**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 (ИУ)**

КАФЕДРА **Программное обеспечение ЭВМ и информационные технологии (ИУ7)**

# ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент Климов Илья Сергеевич
*фамилия, имя, отчество*

Группа ИУ7-42Б

Тип практики Технологическая

Название предприятия НУК ИУ МГТУ им. Н.Э. Баумана

Студент _____ Климов И.С.
*подпись, дата* *фамилия, и.о.*

Руководитель практики _____ Куров А.В.
*подпись, дата* *фамилия, и.о.*

Оценка _____

*2021 г.*

# Оглавление

## Введение

В 1974 году венгерский скульптор Эрнё Рубик изобрел головоломку, которая впоследствии стала одной из самых популярных в мире, и, наверное, нет такого человека, который бы не слышал про нее. Головоломка носит название в честь её изобретателя – кубик Рубика и представляет собой пластмассовый куб с подвижными деталями.

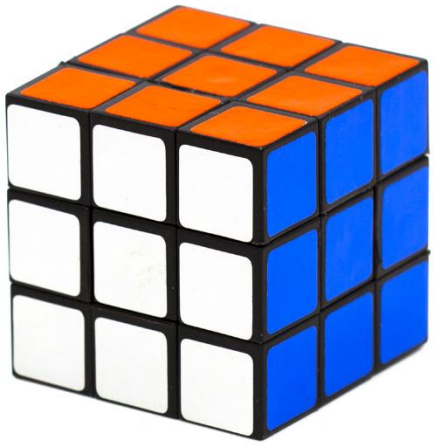

*Рис.1 – Кубик Рубика в собранном состоянии.*

Изначально размерность составляла только 3x3x3, то есть в таком случае кубик состоит из крестовины с 6-ю центральными элементами, 8 углов и 12 реберных элементов. Затем начали появляться различные модификации, включающие в себя не только другую размерность, но и другую форму с отличающимся внутренним строением. Одними из самых популярных являются «пирамидка Мефферта» и «мегаминкс» (головоломка в форме додекаэдра). Они имеют схожий принцип с кубиком, но разное количество граней и деталей.

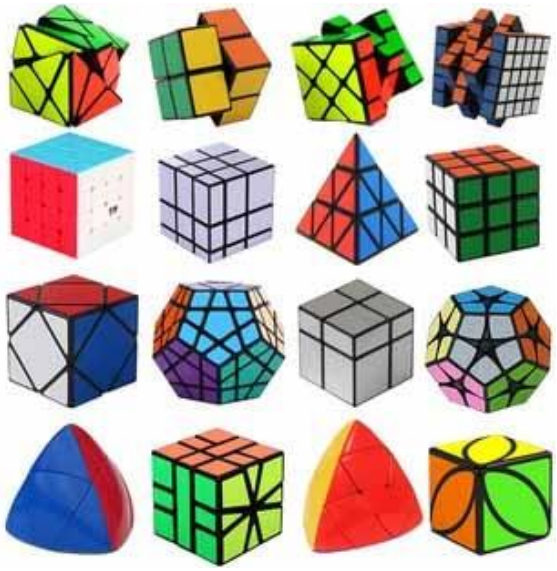

*Рис.2 – Разновидности головоломок.*

*Рис.3 – Пирамидка Мефферта в собранном состоянии.*

*Рис.4 – Мегаминкс в собранном состоянии.*

Задача решения кубика состоит в его сборке в исходное положение путем поворота граней. Сложность обуславливается тем, что число всех достижимых состояний составляет приблизительно 43 квинтиллиона ($43 * 10^{18}$). Занимательно, что из любого положения можно привести к единственному собранному всего за 20 ходов. На сегодняшний день проводится множество соревнований по различным дисциплинам. 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 визуализацией этой головоломки и возможностью ее сборки.

**Цель работы** – создание программного продукта, визуализирующего головоломки «кубик Рубика», «пирамидка Мефферта», «мигаминкс» и позволяющего собирать их. Для достижения цели были выделены следующие **задачи**:

1) определить требования для приложения 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) описать модель головоломки;
6) разработать выбранные алгоритмы;
7) реализовать часть алгоритмов и интерфейса программно.

# 1. Аналитическая часть

В данном разделе рассматриваются существующие программы по заданной теме, анализируются алгоритмы удаления невидимых поверхностей, закрашивания, поворота грани 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»

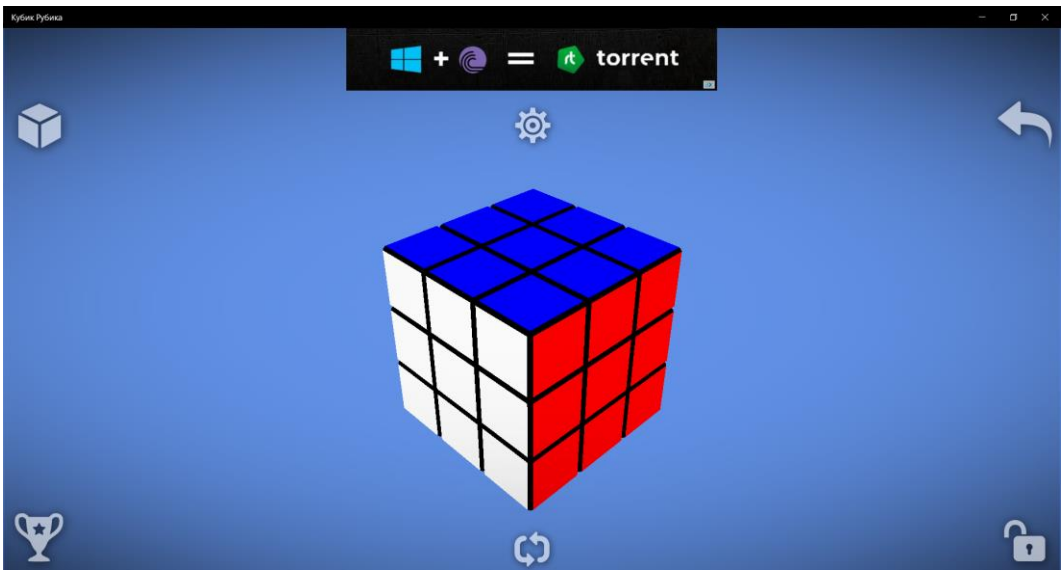


*Рис.5 – Экран приложения «Кубик Рубика»*

**Достоинства:**

- возможность выбора головоломки (кубик Рубика, пирамидка Мефферта, мегаминкс) и её размерности;
- возможность масштабирования;
- возможность вращения головоломки в разных плоскостях;

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

- возможность вернуть предыдущее состояние.

**Недостатки:**

- грани поворачиваются быстро, без контроля;
- отсутствие возможности кастомизации;
- отсутствие кнопок для управления головоломкой.

### 1.1.2. Приложение «Free Super Cube»

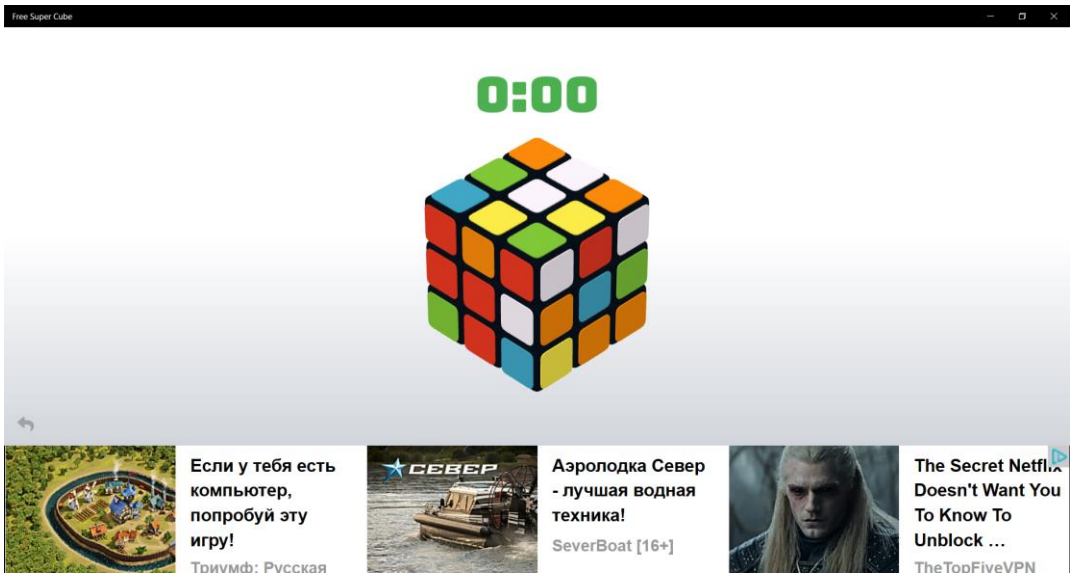


*Рис.6 – Экран приложения «Free Super Cube»*

**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»

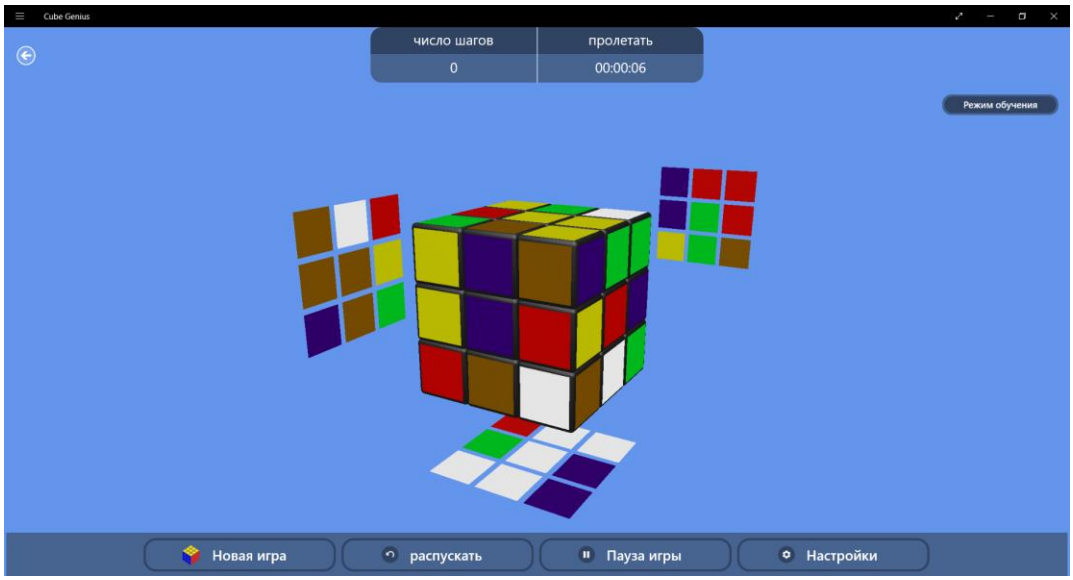


*Рис.7 – Экран приложения «Cube Genius»*

**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.

- плавный поворот граней мышкой, возможность контроля;
- возможность выбора головоломки (кубик, пирамидка, мегаминкс) 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:**

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

**Недостатки:**

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

**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. 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 головоломки

Суть решения головоломки состоит в её переводе в изначальное положение путем поворота граней. Для этого 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 длина равна:

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

А косинус и синус угла поворота равны:

$$\cos(\alpha) = \frac{c_y}{d}; \sin(\alpha) = \frac{c_x}{d}$$

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

$$\cos(\beta) = d; \sin(\beta) = c_x$$

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:

$$I = I_0 \cos(\alpha)$$

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.

**Вывод по аналитической части:** в ходе анализа аналогичных приложений, алгоритмов удаления невидимых поверхностей, алгоритма поворота вокруг произвольной оси, методов закрашивания сделан выбор, который позволил:

1) определить требования будущей программы:
    - плавный поворот граней мышкой, возможность контроля;
    - возможность выбора головоломки (кубик, пирамидка, мегаминкс) 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;
2) выбрать 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) решить проблему поворота каждой грани;
4) решить проблему закраски граней – простой метод закрашивания.

# 2. Конструкторская часть

В данном разделе предоставлены описание формы модели, её элементов, описание выбранных алгоритмов, на основе которых в дальнейшем будет реализована программа. В ходе разработки будет использоваться технология 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. Описание модели

Программа предполагает наличие возможности выбора головоломки из трех. По процессу сборки, по логике вращения они схожи: по сути, все они вращаются в одной из трех плоскостей, и каждый раз происходит поворот грани вокруг определенной точки, поэтому все классы головоломки можно наследовать от одного класса «Model».

Класс «Model» имеет следующие основные поля (для реализации некоторых алгоритмов возможно расширение их количества):

- **n** – размерность головоломки;
- **corners** – угловые элементы головоломки;
- **ribs** – реберные элементы головоломки;
- **centers** – центральные элементы головоломки;
- **k** – текущий коэффициент масштабирования.

В свою очередь **corners**, **ribs**, **centers** являются объектами классов «Corners», «Ribs», «Centers» соответственно. Они содержат в себе массивы самих деталей – объекты классов «Corner», «Rib», «Center», наследуемые от класса «Detail».

Класс «Detail» включает в себя следующие поля:

- **vertices** – вершины детали;
- **edges** – ребра детали;
- **name** – имя детали.

Реберные и угловые элементы представляют собой объемные объекты, так как при повороте граней можно увидеть и те части элементов, которые не покрыты цветными наклейками. Чего не скажешь про центральные элементы, которые можно представить в виде плоских фигур.

Довольно очевидно, что у каждой из выбранных головоломок фиксированное количество углов, несмотря на размерность, в случае ребер и центров – их количество разнится.

Также хочется отметить, что каждый из перечисленных классов имеет часть метолов, имеющих одну цель:

- **draw** – отрисовка;
- **scale** – выполнение масштабирования;
- **move** – перемещение;
- **turn_ox** – поворот вокруг оси абсцисс;
- **turn_oy** – поворот вокруг оси ординат;
- **turn_oz** – поворот вокруг оси аппликат;
- **update_sides** – обновление элементов (присваивание новых имен) после поворота грани.

Также точка и ребро оформлены в класс со своими полями и методами. Методы класса «Point» позволяют производить трансформацию каждой точки., к полям относятся координаты x, y, z. У класса «Edge» к полям относятся first (начало ребра) и second (конец ребра), к методам – получение этих точек в виде списка.

Помимо этого, имеются классы матриц: матрица тела («MatrixBody») и матрица плоскости («MatrixPlane»). Последняя представляет собой поле массива, которые содержит в себе три массива, представляющий точки в каждой строке матрицы. К методам можно отнести получение определителя матрицы. Матрица тела же содержит в себе массив из массивов, хранящих в себе коэффициенты уравнений плоскостей, и имена каждой плоскости. Класс

позволяет производить умножение матриц и векторов и проверять правильность расстановки коэффициентов.

## 2.2. Разработка алгоритмов

Перед реализацией алгоритмов программно необходимо понять из каких этапов каждый из них состоит, привести описание.

### 2.2.1.Алгоритм удаления невидимых поверхностей

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

#### 2.2.1.1. Удаление нелицевых граней

Словесное описание алгоритма.

1. Начало алгоритма;
2. Создать матрицу тела:
    2.1. Получить по три точки для каждой плоскости, составляющей фигуру;
    2.2. Для каждой плоскости:
        2.2.1. Создать объект класса «MatrixPlane»;
        2.2.2. Посчитать коэффициенты плоскости;
        2.2.3. Сохранить в массив коэффициентов;
    2.3. Создать объект класса «MatrixBody»;
    2.4. Проверить правильность знаков коэффциентов;
3. Умножить матрицу тела на вектор точки наблюдателя;
4. Для каждого элемента массива, хранящего результата умножения:
    4.1. Если элемент больше нуля, то грань, соответствующая данной плоскости видима, записать в массив видимых граней его название.
5. Конец алгоритма.

В результате работы алгоритма получен массив с названиями видимых граней. На его основе можно производить отрисовку.

#### 2.2.1.2. Алгоритм художника

Словесное описание алгоритма.

1. Начало алгоритма.
2. Определить среднюю координату Z всех элементов, которые не содержат в себе поворачиваемую грань.
3. Для каждого нецентрального элемента, содержащегося в поворачиваемой грани:
   3.1. Определить среднюю координату Z.
4. Отсортировать все полученные координаты Z от меньшего к большему.
5. Для каждого объекта, соответствующего отсортированным значениям:
   5.1. Если элемент принадлежит поворачиваемой грани, отрисовать элемент, иначе – отрисовать группу элементов.
6. Для каждого центра, принадлежащего центральным элементам поворачиваемой грани:
   6.1. Отрисовать элемент.
7. Конец алгоритма.

### 2.2.2.Алгоритм поворота грани головоломки

Для поворота грани головоломки необходимо совершить поворот точек вокруг произвольной оси.

Словесное описание алгоритма.

1. Начало алгоритма.
2. Перенести модель в начало координат.
3. Определить направляющий вектор оси, проходящей через центр поворачиваемой грани.
4. Нормализовать направляющий вектор.

5. Найти синус и косинус угла, позволяющего перенести ось, проходящую через центр поворачиваемой грани, в плоскость **yz**.
6. Найти синус и косинус угла, позволяющего совместить ось, проходящую через центр поворачиваемой грани, с осью **y**.
7. Выполнить последовательный поворот модели на заданные углы вокруг оси **z** и оси **x** соответственно.
8. Выполнить поворот элементов поворачиваемой грани.
9. Выполнить поворот модели в обратном порядке.
10. Перенести модель в исходное состояние.
11. Конец алгоритма.

### 2.2.3.Алгоритм закрашивания

Для закрашивания элементов был выбран простой метод.

Словесное описание алгоритма.

1. Начало алгоритма.
2. Для каждого элемента модели:

2.1. Для каждой грани элемента:

2.1.1. Рассчитать интенсивность излучаемого света в любой точке.

2.1.2. Если грань видима, то для каждого пикселя, принадлежащего грани:

2.1.2.1. Закрасить пиксель соответствующим цветом с рассчитанной интенсивностью.

3. Конец алгоритма.

**Вывод по конструкторской части:** в результате разработки алгоритмов получено их словесное описание, на основе которого теперь можно получить реализацию в коде. Описание модели позволило понять, какую структуру должны иметь объекты в программе, из каких классов должны состоять.

# 3. Технологическая часть.

В данном разделе приведен выбор используемого языка программирования, приведены листинги реализации некоторых алгоритмов.

## 3.1. Язык программирования

В качестве языка программирования для дальнейшей реализации программного продукта был выбран Python. Благодаря своей гибкости он может быть использован во многих сферах, и компьютерная графика – не исключение. В придачу имеется огромное количество библиотек, в том числе и для графического интерфейса. В качестве такой выбрана библиотека PyQt5, которое позволит создать удобный интерфейс.

Python позволяет довольно быстро и элегантно писать код, который будет понятен. К единственному минусу можно отнести не очень быструю скорость работу за счет динамической типизации и ряда других причин. Однако это можно избежать, эффективно реализуя алгоритмы.

## 3.2. Реализация части выбранных алгоритмов

В ходе работы частично разработан интерфейс приложения, написаны алгоритмы удаления нелицевых граней, поворота граней.

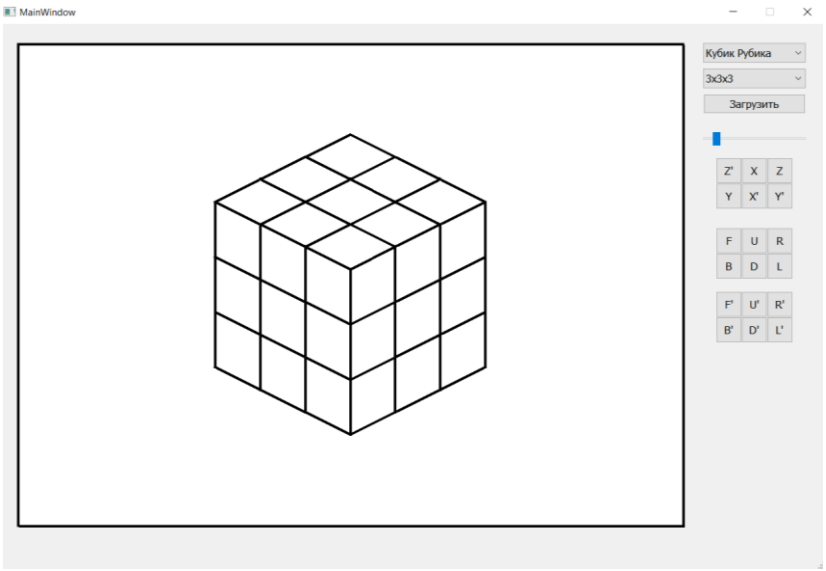


*Рис.8 – Частично разработанный интерфейс*

На листинге 1 представлен алгоритм удаления нелицевых граней. На листинге 2 – алгоритм поворота граней.

```
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]]

    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 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)

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 2*

**Вывод по технологической части:** таким образом, частично был разработан интерфейс приложения, реализованы алгоритмы удаления нелицевых граней, поворота граней головоломки. На данный момент в приложении имеется возможность вращать кубик Рубика вокруг осей, масштабировать, выбирать размерность, поворачивать грани.

# Заключение.

В ходе выполнения работы проведен анализ, на основе которого выбран метод решения той или иной задачи:

- анализ аналогичных приложений, в результате чего составлены собственные требования к программному продукту;
- анализ алгоритмов удаления невидимых поверхностей позволил разработать двухэтапный алгоритм, совмещающий в себе части двух известных (первая – первая часть алгоритма Робертса, вторая – алгоритм художника);
- алгоритм поворота точки вокруг произвольной оси решил проблему поворота граней;
- простой метод закрашивания, в результате – решена проблема закрашивания граней при наличии источника освещения.

Также выбранные алгоритмы разработаны в виде их словесного описания, составлено описание модели с включающими в себя классами. При этом частично разработан интерфейс приложения и программно реализованы алгоритмы удаления нелицевых граней и поворота граней головоломки.

## Список литературы


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).