Результат поворота грани

4. На рисунке 30-31 представлены результаты освещения с одним и двумя источниками соответственно.

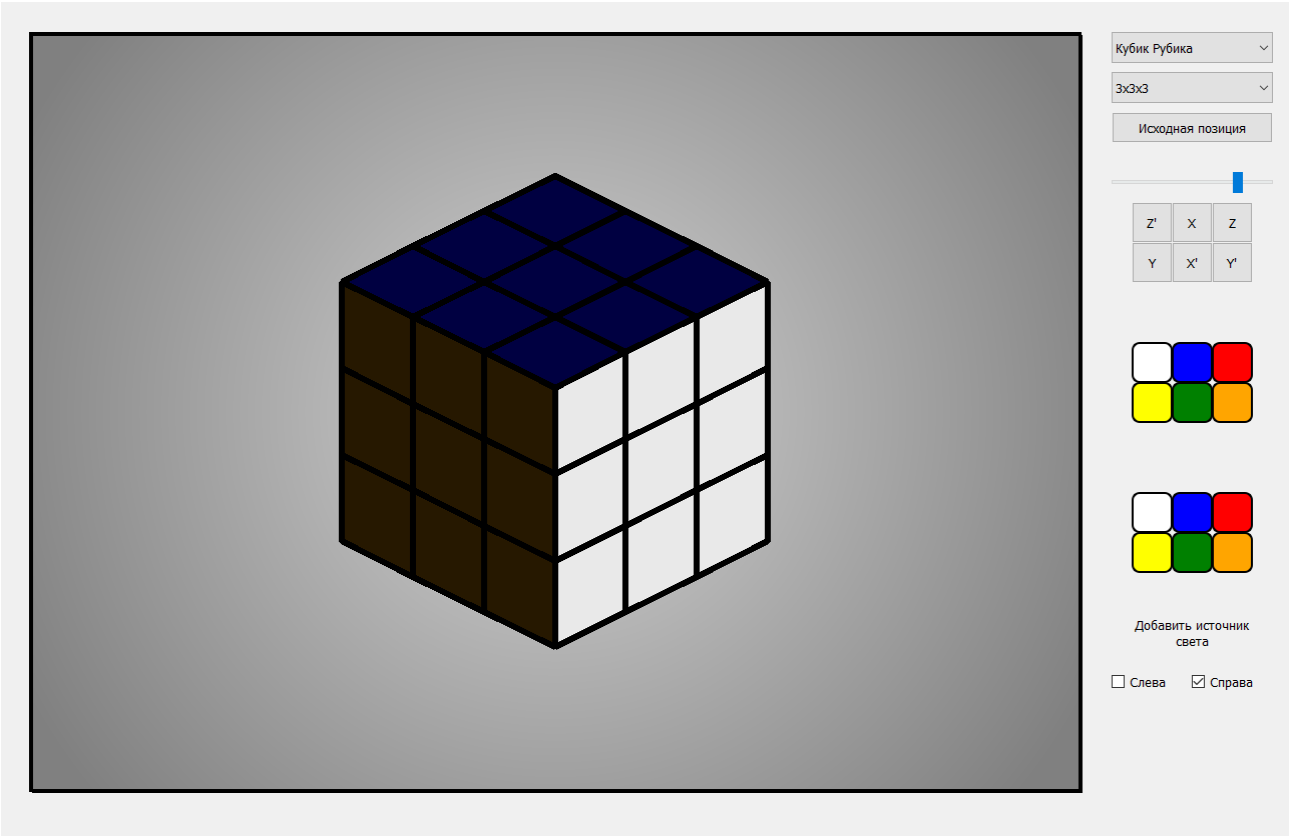

Рисунок 30 – Освещение с одним источником света

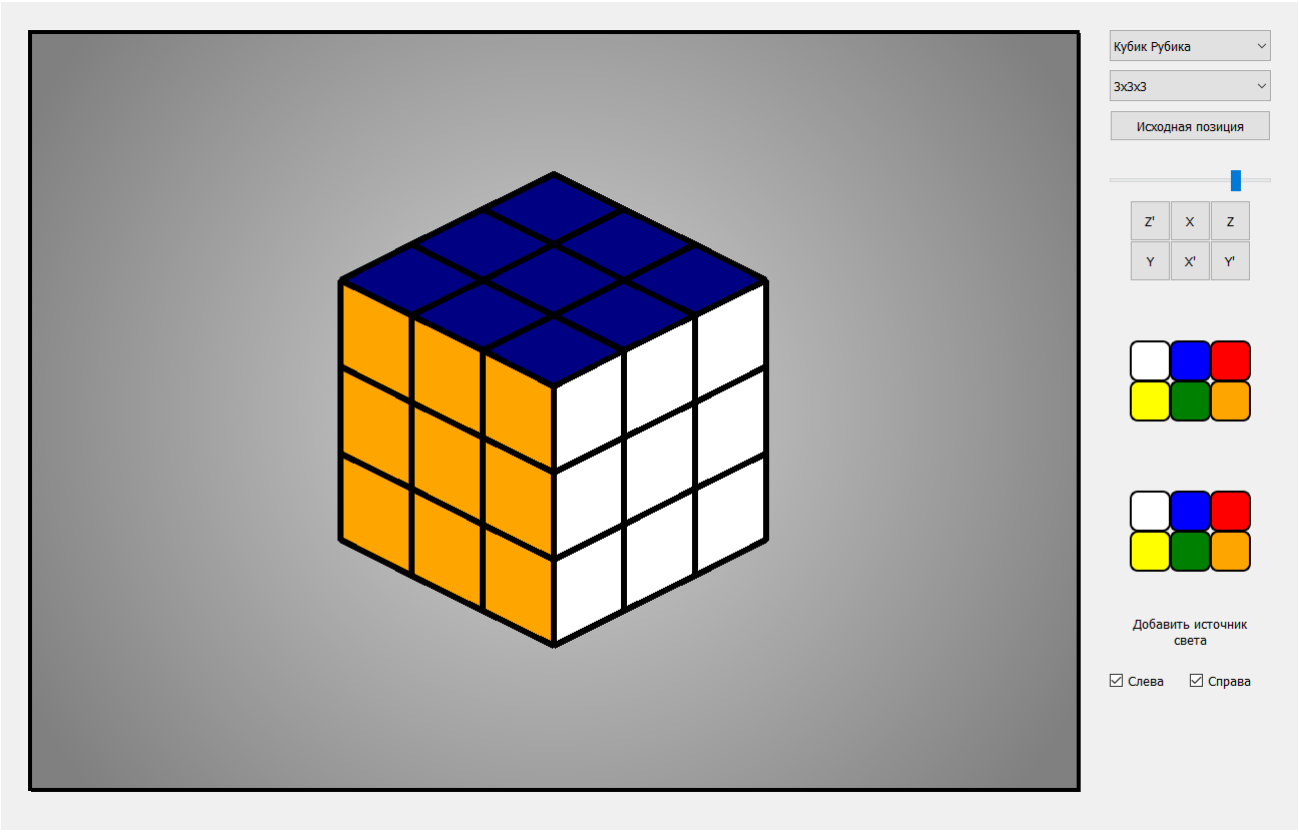

Рисунок 31 – Освещение с двумя источниками света

5. На рисунке 32 показана другая размерность кубика Рубика (5x5).

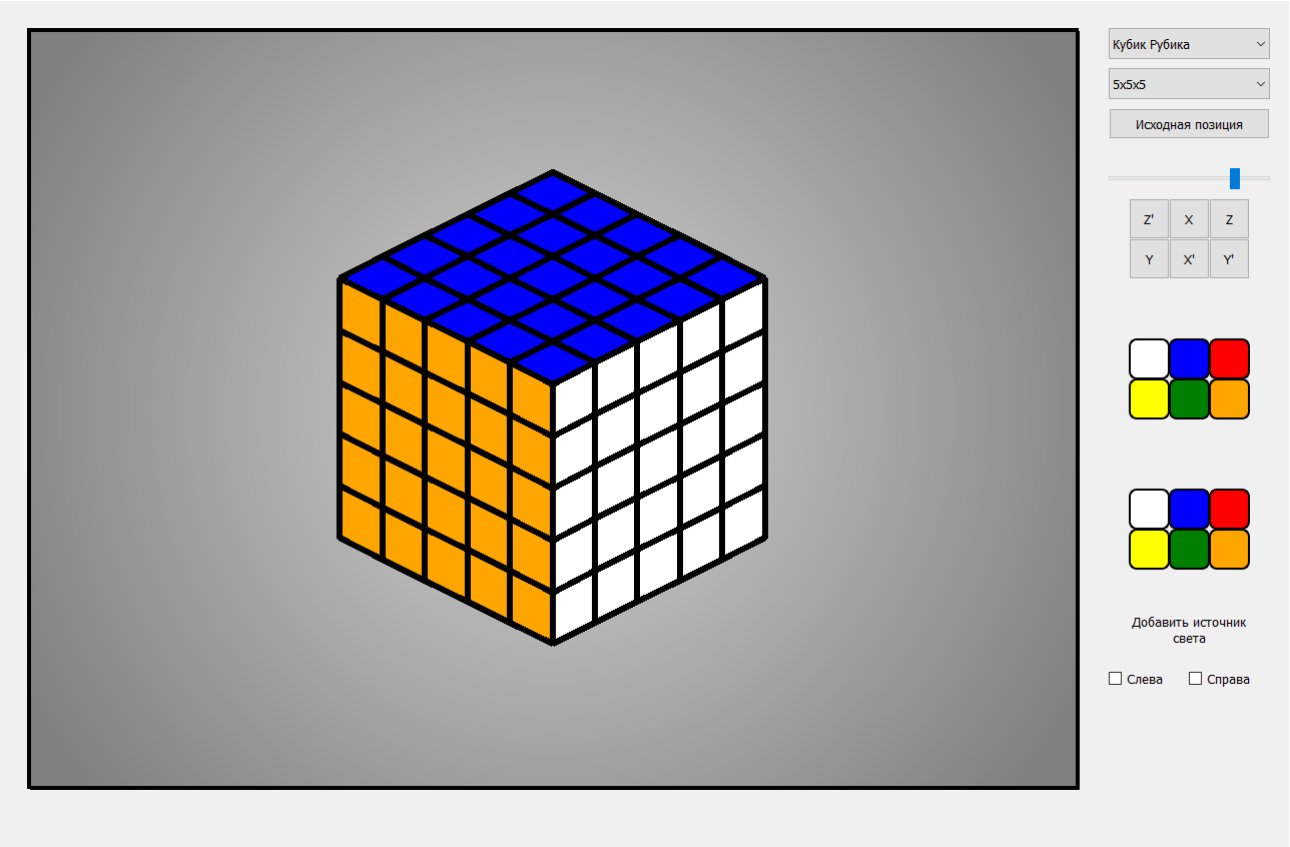

Рисунок 32 – Размерность кубика 5x5

3.6 Вывод

Таким образом, было реализованы рассматриваемые алгоритмы, разработано программного обеспечение, которое позволит пользователю взаимодействовать с головоломкой: вращать, масштабировать, задавать размерность, поворачивать грани, задавать до двух источников света.

4 Исследовательская часть

В данном разделе приводятся примеры работы программы, проведен сравнительный анализ времени поворота грани кубика Рубика на разных размерностях при заданных источниках света и без и сравнение с временем поворота грани пирамидки Мефферта.

4.1 Примеры работы программы

На рисунках 33-36 представлены примеры работы программы во время поворота граней.

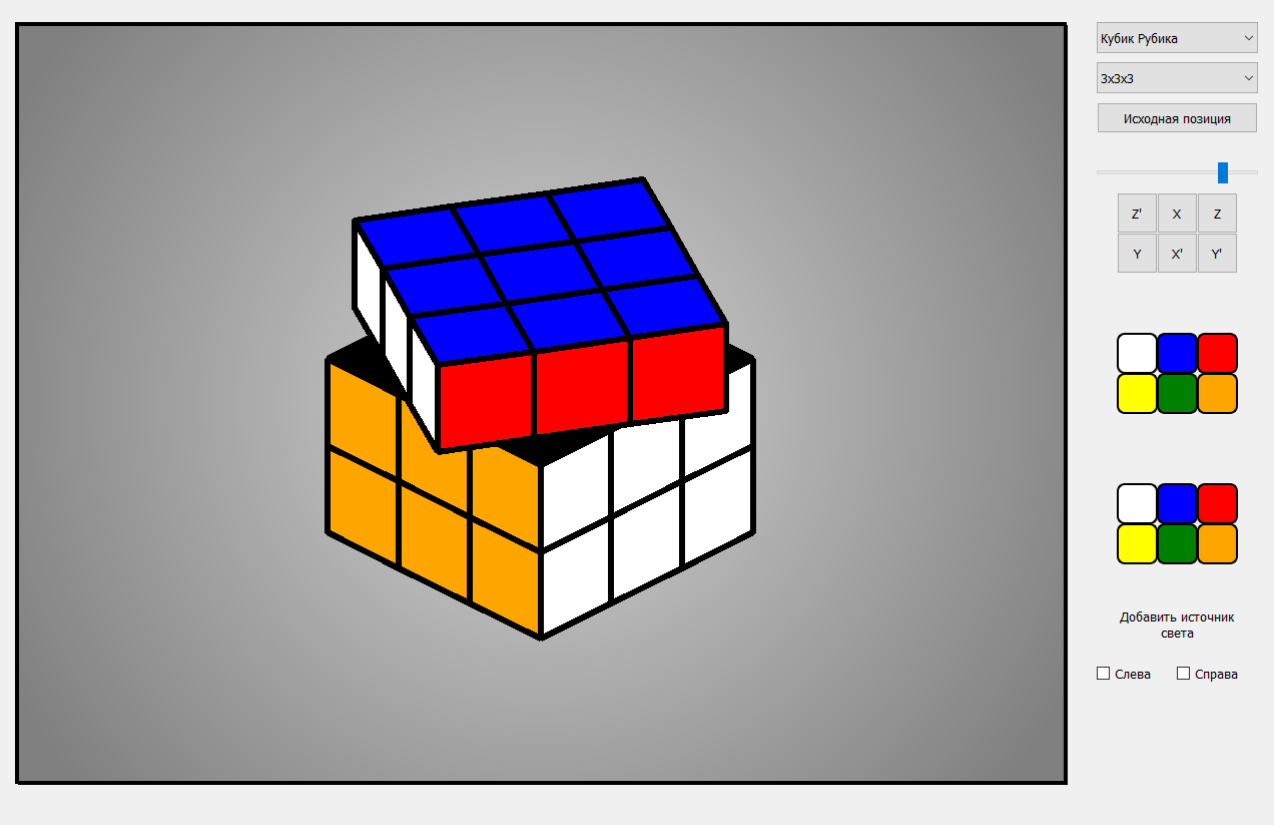

Рисунок 33 – Пример работы программы (часть 1)

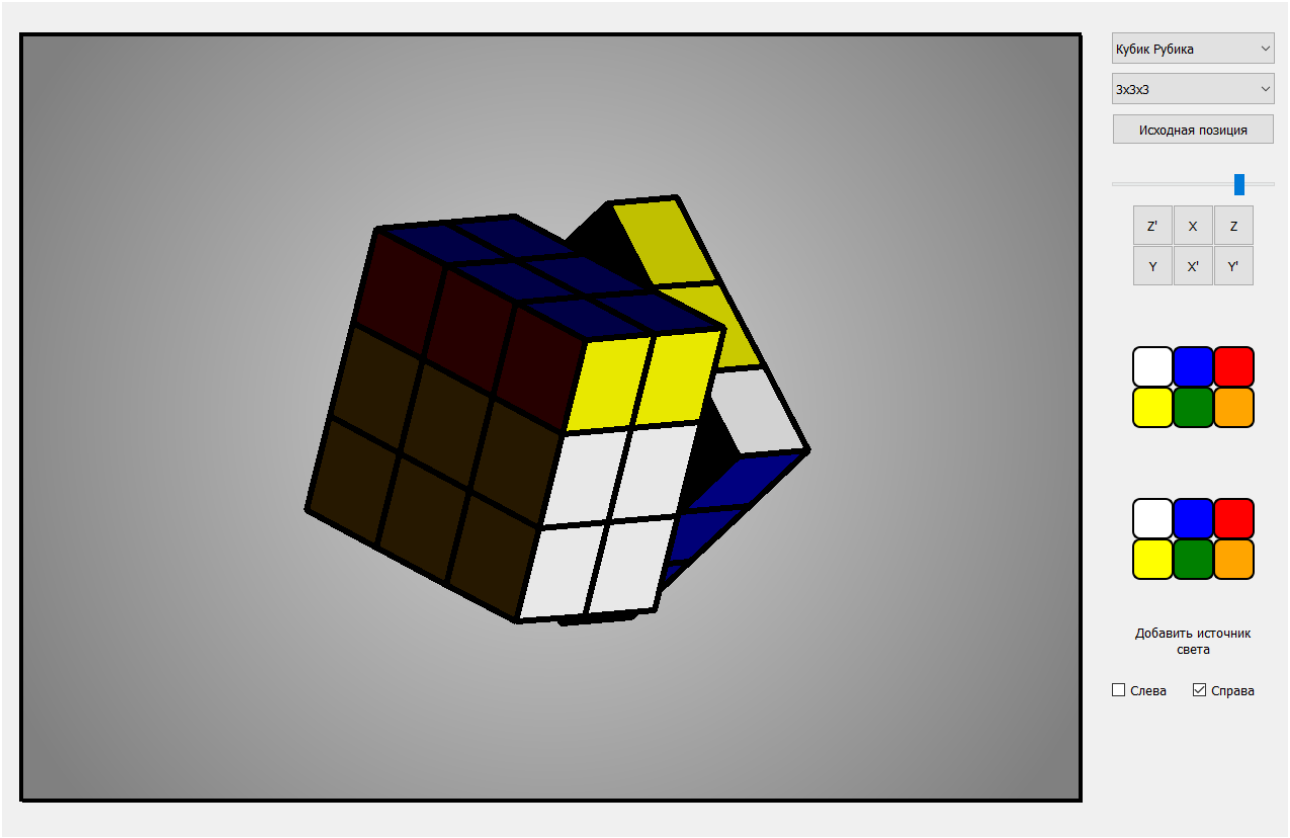

Рисунок 34 – Пример работы программы (часть 2)

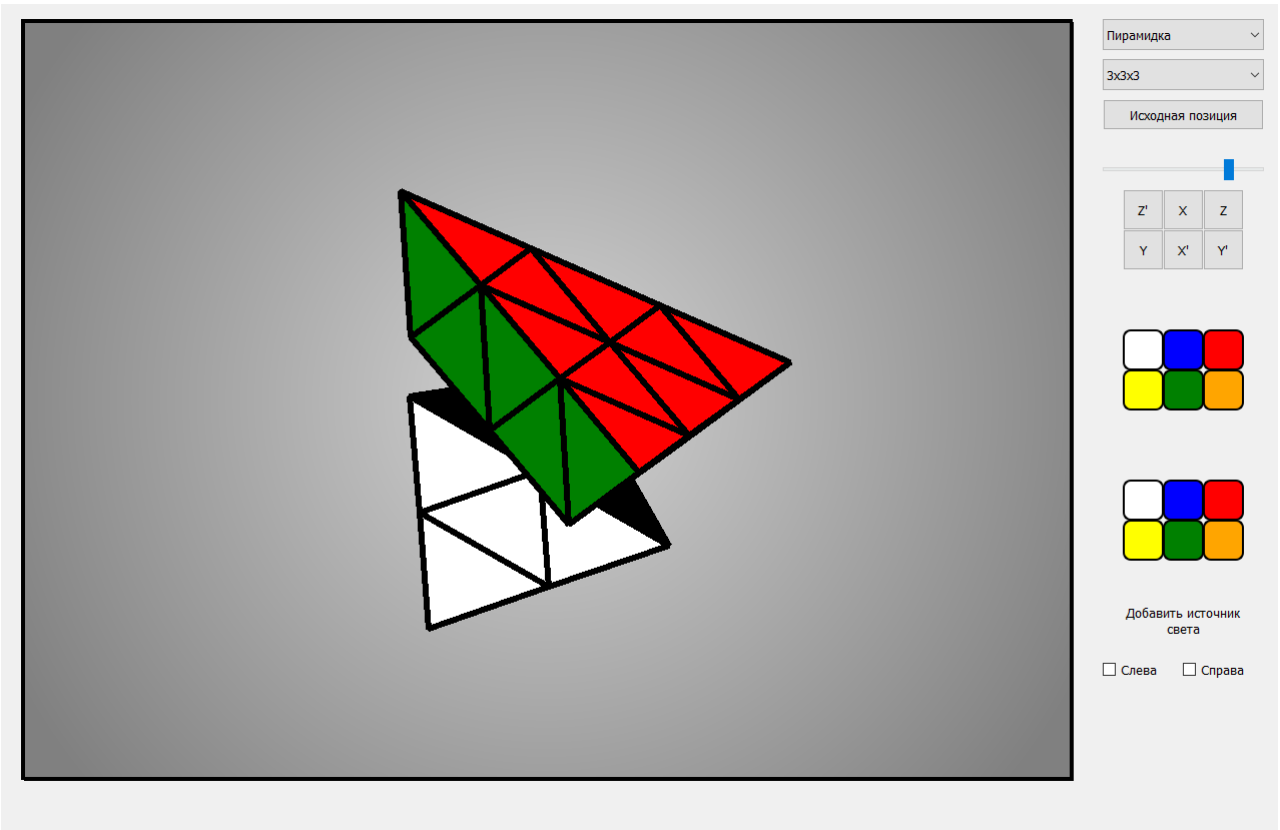

Рисунок 35 – Пример работы программы (часть 3)

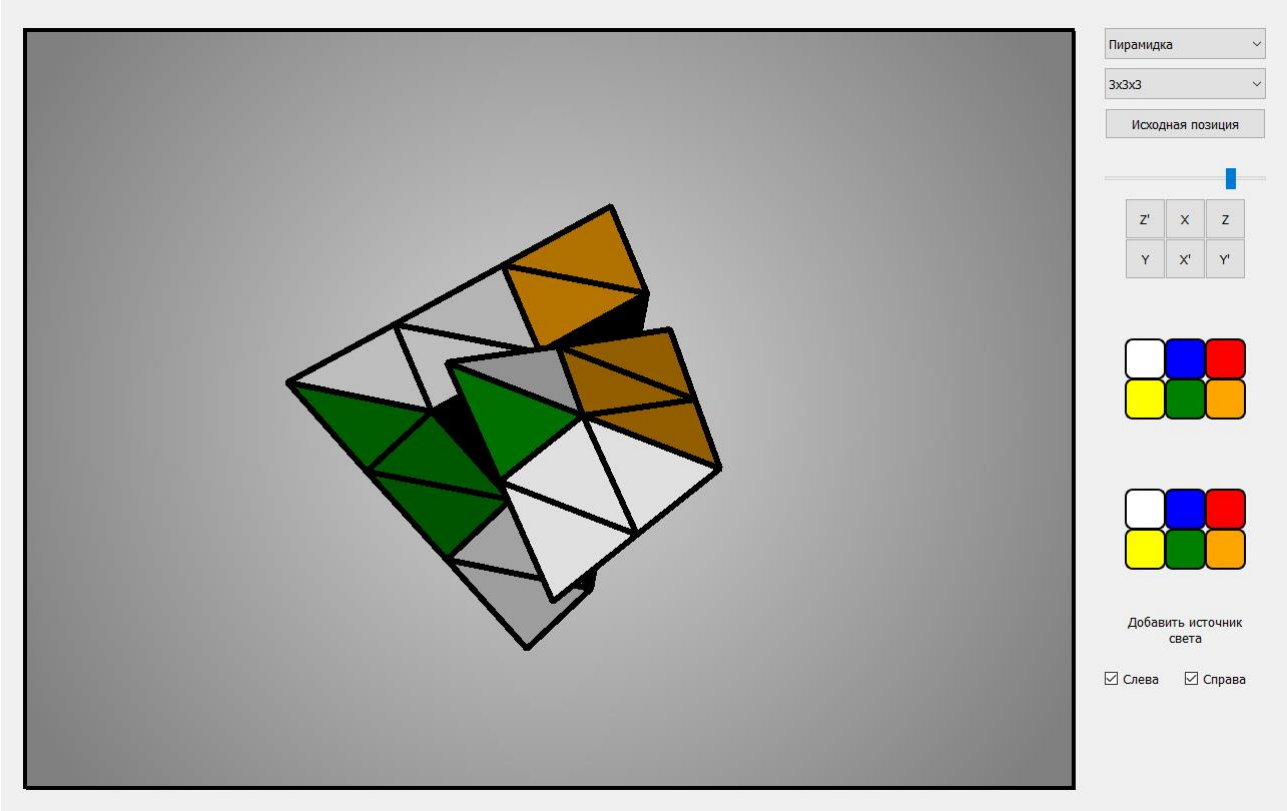

Рисунок 36 – Пример работы программы (часть 4)

4.2 Технические характеристики устройства

При проведении экспериментов необходимо учитывать, на каком устройстве они проводятся, ниже представлены технические характеристики:

- операционная система: Windows 10 (64-разрядная);
- оперативная память: 32 GB;
- процессор: Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz;
- количество ядер: 4;
- количество потоков: 8.

4.3 Сравнительный анализ времени работы

Для исследования времени работы была проведена серия экспериментов и получено время поворота грани кубика в зависимости от размерности и наличия освещения (таблица 1). Дополнительно проведено сравнение кубика и пирамидки с такими же условиями (таблица 2). Время приведено в секундах.

Таблица 1 – Сравнительная таблица времени поворота грани кубика Рубика на разных размерностях

Размерность	**Без освещения**	**С освещением**
2x2x2	0.297	0.328
3x3x3	0.344	0.390
4x4x4	0.375	0.453
5x5x5	0.484	0.563
6x6x6	0.578	0.688
7x7x7	0.703	0.781
8x8x8	0.797	0.922

Таблица 2 – Сравнительная таблица времени поворота грани кубика Рубика и пирамидки Мефферта

Головоломка	**Без освещения**	**С освещением**
Кубик Рубика	0.359	0.375
Пирамидка Мефферта	0.438	0.469

4.4 Вывод

В результате проведенных экспериментов видно, что время поворота грани кубика Рубика растёт пропорционально его размерности (на 15-20%). Освещение замедляет скорость работы приблизительно на 20%. На больших размерностях время близко к одной секунде, что видно визуально. Поворот

грани пирамидки Мефферта медленнее на 22%, так как несмотря на меньшее количество деталей, угол поворота на 30 градусов больше. Наличие освещение делает скорость медленнее приблизительно на 10%.

Заключение

В результате выполнения работы была достигнута поставленная цель – создан программный продукт, визуализирующий «Кубик Рубика», «Пирамидку Мефферта» и позволяющий собирать их. Также были решены все поставленные задачи:

1) определены требования для приложения на основе анализа аналогичных;
2) проведен анализ алгоритмов удаления невидимых поверхностей и выбран подходящий;
3) определен способ поворота грани;
4) выбран наиболее подходящий метод окрашивания граней;
5) составлены диаграммы классов, схемы алгоритмов;
6) разработаны выбранные алгоритмы;
7) проведен сравнительный анализ скорости работы приложения на разных режимах.

Список использованных источников

1) Википедия. Свободная энциклопедия [Электронный ресурс]: Кубик Рубика. – Режим доступа: https://ru.wikipedia.org/wiki/Кубик_Рубика (дата обращения 3.07.2021).
2) А.Ю. Дёмин, А.В. Кудинов, «Компьютерная графика» [Электронный ресурс]: глава «Удаление невидимых линий и поверхностей». – Режим доступа: http://compgraph.tpu.ru/Del_hide_line.htm (дата обращения 6.07.2021).
3) Польский С.В. «Компьютерная графика» [Электронный ресурс]: учебно-методическое пособие. – Издательство Московского государственного университета леса, 2008.
4) А.Ю. Дёмин, А.В. Кудинов, «Компьютерная графика» [Электронный ресурс]: глава «Алгоритм Робертса». – Режим доступа: http://compgraph.tpu.ru/roberts.htm (дата обращения 7.07.2021).
5) Алгоритм Художника [Электронный ресурс]. – Режим доступа: https://studfile.net/preview/954959/page:8/ (дата обращения 14.07.2021).
6) А.Ю. Дёмин, А.В. Кудинов, «Компьютерная графика» [Электронный ресурс]: глава «Алгоритм Z-буфера». – Режим доступа: http://compgraph.tpu.ru/zbuffer.htm (дата обращения 7.07.2021).
7) Д.Роджерс, «Алгоритмические основы машинной графики» [Электронный ресурс]: электронная книга. – перевод на русский язык, с изменениями, «Мир», 1989.
8) Д.Роджерс, «Алгоритмические основы машинной графики» [Электронный ресурс]: глава «Поворот вокруг произвольной оси в пространстве». – Режим доступа: https://scask.ru/a_book_mm3d.php?id=60 (дата обращения 13.07.2021).
9) Межотраслевая Интернет-система поиска и синтеза физических принципов действия преобразователей энергии [Электронный ресурс]: закон Ламберта. – Режим доступа:

http://www.heuristic.su/effects/catalog/est/byId/description/243/index.html (дата обращения 16.07.2021).