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 Радиоэлектронные системы и комплексы

Кафедра Технологии приборостроения (РЛ6)

О Т Ч Е Т П О К У Р С О В О Й Р А Б О Т Е

Тема: Разработка симулятора контроллера управления 3Д принтера с картезианской кинематикой.

Студент Филимонов Степан Владиславович

Группа *РЛ6-81*

фамилия, имя, отчество

Студент

подпись, дата *фамилия, и.о.*

Руководитель Дмитриев Д.Д.

подпись, дата *фамилия, и.о.*

Оценка

2024 г.

ОГЛАВЛЕНИЕ

ЦЕЛЬ

Разработать симулятор 3Д принтера с картезианской кинематикой с по этапаным исполнением. Симулятор должен в себя включать по слойный вывод исполненного набора команд.

ВВЕДЕНИЕ

В современном мире 3Д принтеры стали одним из ключевых инновационных технологических решений, которые привносят революционные изменения в промышленность, дизайн, медицину и другие области. Способность создавать трехмерные объекты из различных материалов с помощью 3Д принтера предоставляет уникальные возможности для проектирования, производства и прототипирования.

Важность 3Д принтеров заключается в их способности решать широкий спектр задач: от создания индивидуальных протезов и медицинских имплантатов до изготовления сложных деталей для авиации и автомобилестроения. Эта технология позволяет значительно сократить время и издержки на разработку новых продуктов, улучшить качество и точность изготовления, а также дать возможность индивидуального подхода к производству.

3Д принтеры также способствуют устойчивому развитию экономики и привлекают внимание инновационных стартапов и компаний, стремящихся к совершенствованию своих продуктов и процессов. Благодаря 3Д технологиям открываются новые возможности для творчества, инженерных разработок и науки, что делает эту технологию неотъемлемой частью современного мира и будущего прогресса.

Одним из примеров применения 3Д принтеров является медицина. С их помощью можно создавать индивидуальные протезы, имплантаты и модели органов для планирования сложных операций. Это ускоряет процесс лечения пациентов и улучшает результаты хирургических вмешательств.

В области авиации и автомобилестроения 3Д принтеры используются для изготовления сложных деталей, прототипов и моделей. Это позволяет сократить время и издержки на разработку новых продуктов, а также улучшить их качество и функциональность.

В архитектуре и дизайне 3Д принтеры используются для создания прототипов зданий, интерьеров и декоративных элементов. Это позволяет дизайнерам и архитекторам визуализировать свои идеи, проводить тестирование концепций и создавать уникальные проекты.

Таким образом, 3Д принтеры играют важную роль в современном мире, обеспечивая новые возможности для инноваций, творчества и развития. Их универсальность и применимость делают их неотъемлемой частью различных отраслей промышленности, науки и искусства.

1.Принцип работы 3Д принтера с картезианской механикой

Перед тем как говорить о картезианской механике надо рассказать о FDM(fused deposition modeling, моделирование методом послойного наплавления) печати.

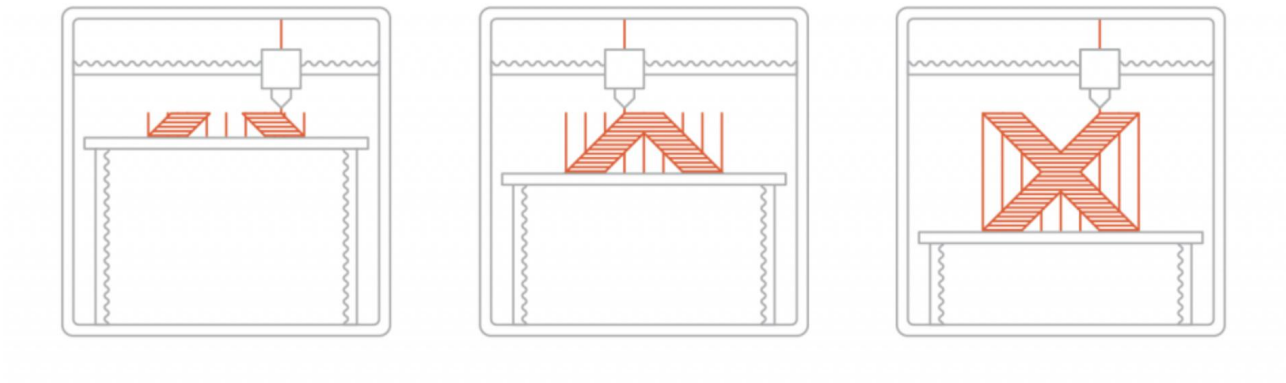

Рис. 1 - Послойная 3Д печать вид сбоку

Вот как работает процесс FDM:

Катушка из термопластичной нити загружается в принтер. Как только сопло достигнет необходимой температуры, нить подается в экструдер и в сопло, где она плавится.

Экструдер прикреплен к 3-осевой системе, которая позволяет ему перемещаться в направлениях X, Y и Z. Расплавленный материал выдавливается в виде тонких нитей и наплавляется послойно в заранее определенных местах, где затем охлаждается и затвердевает. Иногда охлаждение материала ускоряется благодаря использованию вентиляторов, прикрепленных к экструдеру.

Для заполнения печатной области, экструдеру требуется несколько проходов. Когда слой закончен, платформа перемещается вниз (или, как в некоторых моделях принтеров - экструдер перемещается вверх), и новый слой наплавляется на уже схватившийся. Этот процесс повторяется, пока модель не будет напечатана целиком.

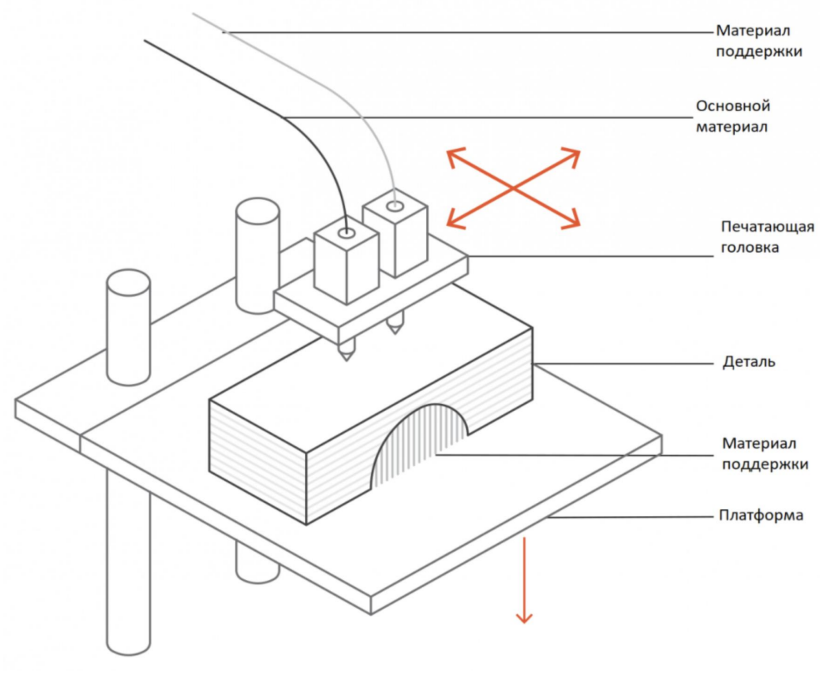

Рис. 2 - Схема области печати

Технологии изготовления объемных предметов методом послойного наплавления материала более тридцати лет, она одна из самых первых, потому что идея ее проста и очевидна. Она же до сих пор самая популярная, и так будет еще долгое время, во всяком случае в быту, благодаря своим преимуществам. Достаточно высокое качество и точность, простота, а следовательно невысокая стоимость оборудования, материалов и готовых изделий - вот причины успеха FDM.

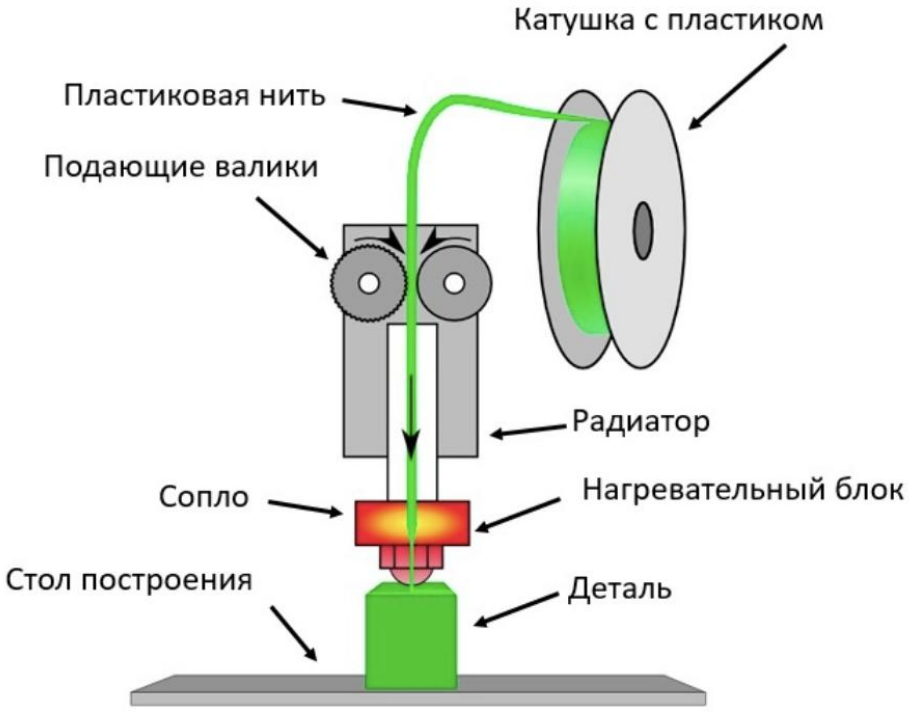

Рис. 3 - Схема устройства Hotend-a

Большинство систем FDM позволяют регулировать несколько параметров процесса печати. Такие как температура сопла, платформы, скорость печати, высоту слоя и скорость вентиляторов охлаждения. Они обычно устанавливаются оператором принтера, и не беспокоят моделлера.

Что важно с точки зрения моделирования, так это учитывать размер стола и высоту слоя самой детали: Стандартный размер печатной области настольного 3Д-принтера обычно составляет 200 x 200 x 200 мм, в то время как для промышленных машин он может достигать 1000 x 1000 x 1000 мм. Если настольный 3Д принтер предпочтительнее (например, из соображений экономии), большую модель можно разбить на более мелкие части и затем собрать/склеить.

Типичная высота слоя, используемая в FDM, варьируется от 50 до 400 микрон и может быть определена на этапе программного слайсинга. Меньшая высота слоя обеспечит более гладкую деталь и более точно отразит сложную геометрию, в то время как большая высота слоя, дает детали распечататься быстрее и с меньшими затратами. Высота слоя 150-200 микрон является оптимальной по соотношению времени печати и её качеству.

Разумеется, у метода есть и недостатки, а именно:

1)необходимость настраивать параметры печати под каждый вид пластика и даже на пластики одного типа разных производителей и партий;

2)точность и разрешение по осям XY ограничена диаметром сопла;
3)по оси Z образуются линии слоев, видимые и ощущаемые даже при минимальном шаге;
4)нависающие части детали нуждаются в поддержке, что приводит к усложнению печати, перерасходу материала, снижению качества поверхности;
5)структура детали получается неоднородной по прочности, поперек слоев максимальная выдерживаемая нагрузка ниже, чем вдоль;
6)детали из пластиков, обладающих даже небольшой термоусадкой, коробятся при печати по причине неравномерного нагрева и остывания, отрываются от стола и искажают геометрические размеры;
7)ограничение по физическим свойствам материалов печати.

Однако недостатки перекрываются плюсами. Вот они, включая упомянутые выше:
1)высокая скорость и низкая стоимость прототипирования;
2)относительно простая и недорогая конструкция принтеров, доступная цена;
3)большой выбор и невысокая стоимость расходных материалов;
4)приемлемая скорость печати, по сравнению с другими технологиями;
5)точность размеров порядка десятой миллиметра и выше, чего достаточно для подавляющего количества изделий;
6)прочность, при некоторых материалах и правильно подобранных параметрах печати, приближается к прочности литых изделий.

Несмотря на то, что все принтеры FDM работают по одному принципу - послойному наплавлению - конструктивные отличия между отдельными их видами весьма существенны. Рассмотрим, в чем между ними разница, а также их сильные и слабые стороны.

3Д принтер с картезианской механикой основан на трех основных осях движения - X, Y и Z. Каждая ось отвечает за определенное направление перемещения печатающей головки и платформы. Движение головки по осям достигается за счет шаговых двигателей, которые управляют перемещениями с заданным шагом.

Ось X обеспечивает движение печатающей головки влево-вправо, ось Y - вперед-назад, а ось Z - подъем и опускание платформы. Это позволяет печатающей головке перемещаться в трехмерном пространстве и наносить пластик или другой материал на платформу в нужных координатах.

Программное обеспечение управляет перемещением печатающей головки по координатам, что позволяет создавать объекты слой за слоем. После нанесения каждого слоя расплавленный материал застывает, образуя очередной слой объекта. Таким образом, 3Д принтер с картезианской механикой позволяет создавать трехмерные объекты с высокой точностью и детализацией.

Этот тип принтера отличается простотой конструкции, что делает его популярным для домашнего использования. Он обеспечивает высокую точность печати и возможность создания сложных объектов. Низкая стоимость и легкость обслуживания делают 3Д принтер с картезианской механикой доступным и удобным инструментом для реализации творческих и инженерных проектов.

2.Почему разрабатывается симулятор именно для картезианской механики

Разработка симулятора именно для картезианской механики имеет несколько причин. Во-первых, картезианская система координат является наиболее простой и понятной для большинства пользователей. Это делает процесс моделирования и визуализации движений и операций на 3D принтерах более доступным и интуитивно понятным.

Во-вторых, алгоритмы и моделирование работы картезианской механики более стандартизированы и распространены, что упрощает разработку симулятора и уменьшает сложность программирования. Это также делает симулятор более удобным для новичков, которым необходимо быстро освоить основы работы 3D принтера.

Кроме того, симулятор для картезианской механики может быть использован как обучающий инструмент для студентов, профессионалов и энтузиастов, желающих изучить принципы работы 3D принтеров. Он помогает лучше понять и визуализировать процесс печати, увеличивая понимание работы устройства и его возможностей.

Таким образом, разработка симулятора для картезианской механики обусловлена ее популярностью, простотой и стандартизацией, что делает такой симулятор удобным и эффективным инструментом для обучения и практического применения в области 3D печати.

3.Алгоритм работы 3Д принтером

Алгоритм работы 3Д принтера достаточно прямолинейн. Происходит точка входа при запуске 3Д принтера, следом происходит инициализация перефии, после инициализации запускается считывание данных о состоянии системы и меню выбора задач, пользователь не может повлиять на считывние данных, может выбрать только пункты доступные в меню, после выбора пункта запускается исполнитель, если после выолнения задания пользователя решает выключить систему, происходит отключение.

Рис. 4 - Общая схема работы 3Д принтера

В процессе инициализации перефирии происходит инициализация периферии в микроконтрлере, а так же инициализация драйверов шаговых

моторов, внешнего дисплея и дополнительных устройство(сетевая карта, тензо датчики).

Подробнее стоит сказать про выбор задач. Выбор задач представляет собой несколько вариантов.

Изменение настроек констант принтера, такие как размеры площади, скорости моторов, установление температур и т. д.

Выполнение процесса печати, открытие внешней системы хранения данных, считывние из неее доступных файов. Определения какие из этих файлов являются gcode и соответственно предоставление пользователю выбора, какой файл отпечатать. После выбора соответствующего файла происходит вызов арбитора печати, который начинает работу над тем чтобы отпечать модель, по данным из файла.

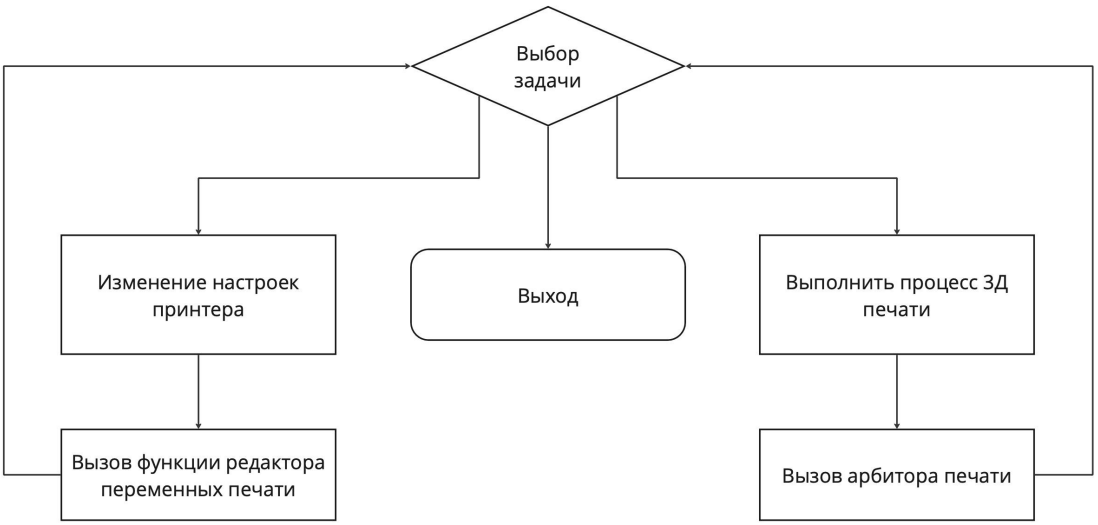

Рис. 5 - Схема выбора задачи в 3Д принтере

Выход - это выход из программы 3д принтера, следовательно его выключение

Изменение настроек принтера - это просто функция, которая изменяет соответствующую настройку 3Д принтера

Арбитр печати представляет из себя сложную структуру.

Арбитр содержит в себе данные по всем G-кодам из документации, так же к ним относятся M-коды.

При вызове арбитора он открывает файл с рашисрением gcode и по строчн начинает выполнять команды из него, пока не закончится файл.

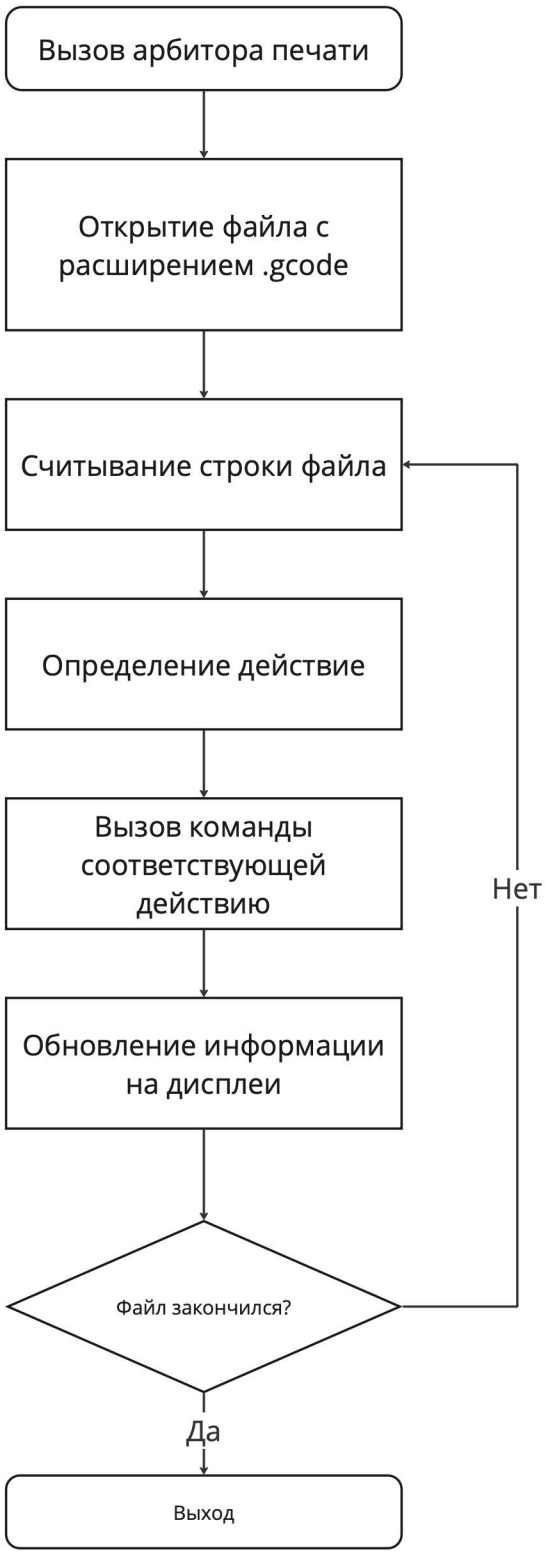

Рис. 6 - Схема работы арбитора в 3Д принтере

4.Алгоритм работы симулятора

Как было сказано ранее перед тем как разрабатывать 3Д принтер стоило сделать его его симулятор. Но весь 3д принтере технически дорого просимулировать, а некоторые узлы в этом не нуждаются. Но в 3Д принтере как было написанно ранее присутствует арбитр, симмулятор которого был написан.

Повторим алгоритм арбитора: При вызове арбитора он открывает файл с рашисрением gcode и по строчн начинает выполнять команды из него, пока не закончится файл.

Арбитр так же содержит в себе данные по всем G-кодам из документации, так же к ним относятся M-коды.

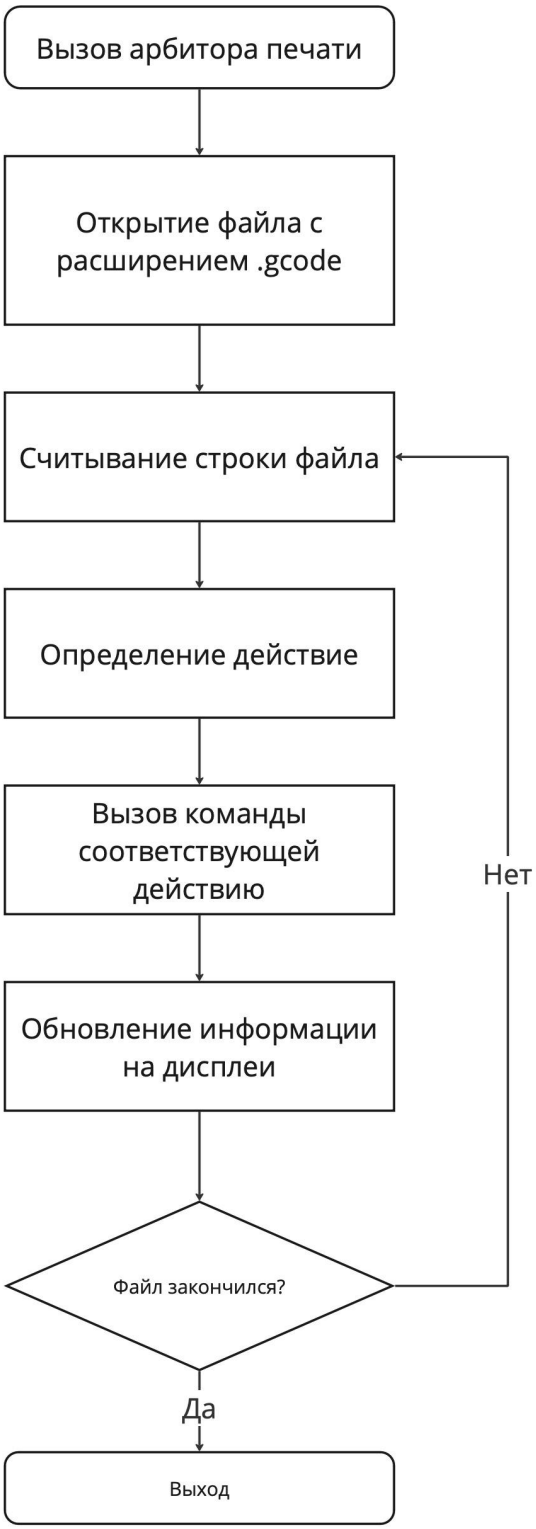

Рис. 7 - Схема работы арбитора в симуляторе

В симмуляторе был реализованн именно такой арбитр, как на схеме, тем самым он максимально приближен к реальным условиям.

Контроль работы выполнялся путем отрисовки слоев на снимках(пример на рисунке). Отрисованный слой можно отследить на достоверность и правильность, а так же определить где возможна ошибка при печати. Отдельные команды отвечающие за булевые операции или изменения режимов работы дублировались в терминал логированием. И позже при анализе кода и логов можно было сопоставить работу виртуального с ожидаемым арбитаром.

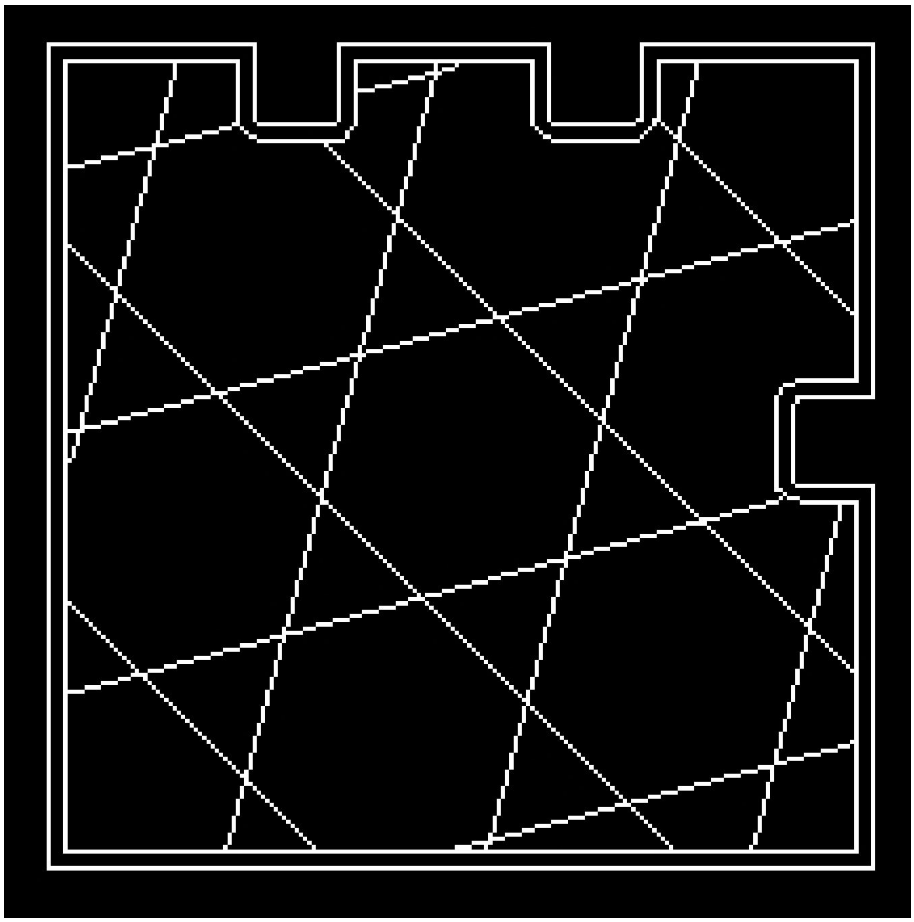

Рис. 9 - Пример снимка слоя(увеличенного)

Предварительно стоит уточнить по коды G0 и G1.

Коды G0 и G1 являются командами в языке G-кода (стандартный язык программирования для управления движением оборудования, включая 3D принтеры), которые управляют перемещением печатающей головки или рабочего инструмента по осям X, Y и Z. Основное отличие между командами G0 и G1 заключается в режиме движения:

1. G0: Команда G0 используется для быстрого перемещения печатающей головки или рабочего инструмента с максимальной скоростью. Эта команда обычно используется для перемещений между деталями печати без

нанесения материала или для быстрого перемещения в зону начала следующего слоя. Движение по команде G0 может выполняться со скоростью, максимально допустимой для конкретной 3D-принтера.

2. G1: Команда G1 используется для обычного движения печатающей головки или рабочего инструмента с определенной скоростью с учетом параметров, таких как скорость движения, длина перемещения и т.д. Эта команда применяется для нанесения материала и создания самого объекта. Движение по команде G1 выполняется с более точной скоростью и позиционированием по сравнению с G0.

Таким образом, основное отличие между командами G0 и G1 заключается в скорости и режиме движения. G0 - для быстрого перемещения без нанесения материала, а G1 - для точного печатного движения с нанесением материала.

В симуляторе отрисовка происходила только линий кода G1, G0 учитывался как перемещение. Таким образом проверялось и подача и перемещение.

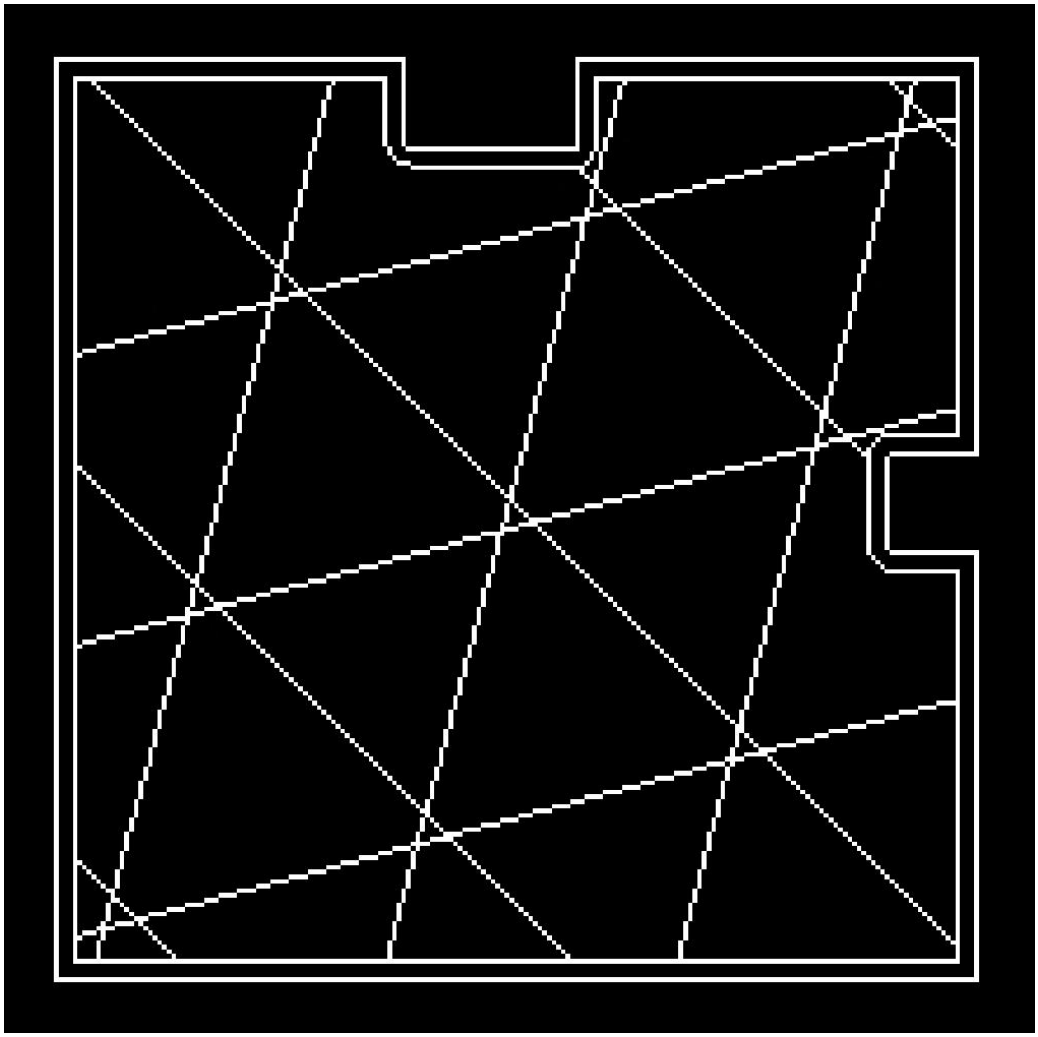

Рис. 10 - Снимок слоя(увеличенный)

Пример общего плана снимка слоя:

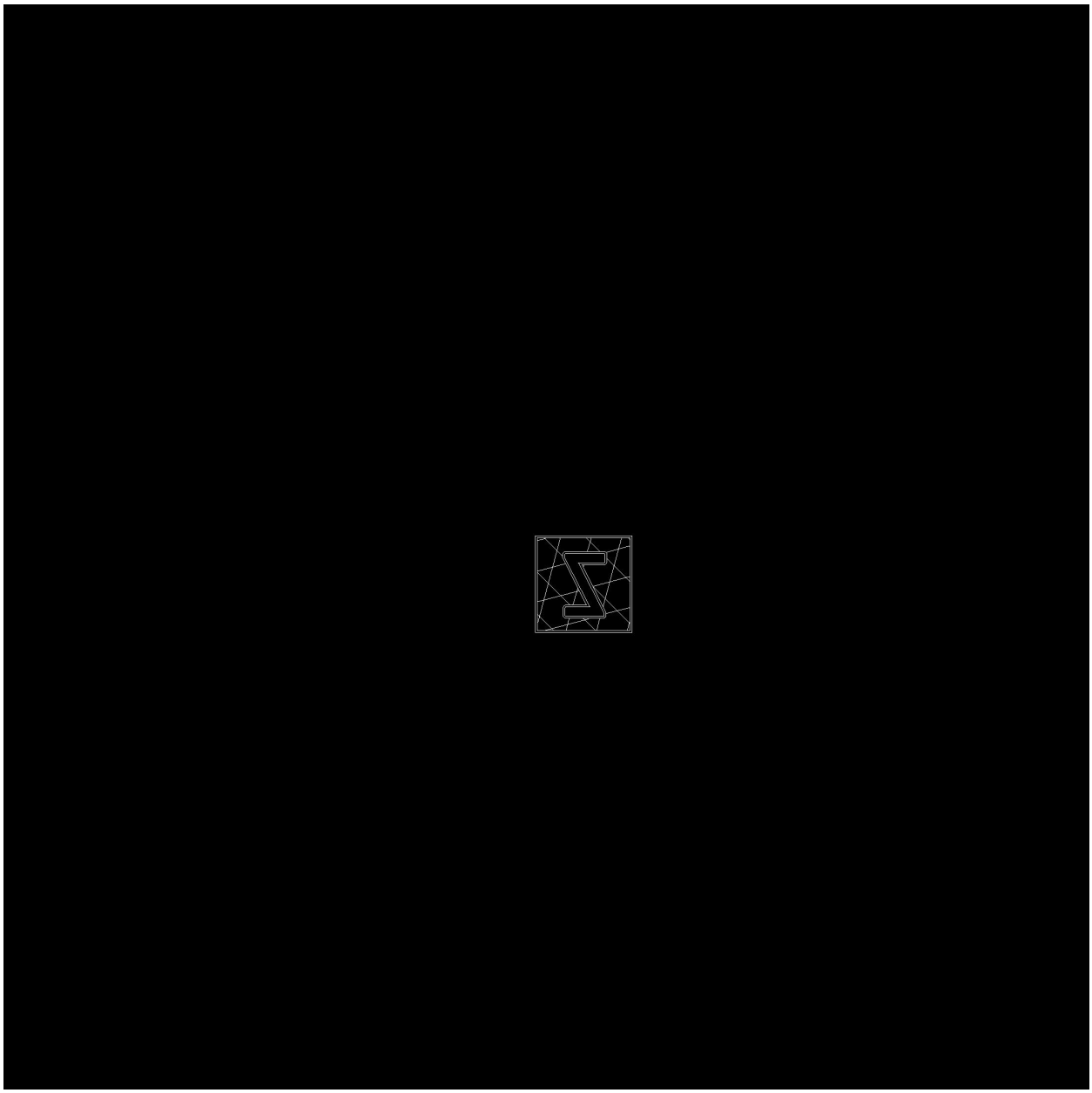

Рис. 12 - Пример слоя(реальный размер)

По результату симуляции результат корректный, так как между линиями подачи присутствуют технические отступы расчитанные на заполнение материалом. Геометрия соответствут геометрии модели и считывние код корректно, что подтверждает логирование кодов:

```
stepanfilimonov@MacBookPro [18:33:25] [~/institute/lessons/DDD] [main *]
-> % ./a.out
M140: Установить температуру стола на 50 Градусов. Не ждать установки
M105: Получить данные о температуре экструдера и стола
M190: Установить температуру стола на 50 Градусов. Ждать установки
M104: Установить температуру экструдера на 200 Градусов. Не ждать установки
M105: Получить данные о температуре экструдера и стола
M109: Установить температуру экструдера на 200 Градусов. Ждать установки
M82: Установить экструдер в абсолютный режим
G92: сброс всех значений
G28: Перейти в точку 0
G92: сброс всех значений
G92: сброс всех значений
G92: сброс всех значений
M107: Выключить вентилятор охлаждения модели
M106: Включить вентилятор охлаждения модели. Мощность: 33 %
M106: Включить вентилятор охлаждения модели. Мощность: 67 %
M106: Включить вентилятор охлаждения модели. Мощность: 100 %
M140: Установить температуру стола на 0 Градусов. Не ждать установки
M107: Выключить вентилятор охлаждения модели
G91: Установка относительных координат
---> Установлены относительные координаты
G90: Установка абсолютных координат
---> Установлены абсолютные координаты
M106: Включить вентилятор охлаждения модели. Мощность: 0 %
M104: Установить температуру экструдера на 0 Градусов. Не ждать установки
M140: Установить температуру стола на 0 Градусов. Не ждать установки
M84: Отключить моторы
---> Моторы отключены
M82: Установить экструдер в абсолютный режим
M104: Установить температуру экструдера на 0 Градусов. Не ждать установки
---> Моторы отключены
```

Рис. 13 - Лог файл

По итогу симулятор арбитора выполняет поставленные задачи.

5.Проблемы возникшие при разработке

При разработке 3Д принтеров и симуляторов для них могут возникнуть различные проблемы и сложности. Некоторые из основных проблем включают в себя:

1. Точность и качество печати: Одной из основных проблем при разработке 3Д принтеров является обеспечение высокой точности и качества печати. Необходимо учесть различные параметры, такие как управление движением печатающей головки, температура расплавленного материала и скорость печати.

2. Калибровка и настройка: Настройка 3Д принтера может быть сложным процессом, требующим опыта и терпения. Необходимо правильно настроить параметры печати, материал и скорость движения, чтобы получить желаемый результат.

3. Проблемы с материалами: Использование различных материалов для печати также может столкнуться с проблемами, такими как сцепление, усадка материала и искажения формы объекта.

4. Программное обеспечение: Разработка и отладка программного обеспечения для 3Д принтера и симуляторов может быть сложным и трудоемким процессом. Необходимо учитывать различные аспекты, такие как визуализация модели, управление движением и обработка данных.

5. Интеграция и совместимость: В случае разработки симуляторов для 3Д принтеров, важно обеспечить их совместимость с различными моделями принтеров и программным обеспечением. Это может потребовать дополнительного тестирования и адаптации.

В целом, разработка 3Д принтеров и симуляторов для них является сложным и многоэтапным процессом, который требует учета множества факторов и особенностей работы данного устройства.

Заключение

В заключение, разработка симулятора контроллера управления 3Д принтера с картезианской кинематикой является важной и актуальной задачей в сфере промышленной автоматизации и создания трехмерных объектов. Картезианская кинематика позволяет управлять печатной головкой в трех направлениях XYZ, что обеспечивает простоту и надежность работы устройства.

Разработка симулятора контроллера позволяет тестировать и отлаживать алгоритмы управления принтером, учитывая различные параметры и ограничения. Важным аспектом является обеспечение точности и стабильности движения печатающей головки, а также управление процессом печати для достижения высокого качества и эффективности.

Продуманная разработка симулятора контроллера управления 3Д принтера позволит повысить эффективность процесса создания трехмерных объектов, сократить время настройки устройства и повысить качество печати. Таким образом, дальнейшее развитие данной области технологий позволит создавать более сложные и функциональные объекты с использованием 3Д принтеров с картезианской кинематикой.

Приложение А

Код программы:

Код

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <vector>
#include <map>
#include <limits>
#include <istream>
#include <iomanip>
#include <cmath>
#include <numbers>
#include <stdexcept>
#include <cstdlib>
#include <filesystem>
#include <sys/stat.h>
#include <unistd.h>

extern "C" {  // jpeglib.h
#include <stdio.h>
#include <setjmp.h>
#include <unistd.h>
#include <stdlib.h>
#include <jconfig.h>
#include <jpeglib.h>
}

#define SIZE_STEPS          200  // количество шагов на оборот для двигателя (в наших примерах 200)
#define MICROSTEP           16  // микрошаг (1, 2, 4, 8 и т. д.)
#define BELT_PITCH          2  // шаг ремня (например, 2 мм)
#define NUMBER_TEETH_PULLEY 20  // количество зубьев на шкиве, на валу двигателя.

const std::string FILE_NAME = "CE3E3V2_xyzCalibration_cube.gcode";

class Matrix;

/** @brief Ошибка матрицы, с выводом сообщения
 * */
```

```
class MatrixException : public std::exception {
    std::string m_msg;
public:
    explicit MatrixException(const std::string &msg) : m_msg(msg) {}
    const char *what() const noexcept override { return m_msg.c_str(); }
};

/** @brief Ошибка матрицы при чтении из потока
 * */
class InvalidMatrixStream : public MatrixException {
public:
    InvalidMatrixStream() : MatrixException("Произошла ошибка при чтении из потока") {}
};

/** @brief Ошибка матрицы, выход за пределы матрицы
 * */
class OutOfRange : public MatrixException {
public:
    OutOfRange(size_t i, size_t j, const Matrix &matrix);
};



/* @class Matrix класс матрицы двумерной. Различные операции для расчетов
 * @param rows Строки
 * @param cols Колонки
 * @param matrix Массив элементов матрицы
 * */
class Matrix {
private:
    size_t rows;
    size_t cols;
    uint8_t* matrix;

    static inline uint8_t get_grey(const uint8_t& R, const uint8_t& G, const uint8_t& B) {
        return  uint8_t(float(R) * float(0.299) + float(G) * float(0.587) + float(B) * float(0.114));
    }
public:
    /** @defgroup Базовые операции
     *  В данной группе содержатся различные конструкторы, оператор присваивания, а так же методы
     *  получения/изменения элемента и получения размеров матрицы
     *  @{
```

```
     */

    /** @brief Конструктор выделяет под матрицу размерами rows x cols область памяти, заполненную 0-
ями
     *  @param rows Колличество строк в матрице, считается по умолчанию, что он >= 0
     *  @param cols Колличество колонок в матрице, считается по умолчанию, что он >= 0
     * */
    explicit Matrix(const size_t& _rows = 0, const size_t& _cols = 0) : rows(_rows), cols(_cols) {
        matrix = new uint8_t[( rows * cols )];
        for ( size_t i = 0, size = ( rows * cols ); i < size; ++i )
                 matrix[i] = 0;
    }

    /** @brief Конструктор копирования
     *  @param rhs Другая матрица
     * */
    Matrix(const Matrix& rhs) noexcept {
        rows = rhs.getRows();
        cols = rhs.getCols();
          matrix = new uint8_t[( rows * cols )];
        for ( size_t i = 0; i < rows; ++i )
                 for ( size_t j = 0; j < cols; ++j )
                matrix[( j + ( i * cols ) )] = rhs.matrix[( j + ( i * cols ) )];
    }

    /** @brief Оператор присваивания
     *  @param rhs Другая матрица
     *  @return Возвращает присвоенную матрицу
     * */
    Matrix &operator=(const Matrix& rhs) noexcept {
        if ( &rhs != this ) {
            delete[] matrix;
                 rows = rhs.getRows();
            cols = rhs.getCols();
            matrix = new uint8_t[( rows * cols )];
                 for ( size_t i = 0; i < rows; ++i )
                for ( size_t j = 0; j < cols; ++j )
                    matrix[( j + ( i * cols ) )] = rhs.matrix[( j + ( i * cols ) )];
          }
        return *this;
    }
```

```
/** @brief Деструктор, должен очистить память
 * */
~Matrix() noexcept {
    delete[] matrix;
}

/** @brief Метод получения колличества строк
 *  @return Возвращает переменную rows
 * */
size_t getRows() const noexcept { return cols; }

/** @brief Метод получения колличества колонок
 *  @return Возвращает переменную cols
 * */
size_t getCols() const noexcept { return cols; }

  /** @brief Проверка на нулевую матрицу
   *  @return Если матрица пустая вернет true, в противном случае false
   * */
bool isNull() const noexcept { return ((cols == 0) || (rows == 0)); }

/** @brief Спомощью такой перегруженной функциональные формы происходит
 *         извлечение элемента без его изменения.
 *  @param i номер строки, если не входит в границы, вернуть 0
 *  @param j номер колонки, если не входит в границы, вернуть 0
 *  @return Возвращает элемент под номер строки i и колонки j
 *  @exception OutOfRange() Выход за пределы матрицы
 * */
uint8_t operator()(size_t i, size_t j) const {
    if ( ( ( i >= rows ) || ( j >= cols ) ) )
        throw OutOfRange( i, j, *this );
      return matrix[( j + ( i * cols ) )];
}

/** @brief Спомощью такой перегруженной функциональные формы происходит
 *         извлечение элемента для его дальнейшего изменения.
 *  @param i номер строки, если не входит в границы, вернуть 0
 *  @param j номер колонки, если не входит в границы, вернуть 0
 *  @return Возвращает элемент под номер строки i и колонки j, для его изменения
 *  @exception OutOfRange() Выход за пределы матрицы
 * */
```

```
    uint8_t& operator()(size_t i, size_t j) {
        if ( ( ( i >= rows ) || ( j >= cols ) ) )
            throw OutOfRange( i, j, *this );
          return ( ( uint8_t& )matrix[( j + ( i * cols ) )] );
    }

    /** @} */ // Конец группы: Базовые операции

    /** @defgroup Математические операции над матрицами
     *  Булевые операции и арифметические операции(сложение, вычитание, умножение)
     *  @{
     */

    /** @brief Булевая операции равенство
     *  @param rhs Матрица
     *  @return true если матрицы равны, false если не равны
     * */
    bool operator==(const Matrix& rhs) const noexcept {
        if ( ( ( rows != rhs.rows ) || ( cols != rhs.cols ) ) )
            return false;
          for ( size_t i = 0; i < rows; ++i )
            for ( size_t j = 0; j < cols; ++j )
                        if ( std::abs( matrix[( j + ( i * cols ) )] - rhs.matrix[( j + ( i *
cols ) )]) > 1e-7 )
                    return false;
        return true;
    }

    /** @brief Булевая операци неравенства
     *  @param rhs Матрица
     *  @return false если матрицы равны, true если матрицы не равны
     * */
    bool operator!=(const Matrix& rhs) const noexcept { return !( *this == rhs ); }

    /** @} */ // Конец группы: Математические операции над матрицами

    /** @defgroup Дополнительные операции над матрицами
    *  @{
    */

    void openJpeg( const std::string& fileName ) {
        struct jpeg_decompress_struct d1;
```

```
        struct jpeg_error_mgr m1;
        d1.err = jpeg_std_error(&m1);
        jpeg_create_decompress(&d1);
        FILE *f = fopen(fileName.c_str(),"rb");
        jpeg_stdio_src(&d1, f);
        jpeg_read_header(&d1, TRUE);
        if( ( cols != 0 ) || ( rows != 0 ) )
            delete[] matrix;
        rows = d1.image_height;
        cols = d1.image_width;
        matrix = new uint8_t[( rows * cols )];

        jpeg_start_decompress(&d1);
        uint8_t *pBuf = new uint8_t[rows * cols * d1.num_components]{};
        for (size_t i = 0; d1.output_scanline < d1.output_height;) {
            i += jpeg_read_scanlines(&d1, (JSAMPARRAY)&(pBuf), 1);
            for (size_t j = 0; j < cols; ++j) {
                matrix[j + (i - 1) * cols] = get_grey(pBuf[j * d1.num_components + 2],
                                                      pBuf[j * d1.num_components + 1],
                                                      pBuf[j * d1.num_components + 0]);
            }
        }

        jpeg_finish_decompress(&d1);
        jpeg_destroy_decompress(&d1);
        fclose(f);
        delete[] pBuf;
    }

    void saveJpeg( const std::string& fileName ) {
        struct jpeg_compress_struct cinfo;  /* Шаг 1: выделите и инициализируйте объект сжатия JPEG
*/
        struct jpeg_error_mgr jerr;
        JSAMPROW row_pointer[1];
        cinfo.err = jpeg_std_error(&jerr);

        jpeg_create_compress(&cinfo);  /* Шаг 2: укажите место назначения данных  (eg, a file) */
        FILE *outfile = fopen( fileName.c_str(),"wb" );

        jpeg_stdio_dest(&cinfo, outfile); /* Шаг 3: установите параметры для сжатия */
        cinfo.image_width = cols;    /* Количество столбцов в изображении */
        cinfo.image_height = rows;   /* Количество строк в изображении */
```

```
        cinfo.input_components = 1;     // Каналы, RGB 3 ; GRAY 1
        cinfo.in_color_space = J_COLOR_SPACE(1);
        JSAMPLE* image_buffer = new JSAMPLE[cols * rows]();  /* Указывает на большой массив данных R,
G, B-порядка */
        for (size_t i = 0; i < cols * rows; ++i) {
            image_buffer[i] = matrix[i];
        }
        jpeg_set_defaults(&cinfo);
        jpeg_set_quality(&cinfo, 100, true);


        jpeg_start_compress(&cinfo, true);  /* Шаг 4: Запустите компрессор */


        while (cinfo.next_scanline < cinfo.image_height) {  /* Шаг 5: пока (строки сканирования еще
предстоит записать)*/
            row_pointer[0] = (JSAMPROW)&image_buffer[cinfo.next_scanline * cols];
            (void)jpeg_write_scanlines(&cinfo, row_pointer, 1);
        }


        jpeg_finish_compress(&cinfo);  /* Шаг 6: Завершите сжатие */
        jpeg_destroy_compress(&cinfo);  /* Шаг 7: освободите объект сжатия JPEG */
        fclose(outfile);
    }


    void drawLine( int x0, int y0, int x1, int y1, const uint8_t& color ) {
        int A = ( y1 - y0 ), B = ( x0 - x1 );
        int sign = ( ( std::abs(A) > std::abs(B) ) ? 1 : -1 );
        int signa = ( ( A < 0 ) ? -1 : 1 ), signb = ( ( B < 0 ) ? -1 : 1 );
        int f = 0;
        matrix[(  x0 + ( y0 * cols )  )] = color;
        int x = x0, y = y0;
        if( sign == -1 ) {
            do {
                f += ( A * signa );
                if( f > 0 ) {
                    f -= ( B * signb );
                    y += signa;
                }
                x -= signb;
                matrix[( x + ( y * cols ) )] = color;
            } while (x != x1 || y != y1);
        } else {
            do {
```

```
                f += B * signb;
                if( f > 0 ) {
                    f -= A * signa;
                    x -= signb;
                }
                y += signa;
                matrix[( x + ( y * cols ) )] = color;
            } while( ( x != x1 ) || ( y != y1 ) );
        }
    }

    void clear() {
        for( size_t i = 0; i < ( rows * cols ); ++i )
            matrix[i] = 0;
    }

    /** @} */ // Конец группы: Дополнительные операции над матрицами
};

OutOfRange::OutOfRange(size_t i, size_t j, const Matrix &matrix) : MatrixException(
    "Индексы (" + std::to_string(i) + ", " + std::to_string(j) +
    ") выход за границы матрицы. Размер матрицы [" +
    std::to_string(matrix.getRows()) + ", " + std::to_string(matrix.getCols()) + "]"
) {}

class CNCException : public std::exception {
    std::string m_msg;
public:
    explicit CNCException(const std::string &msg) : m_msg(msg) {}
    const char *what() const noexcept override { return m_msg.c_str(); }
};

class UnknownGCode : public CNCException {
    std::string m_msg;
public:
    explicit UnknownGCode( const std::string &code ) : CNCException( "Неизвестный код!: " + code ) {}
};

class FileNotOpen : public std::exception {
    std::string m_msg;
public:
```

```
    explicit FileNotOpen(const std::string &msg) : m_msg(msg) {}
    const char *what() const noexcept override { return m_msg.c_str(); }
};

struct Axes {
    float _x;
    float _y;
    float _z;
    float _e;
    uint16_t _f;
    Axes() : _x(0), _y(0), _z(0), _e(0), _f(0.0) {}
    Axes( const float& x, const float& y, const float& z, const float& e, const uint16_t& f ) : _x(x), _y(y),
        _z(z), _e(e), _f(f) {}
};

class StepperMotor {
public:
    virtual ~StepperMotor() {}

    virtual void moveE( const Axes& ax ) = 0;
    virtual void move( const Axes& ax ) = 0;
    virtual void setting( const Axes& ax ) = 0;

    virtual void on() = 0;
    virtual void off() = 0;

    virtual void relativeAxes() = 0;
    virtual void absoluteAxes() = 0;
};

#define MATRIX_SCALER_SIZE 1000
#define TABLE_SIZE 2.2

class MatrixMotor : public StepperMotor {
    int _prevX = 0, _prevY = 0, _prevZ = 0, _prevE = 0;
    int _x = 0, _y = 0, _z = 0, _e = 0;
    Matrix m;

    bool isWork;
    void saveLayer( const float& layer ) {
        static int i = 0;
```

```
        std::string strL = std::to_string( ( std::round( layer * 10 ) / 10 ) );
        strL = strL.substr( 0, ( strL.length() - 5 ) );
        m.saveJpeg( ( "img/layer_" + std::to_string( i++ ) + "_" + strL + ".jpg" ) );
        m.clear();
    }

public:
    MatrixMotor() {
        Matrix mat( ( TABLE_SIZE * MATRIX_SCALER_SIZE ), ( TABLE_SIZE * MATRIX_SCALER_SIZE ) );
        m = mat;
        isWork = true;
    }

    ~MatrixMotor() {}

    void moveE( const Axes& ax ) override {
        if( !isWork )
            return;

        _x = ( ( ax._x != 0 ) ? std::round( ax._x * 10 ) : _prevX );
        _y = ( ( ax._y != 0 ) ? std::round( ax._y * 10 ) : _prevY );
        if( ax._z != 0 )
            saveLayer( ax._z );

        if( ( ( _prevX == _x ) && ( _prevY == _y ) ) )
            return;

        m.drawLine( _prevX, _prevY, _x, _y, 255 );
        setting( ax );
    }

    void move( const Axes& ax ) override {
        if( !isWork )
            return;

        _x = ( ( ax._x != 0 ) ? std::round( ax._x * 10 ) : _prevX );
        _y = ( ( ax._y != 0 ) ? std::round( ax._y * 10 ) : _prevY );
        if( ax._z != 0 )
            saveLayer( ax._z );

        if( ( ( _prevX == _x ) && ( _prevY == _y ) ) )
```

```
            return;
        setting( ax );
    }

    void setting( const Axes& ax ) override {
        _prevX = ( ( ax._x != 0 ) ? std::round( ax._x * 10 ) : _prevX );
        _prevY = ( ( ax._y != 0 ) ? std::round( ax._y * 10 ) : _prevX );
        _prevZ = ( ( ax._z != 0 ) ? std::round( ax._z * 10 ) : _prevX );
        _prevE = ( ( ax._e != 0 ) ? std::round( ax._e * 10 ) : _prevX );
    }

    void on() override {
        isWork = true;
        std::cout << "---> Моторы включены" << std::endl;
    }

    void off() override {
        isWork = false;
        std::cout << "---> Моторы отключены" << std::endl;
    }

    void relativeAxes() override {
        std::cout << "---> Установлены относительные координаты" << std::endl;
    }

    void absoluteAxes() override {
        std::cout << "---> Установлены абсолютные координаты" << std::endl;
    }
};

class Arbitr {
private:
    using cfp = std::pair<char, float>;
    std::ifstream inFile;
    size_t fileSize;
    size_t currentSize;
    StepperMotor* motors;

    std::vector<std::string> split( const std::string &s, char delim ) {
        std::vector<std::string> elems;
        std::stringstream ss(s);
```

```
        std::string item;
        while( std::getline( ss, item, delim ) )
            elems.push_back(item);
        return elems;
    }

    cfp getCmdId( std::string id ) {
        char code = id[0];
        id.erase(0, 1);
        return cfp( code, ( ( !id.empty() ) ? std::stof(id) : std::numeric_limits<float>::min() ) );
    }

    Axes getAxes( const cfp* pairs, const size_t& size ) {
        Axes ax;
        for( size_t i = 0; i < size; ++i ) {
            switch( pairs[i].first ) {
                case 'X':
                    ax._x = pairs[i].second;
                    break;
                case 'Y':
                    ax._y = pairs[i].second;
                    break;
                case 'Z':
                    ax._z = pairs[i].second;
                    break;
                case 'E':
                    ax._e = pairs[i].second;
                    break;
                case 'F':
                    ax._f = pairs[i].second;
                    break;
                default:
                    break;
            }
        }
        return ax;
    }

    void G0( const cfp* pairs, const size_t& size ) {
        Axes ax = getAxes( pairs, size );
        motors->move( ax );
```

```
}

void G1( const cfp* pairs, const size_t& size ) {
    Axes ax = getAxes( pairs, size );
    motors->moveE( ax );
}

void G28() {
    std::cout << "G28: Перейти в точку 0" << std::endl;
    motors->move( Axes() );
}

void G90() {
    std::cout << "G90: Установка абсолютных координат" << std::endl;
    motors->absoluteAxes();
}

void G91() {
    std::cout << "G91: Установка относительных координат" << std::endl;
    motors->relativeAxes();
}

void G92() {
    std::cout << "G92: сброс всех значений" << std::endl;
    motors->setting( Axes() );
}

void M82() {
    std::cout << "M82: Установить экструдер в абсолютный режим" << std::endl;
}

void M84() {
    std::cout << "M84: Отключить моторы" << std::endl;
    motors->off();
}

void M104( const cfp* pairs, const size_t& size ) {
    std::cout << "M104: Установить температуру экструдера на "
              << std::to_string(((int)pairs[0].second))
              << " Градусов. Не ждать установки" << std::endl;
}
```

```
void M105( const cfp* pairs, const size_t& size ) {
    std::cout << "M105: Получить данные о температуре экструдера и стола" << std::endl;
}

void M106( const cfp* pairs, const size_t& size ) {
    std::cout << "M106: Включить вентилятор охлаждения модели. Мощность: "
              << std::round( pairs[0].second / 255 * 100 ) << " %" << std::endl;
}

void M107() {
    std::cout << "M107: Выключить вентилятор охлаждения модели" << std::endl;
}

void M109( const cfp* pairs, const size_t& size ) {
    std::cout << "M109: Установить температуру экструдера на "
              << std::to_string(((int)pairs[0].second))
              << " Градусов. Ждать установки" << std::endl;
}

void M140( const cfp* pairs, const size_t& size ) {
    std::cout << "M140: Установить температуру стола на "
              << std::to_string(((int)pairs[0].second))
              << " Градусов. Не ждать установки" << std::endl;
}

void M190( const cfp* pairs, const size_t& size ) {
    std::cout << "M190: Установить температуру стола на "
              << std::to_string(((int)pairs[0].second))
              << " Градусов. Ждать установки" << std::endl;
}

void callCode( const std::string& cmd, const cfp* pairs, const size_t& size ) {
    if( cmd == "G0" )
        G0( pairs, size );
    else if( cmd == "G1" )
        G1( pairs, size );
    else if( cmd == "G28" )
        G28();
    else if( cmd == "G90" )
        G90();
```

```
        else if( cmd == "G91" )
            G91();
        else if( cmd == "G92" )
            G92();
        else if( cmd == "M82" )
            M82();
        else if( cmd == "M84" )
            M84();
        else if( cmd == "M104" )
            M104( pairs, size );
        else if( cmd == "M105" )
            M105( pairs, size );
        else if( cmd == "M106" )
            M106( pairs, size );
        else if( cmd == "M107" )
            M107();
        else if( cmd == "M109" )
            M109( pairs, size );
        else if( cmd == "M140" )
            M140( pairs, size );
        else if( cmd == "M190" )
            M190( pairs, size );
        else
            throw UnknownGCode(cmd);
    }


public:


    Arbitr( const std::string& fileName, StepperMotor* m ) : inFile(fileName), fileSize(0),
currentSize(0) {
        if( !inFile )
            throw FileNotOpen( fileName );
        inFile.seekg( 0, std::ios::end );
        fileSize = inFile.tellg();
        inFile.seekg(0);
        motors = m;
    }


    ~Arbitr() {
        motors->off();
        inFile.close();
    }
```

```
    int make() {
        for( std::string strReaded = "", strClear = ""; inFile; std::getline( inFile, strReaded ) ) {
            currentSize += ( strReaded.size() + 1 );
            strClear = strReaded.substr( 0, strReaded.find(";") );
            if( strClear.empty() )
                continue;
            auto commands = split( strClear, ' ' );
            const size_t size = ( commands.size() - 1 );
            cfp* pairs = new cfp[size];
            const std::string cmd = commands[0];
            for( size_t i = 1; i < commands.size(); ++i )
                pairs[( i - 1 )] = getCmdId(commands[i]);
            try {
                callCode( cmd, pairs, size );
            } catch ( const UnknownGCode& ugc ) {
                delete[] pairs;
                std::cout << ( std::round( float(currentSize) / float(fileSize) * 100.0 ) / 100.0 )
<< " %" << std::endl;
                std::cout << ugc.what() << std::endl;
                return -1;
            };
            delete[] pairs;
        }
        return 0;
    }
};


int main() {
    struct stat st;
    if( !( ( stat( "img", &st ) == 0 ) && S_ISDIR(st.st_mode) ) )
        if ( mkdir( "img", ( S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH ) ) != 0 )
            throw;
    MatrixMotor mm;
    Arbitr arbitr( FILE_NAME, &mm );
    return arbitr.make();
}
```

Список литературы

1. "Язык программирования C++. Краткое руководство для программистов" - Стратегическая книга

2. "C++ Primer Plus" - Стивен Прата

3. "C/C++. Программирование на языке высокого уровня" - Д. Хокинс

4. "3D Printing for Dummies" - Kalani Kirk Hausman, Richard Horne