МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

РАБОТА ПРОВЕРЕНА
Рецензент
_____
«___»_____ 2021 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____Г.И. Радченко
«___»_____ 2021 г.

Разработка системы моделирования 5-осевой фрезерной обработки на Unity

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090401.2021.594 ПЗ ВКР

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____В.А.Парасич
«___»_____2021 г.

Автор работы,
студент группы КЭ-222
_____ Д.И.Захаров
«___»_____2021 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____
«___»_____2021 г.

Челябинск-2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____2021 г.

ЗАДАНИЕ

на выпускную квалификационную работу магистра

студенту группы КЭ-222
Захарову Даниле Игоревичу
обучающемуся по направлению
09.04.01 «Информатика и вычислительная техника»

Тема работы: «Разработка системы моделирования 5-осевой фрезерной обработки на Unity» утверждена приказом по университету от 26 апреля 2021 года №714-13/12 (приложение №73)

Срок сдачи студентом законченной работы: 01 июня 2021 г.

Исходные данные к работе:

- Степанов, Д. Д. Вопросы разработки обучающих симуляторов крановых операций в морском порту в среде Unity / Д.Д. Степанов // Системный анализ и логистика: журнал.: выпуск No3(25), ISSN 2007-5687. – 2020. – СПб.: ГУАП. с. 25-32. РИНЦ
- Grabowski, R. Enhanced simulation of the milling process for a novel roughing and finishing tool / R. Grabowski // Garbsen : TEWISS Verlag – 2019. – 10.15488/9944
- Chen, W. A forward closed-loop virtual simulation system for milling process considering dynamics processing-machine interactions / Chen W,

Tong Z, Huo D, Zhong W, Jiang X // Springer London – 2019 – oai:eprint.ncl.ac.uk:258813

- Емелин, А. Г. Геометрическое моделирование технологического процесса фрезерной обработки на станках с ЧПУ : специальность 05.13.12 "Системы автоматизации проектирования (по отраслям)" : автореферат диссертации на соискание ученой степени кандидата технических наук / Емелин А. Г. – Екатеринбург, 2004. – 18 с.

- Юсупова, Ф. Э. Симуляторы в образовательном процессе / Ф. Э. Юсупова, М. Об. К. Солижонова // Вопросы науки и образования. 2018. №10 (22). URL: https://cyberleninka.ru/article/n/simulyatory-v-obrazovatelnom-protsesse (дата обращения: 01.04.2021).

Перечень подлежащих разработке вопросов:

- Изучение существующих разработок в области эмуляции процессов фрезерной обработки;
- Исследование алгоритмов компьютерной графики и особенностей создания 3D моделей путем генерирования точек;
- Анализ существующих решений;
- Проектирование системы моделирования фрезерной обработки;
- Реализация системы;
- Тестирование;

Дата выдачи задания: 13 января 2021 г.

Руководитель работы:

Доцент каф. ЭВМ, к.т.н. _____/*В.А. Парасич* /

Студент_____/*Д.И. Захаров* /

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	01.03.2021	
Разработка модели, проектирование	01.04.2021	
Реализация системы	01.05.2021	
Тестирование, отладка, эксперименты	15.05.2021	
Компоновка текста работы и сдача на нормоконтроль	27.05.2021	
Подготовка презентации и доклада	30.05.2021	

Руководитель работы_____/*В.А. Парасич* /

Студент _____/*Д.И. Захаров* /

Аннотация

Д.И. Захаров. Разработка системы моделирования 5-осевой фрезерной обработки. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2021, 61 с., 28 ил., библиогр. список – 23 наим.

В рамках данной выпускной квалификационной работы производится разработка системы моделирования 5-осевой фрезерной обработки на станках с числовым программным управлением (ЧПУ). Осуществляется анализ алгоритмов компьютерной графики, а также особенности генерации 3D-моделей в среде разработке компьютерных игр Unity. Производится обзор существующих решений на рынке CAD/CAM-систем.

Пояснительная записка включает в себя введение, оглавление, основную часть, заключение и библиографический список.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время использование компьютерных симуляторов и тренажеров перестало быть дорогостоящим удовольствием и стало распространенным источником получения навыков специалиста.

Обучение технического персонала влечет за собой ошибки, цена которым иногда бывает испорченное оборудование, которое зачастую является очень дорогим. Симуляторы же дают возможность без каких-либо инструментов моделировать физические процессы и виртуально проводить лабораторные занятия. Преимущество симуляторов, не требующее финансовых затрат, дает возможность проводить студентами некоторые исследования стократно. Также, они абсолютно безопасны, но при этом позволяют отследить многие ошибки, которые в реальном мире могут привести к непоправимым последствиям.

На данный момент в компании «Учтех-Профи» существует тренажер-эмулятор «Оператор токарного и фрезерного станков с ЧПУ». Он позволяет:

- обеспечить подготовку операторов токарного и фрезерного станков (знания, умения, навыки);
- обеспечить визуализацию пульта управления и рабочего пространства;
- позволяет программировать стойки ЧПУ моделей Siemens Sinumerik 828D, Siemens Sinumerik 840D, Fanuc 0i, Haas;
- имитировать действия оператора и движения рабочих органов машины;
- контролировать уровень подготовленности оператора;
- ознакомиться с техникой безопасности;
- демонстрировать работу имитатора на большом видеоэкране для наблюдения соучеников и инструктора (преподавателя);
- визуально отобразить процесс изготовления деталей с поэтапными технологическими комментариями.

Однако, он позволяет визуализировать только сам процесс обработки детали на токарных и фрезерных станках без ознакомления непосредственно

с устройством станка. Кроме того, графика в этом тренажере достаточно устарела.

Новое программное обеспечение разрабатывается в Unity. Он позволяет разрабатывать кроссплатформенные приложения, имея при этом подходящие инструменты разработки.

Для достижения поставленной цели необходимо решить следующие поставленные задачи:

1. Провести обзор и анализ существующих аналогичных или похожих решений.
2. Провести анализ существующих алгоритмов компьютерной графики для редактирования 3D моделей путем генерирования точек.
3. Разработать парсер программ на G-кодах, используемых во фрезерных станках с ЧПУ.
4. Разработать модуль визуализации обработки 3D модели детали.
5. Разработать архитектуру приложения.
6. Провести тестирование и оценить работоспособность системы.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. АКТУАЛЬНОСТЬ

Использование моделей в учебном процессе не считается новым методом. Модели используются издавна. Симуляторы используются во всех отраслях учебного процесса, начиная с начальных классов до высших образовательных учреждений. С помощью симуляторов студенты, хоть и виртуально, имеют возможность воплотить в жизнь изученные на уроках знания. В ходе проведения этих исследований студенты укрепляют теоретические знания и вносят свою пользу в развитие внедрения их в жизнь. Кроме того вносят свой вклад в развитие и прогрессирование самих симуляторов, поднятию уровня симуляторов до того уровня, когда они будут давать результаты одинаковые с реальными исследованиями. [1]

Разработка программ для обучения технического персонала становится особенно выгодной в наше время. Этому способствуют следующие факторы:

– совершенствование сред разработки программного обеспечение, появление новых алгоритмов рендеринга, визуализации, обработки физических процессов;

– рост производительности аппаратного обеспечения. В наши дни запускать подобные программы можно на компьютерах со сравнительно недорогими процессорами и видеокартами. Естественно, все зависит от сложности программы и вычисляемого процесса;

– дешевизна подобных симуляторов в сравнении с обучением на реальном оборудовании. Стоимость станка может достигать нескольких миллионов рублей, соответственно ремонт тоже стоит очень дорого. Использование симуляторов позволяет решить эту проблему;

– возможность многократных повторений испытаний и отработка навыков студентом. Данный фактор особенно важен для студентов,

обучающихся на технологов и наладчиков станков с ЧПУ, так как некоторые действия необходимо произвести многократно, а также отработать ошибки, чтобы не испортить заводское оборудование.

Таким образом, разработка подобных программ является перспективной развивающейся отраслью.

1.2. ОБЗОР АНАЛОГОВ

На рынке программного обеспечения существует множество программ эмуляции фрезерного станка. [2]

Данные программы принадлежат к классу CAM-программ, в которые можно загрузить параметры заготовки, резцов, а также программу обработки, написанной на G-code и она покажет процесс обработки и результат. [3]

Рассмотрим некоторые из них.

1.2.1. LinuxCNC

Свободное программное обеспечение, предназначенное для связи с контроллерами станка и выполнения программ обработки, однако имеет функционал предварительной визуализации. Предназначено только для платформы Linux. [3]

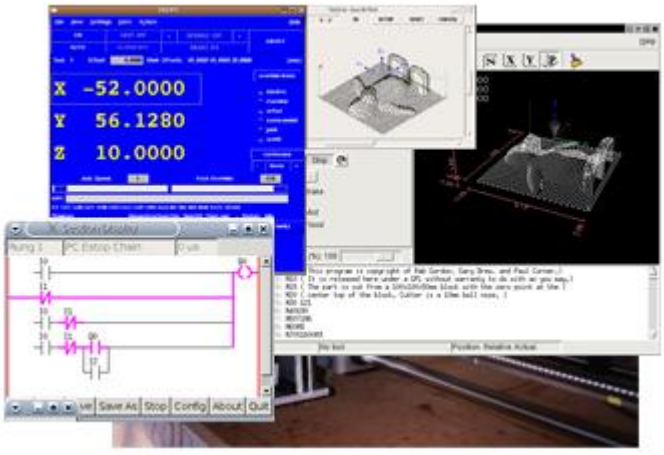

Рисунок 1 – Скриншот работы программы LinuxCNC

1.2.2. Universal Gcode Sender (UGS)

Кроссплатформенное свободное приложение, написанное на Java. Визуализирует процесс обработки детали и как и предыдущее приложение позволяет выполнять работу на реальном станке. Однако, визуализирует только сам процесс выполнения программы, без модели станка. [3]

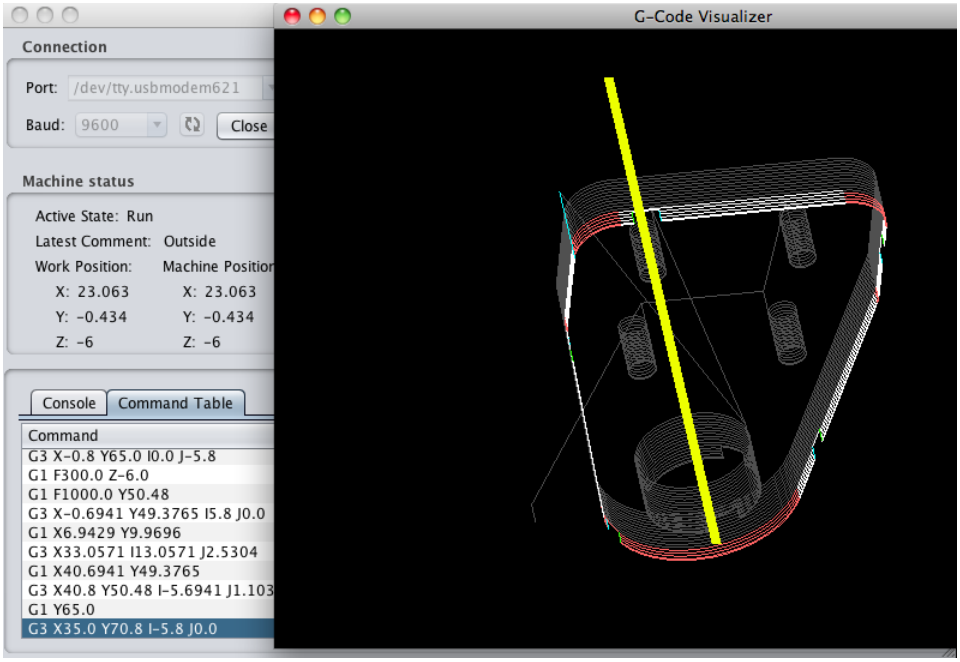

Рисунок 2 – Скриншот работы программы UGS

1.2.3. Autodesk ArtCAM Pro

Autodesk ArtCAM Pro – это программный пакет для пространственного моделирования/механообработки, который позволяет автоматически генерировать пространственные модели из плоского рисунка и получать по ним изделия на станках с ЧПУ. ArtCAM Pro предлагает мощный, легкий в использовании набор средств моделирования, который предоставляет дизайнеру свободу при создании сложных пространственных рельефов. Предоставляется по подписке.

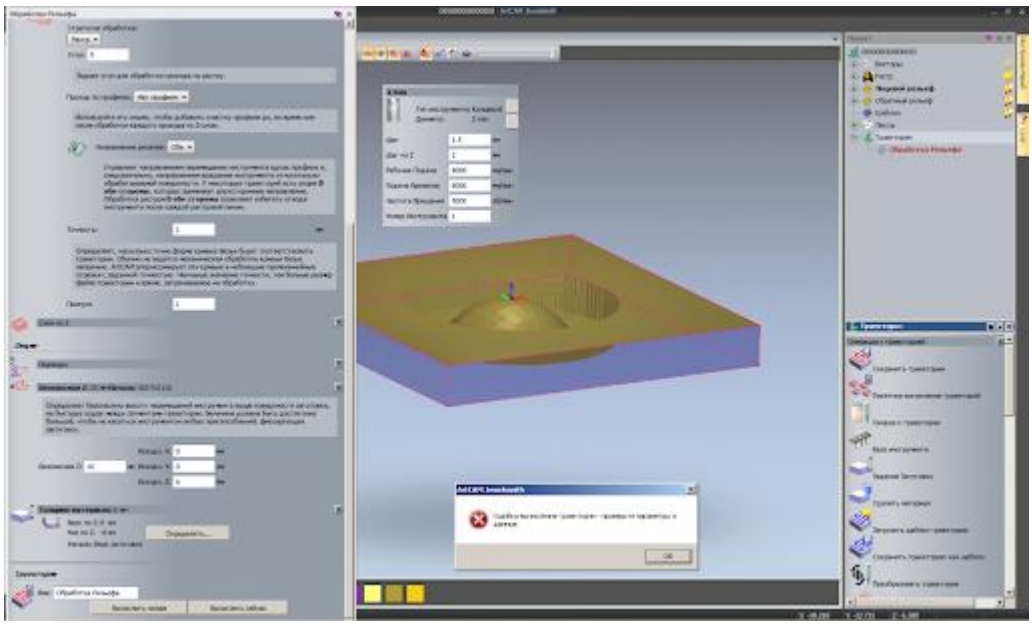

Рисунок 3 – Скриншот работы программы Autodesk ArtCAM Pro

1.2.4. ADEM

Программный комплекс САПР для автоматизации работ промышленного предприятия на этапах конструкторской и технологической подготовки производства. Предоставляется по коммерческой лицензии.

1.2.5. CNC Simulator Pro

Симулятор, позволяющий выполнить полное трехмерное моделирование различных станков с виртуальным контроллером ЧПУ. Распространяется по подписке [4].

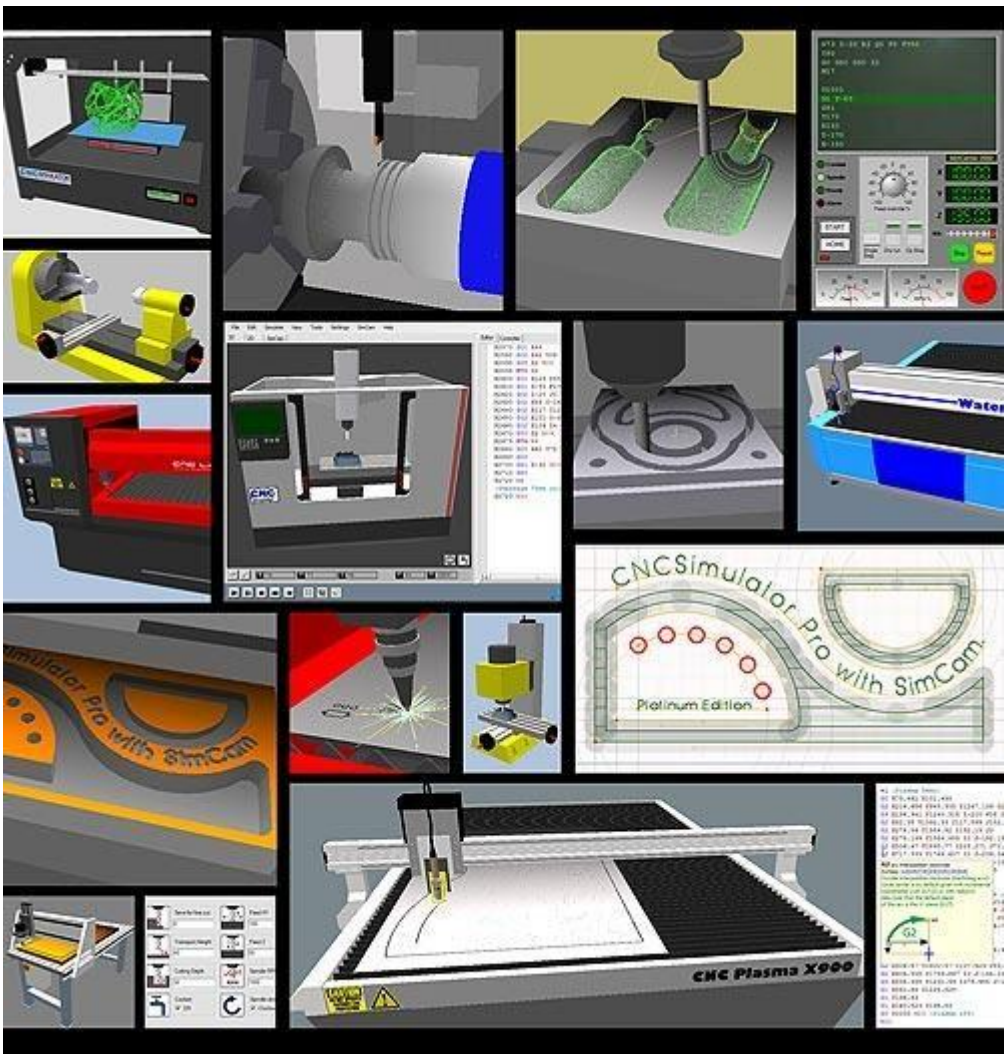

Рисунок 4 – Скриншот работы программы CNC Simulator Pro

1.2.6. Control Simulator

Control Simulator – программное обеспечение от производителей пультов Haas. Обеспечивает запуск управляющей программы на G-кодах и визуализацию процесса фрезерования. Поставляется вместе с пультами Haas [8].

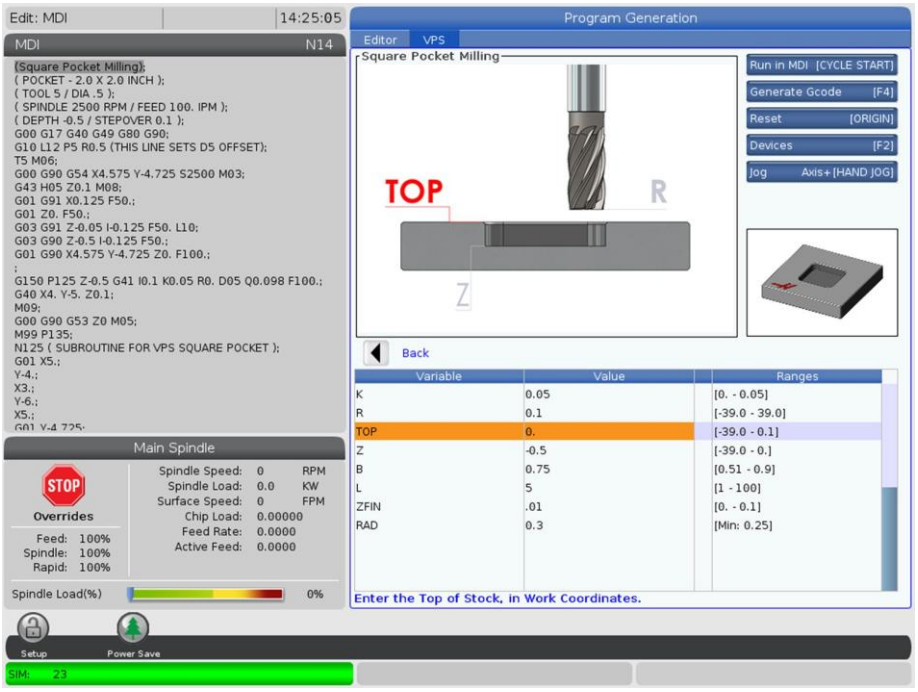

Рисунок 5 – Скриншот работы программы Control Simulator

1.2.7. Итоги обзора аналогов

Подведя итоги, существующие программные продукты не отвечают требованиям, необходимым для данного проекта.

Продукт либо предоставляется по подписке или коммерческой лицензии, что не позволяет его поставить заказчику вместе с оборудованием, либо, если является свободным ПО, визуализирует только процесс обработки детали, без работы непосредственно со станком, а значит не может в полной мере обеспечить отработку навыков, требуемых наладчику и оператору станка.

1.3. СТАНКИ С ЧПУ

Для создания эмулятора фрезерного станка необходимо знать особенности работы реальных фрезерных станков с ЧПУ.

1.3.1. Станочная система координат

Система координат станка с ЧПУ является главной расчетной системой, определяющей перемещения исполнительных органов. Оси координат располагают параллельно направляющим станка, что позволяет при создании УП легко задавать направления и расстояния перемещений.

Правая система координат является стандартной для всех станков с ЧПУ. В этой системе положительные направления координатных осей определяются по правилу «правой руки». Если большой палец указывает положительное направление оси X, указательный – оси Y, то средний укажет на положительное направление оси Z. В качестве положительного направления оси Z принимают вертикальное направление вывода инструмента (например, сверла) из заготовки. То есть ось Z всегда связана со шпинделем станка. Как правило, за X принимают ось, вдоль которой возможно наибольшее перемещение исполнительного органа станка. При этом ось X перпендикулярна оси Z и параллельна плоскости рабочего стола. Если вы определили на станке направления осей X и Z, то по правилу «правой руки» вы однозначно сможете сказать, куда «смотрит» ось Y. Оси X, Y, Z указывают положительные направления перемещений инструмента относительно неподвижных частей станка. [9]

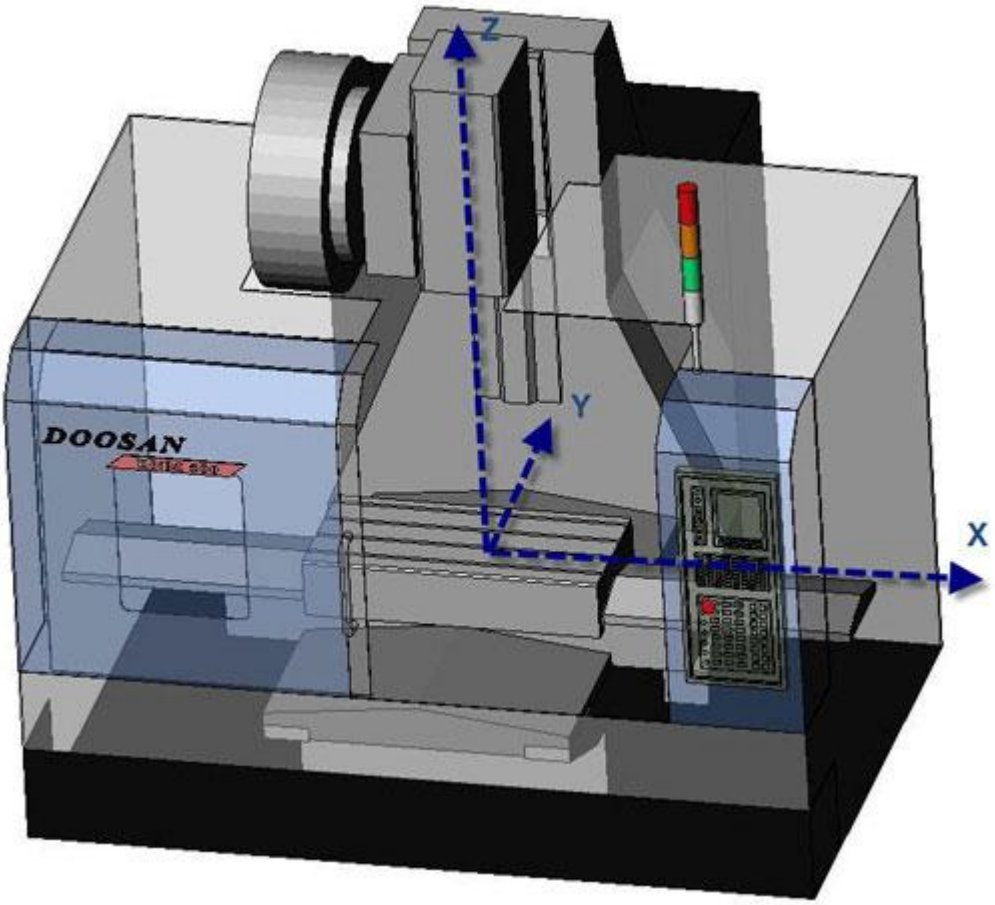

Рисунок 6 – Оси координатной системы фрезерного станка

Положения исполнительных органов характеризуют их базовые точки, которые выбираются с учетом конструкции станка. Например, базовой точкой для шпинделя фрезерного станка с ЧПУ является точка пересечения его торца с собственной осью вращения. Для рабочего стола – точка пересечения его диагоналей или один из углов. [7]

Положение базовой точки относительно начала координат станка с ЧПУ (нулевой точки станка) называется позицией исполнительного органа в системе координат станка или машинной позицией (от англ. *machine* станок). При работе станка в любой момент времени вы можете увидеть на экране стойки ЧПУ текущую машинную позицию (например, рабочего стола) по любой из осей относительно «нуля станка». В документации станка пределы возможных перемещений рабочих органов, как правило, указывают пределами смещений базовых точек. Эти данные являются очень важной

характеристикой станка, так как они определяют максимально возможные габариты обрабатываемой заготовки.

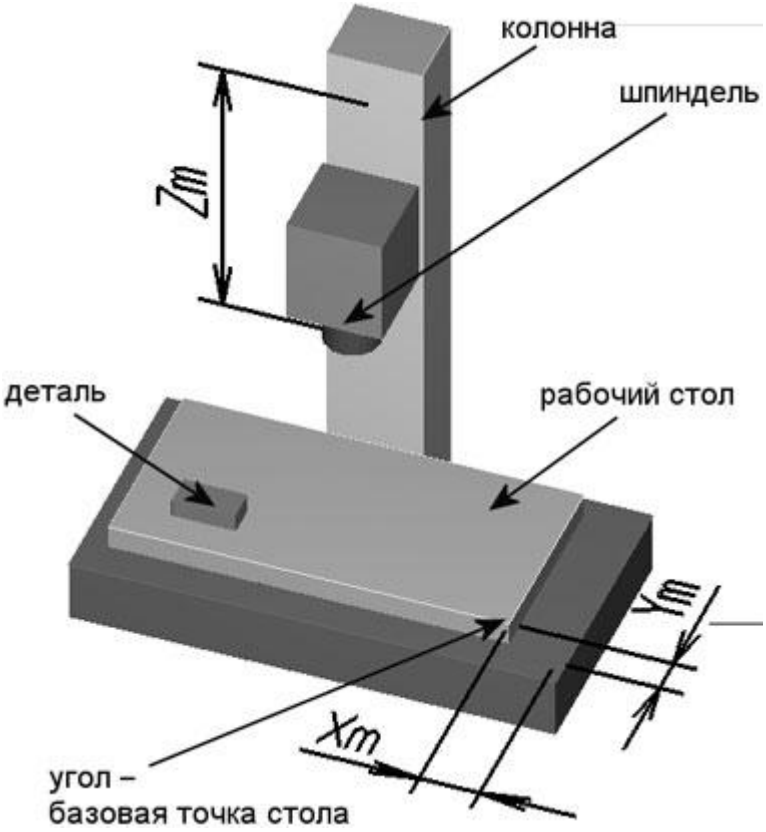

Рисунок 7 – Расстояния Xm, Ym и Zm от нулевой точки станка до базовых точек исполнительных органов определяют машинные позиции

Таким образом, для работы станка необходимо передавать ему координаты, которые контроллер станка будет обрабатывать и перемещать фрезу.

1.3.2. Управляющие программы Gcode

Программирование обработки на современных станках с ЧПУ осуществляется на языке, который обычно называют языком ИСО (ISO) 7 бит, или языком G- и M-кодов. Коды с адресом G, называемые подготовительными, определяют настройку СЧПУ на определенный вид работы. Коды с адресом M называются вспомогательными и предназначены для управления режимами работы станка. [5]

Для данного проекта нам понадобятся команды, указанные в таблице 1.

Таблица 1 – Управляющие команды, используемые в проекте

Название	Пример кадра	Применение	Обозначения переменных в кадре
Ускоренный ход	G0 X0 Y0 Z0	Перемещение на очень высокой скорости в указанную точку. Используется для подвода инструмента к началу траектории.	*(X.., Y.., Z..)* – координата перемещения;
Линейная интерполяция	G1 X0 Y0 Z0 F100	Перемещение по прямой в указанную координату с указанной скоростью. Используется как правило для шлифовки поверхности и создания фасок.	*(X.., Y.., Z..)* – координата перемещения; *F..* – скорость перемещения;
Круговая интерполяция по часовой стрелке	G2 X10. Y20. R10. F100	Перемещение по дуге в плоскости XY по часовой стрелке на указанной скорости подачи. Предназначается как правило для расточки отверстий.	*(X.., Y.., Z..)* – координата перемещения; *F..* – скорость перемещения; *R..* – радиус;
Круговая интерполяция против часовой стрелки	G3 X10. Y20. R10. F100	Перемещение по дуге в плоскости XY против часовой стрелки на указанной скорости подачи.	*(X.., Y.., Z..)* – координата перемещения; *F..* – скорость перемещения; *R..* – радиус;
Абсолютное позиционирование	G90	Координаты отсчитываются от базовой точки станка	
Относительное позиционирование	G91	Координаты отсчитываются от предыдущей позиции	
Смена инструмента	T1 D1 M6	Если в магазине несколько инструментов, головка барабана перемещается таким образом, чтобы активным был инструмент Tn	

Данные команды сохраняются в файле с определенным расширением (разное для каждой модели станка). В данном проекте эти команды будут сохраняться в файл с расширением *.txt*. Файл должен загружаться и парситься в набор координат, которые в дальнейшем будут передаваться модулю фрезерной обработки.[10]

1.3.3. 3 и 5-осевая фрезерная обработка

Фрезерные станки с 3-осевой фрезерной обработкой позволяют только перемещать инструмент в трехмерном пространстве.

5-осевая обработка - это использование ЧПУ для перемещения детали или режущего инструмента по пяти различным осям одновременно. Такая обработка позволяет изготавливать очень сложные детали, и именно поэтому она особенно популярна, например, в аэрокосмической отрасли или машиностроении. [11]

Однако несколько факторов способствовали широкому применению 5-осевой обработке больше всего. Среди них:

1. Максимальная приближенность к принципу – одна обработка за одну установку (иногда называемой «сделано за один раз»), что сокращает время выполнения и повышает эффективность.
2. Удобство доступа к сложным частям геометрии изделия и возможность избежать столкновения с держателем инструмента благодаря возможности наклонять режущий инструмент или стол.
3. Оптимизация и улучшение срока службы инструмента станка и времени цикла обработки. Это достигается путем наклона инструмента/стола, в результате чего поддерживается оптимальное положение и траектория резки.

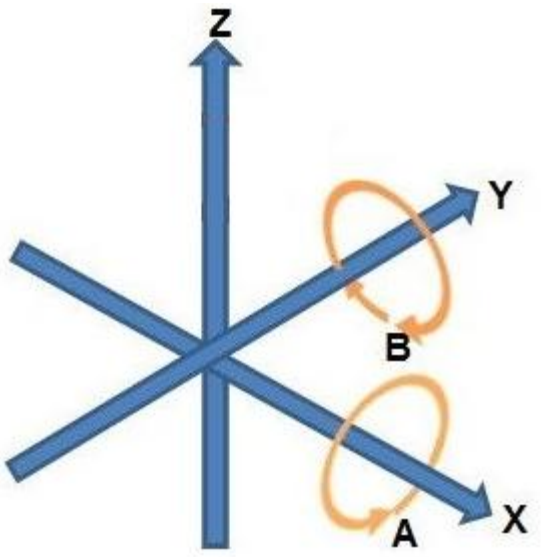

Рисунок 8 – Расположение осей в 5-осевых фрезерных станках

Таким образом, к указанным выше командам может прибавиться еще *G0 A5 B5*, что означает поворот на 5 градусов вокруг осей X и Y соответственно

Для модуля фрезерной обработки это означает, что нам нельзя привязываться к вертикальному/горизонтальному фрезы, так как инструмент может принимать любое положение.

1.4. 3D-МОДЕЛИРОВАНИЕ

Основой 3D-модели является полигональная сетка (чаще её называют меш, от англ. mesh). Это набор из вершин, ребер и граней, которые определяют форму модели [13].

В Unity грани состоят из 3 вершин. В данном проекте нам потребуется регулярно обновлять наш меш, соответственно, он будет генерироваться процедурно.

Для создания модели процедурно с помощью скрипта, необходимо сначала создать массив вершин. Это будет массив объектов типа Vector3 (класса, который хранит 3 числа типа float – соответственно, координаты X, Y и Z).

Для дальнейшего затягивания вершин гранями, необходимо создать массив чисел типа int (целочисленных). Это будут индексы вершин. Число элементов данного массива должно быть кратно 3, так как эти индексы будут считываться игровым движком по 3 штуки и между ними будет натягиваться грань. [12, 14]

Рассмотрим пример. Пусть нам надо начертить квадрат размером 1x1 [18].

Нам необходимо сделать следующее:

1. Создать массив vertices (вершины);

```
Vector3[] vertices = new Vector3[4]
    {
        new Vector3(0, 0, 0),
        new Vector3(1, 0, 0),
        new Vector3(1, 1, 0),
        new Vector3(0, 1, 0),
    };
```

2. Создать массив индексов вершин в правильном порядке;

```
int[] triangles = new int[6] {0, 2, 1, 0, 3, 2};
```

3. Передать оба массива в Mesh.

```
var createdMesh = new Mesh();
    createdMesh.vertices = vertices;
    createdMesh.triangles = triangles;
```

Итоговый результат выглядит так:

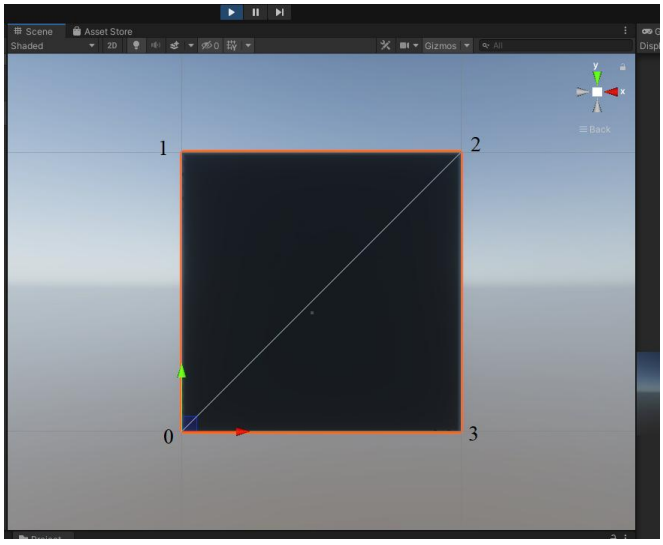

Рисунок 9 – Результат генерации меша

Материал в данном проекте будет использоваться из стандартных ассетов, предоставляемых средой разработки. В данном случае, это Metal_Simple.

Таким образом, основные задачи при разработке модуля фрезерной обработки являются:

- Построить модель заготовки;
- Обновлять точки модели заготовки через какой-то промежуток времени на основании положения резца;
- Строить полигоны в правильной последовательности [19];

1.5. ВЫВОД

Так как для поставленной задачи не существует аналогов, которые бы в полной мере отвечали нынешним потребностям, необходимо создать эмулятор, который бы им соответствовал.

2. ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

2.1. ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

Для определения функциональных требований необходимо выделить типы пользователей данного приложения. Для приложения «Эмулятор фрезерного станка» группу пользователей представляет обучающийся технический персонал.

Определим требования для данной группы [22]:

- интуитивно понятный, не перегруженный интерфейс;
- реализация основных управляющих команд GCode;
- возможность поставить выполнение на паузу;
- детализированная отрисовка получившейся 3d модели;
- реализация нескольких режимов работы станка.

2.2. НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

При постановке нефункциональных требований можно выделить следующие пункты:

- приложение должно работать на платформах Windows и Linux;
- должна быть обеспечена стабильная работа приложения (отсутствие «лагов» во время выполнения программы);
- приложение должно осуществлять правильное преобразование управляющих команд и предупреждать пользователя о попытке запустить программу, выполнение которой невозможно.

2.3. ВЫВОДЫ

В главе 2 был проведен анализ требований к приложению. Определена целевая группа пользователей, которой является обучающийся технический персонал (резчики, наладчики станков с ЧПУ). Выявлены основные функциональные требования для корректной работы приложения.

В функциональные требования входят реализация основных режимов работы фрезерного станка с ЧПУ, детализованная отрисовка 3D-модели, получившейся в результате выполнения обработки, а также реализация основных управляющих команд языка управляющих команд G-code.

Также сформированы нефункциональные требования необходимые для реализации приложения и поддержки необходимого функционала.

К нефункциональным требованиям можно отнести стабильную работу приложения на Windows и Linux, а также удобный интерфейс, позволяющий осуществлять работу на данном эмуляторе.

3. ПРОЕКТИРОВАНИЕ

3.1. АРХИТЕКТУРА ПРИЛОЖЕНИЯ

Приложение состоит из:

- Обработчик интерфейса;
- Парсер загруженной управляющей команды;
- Модуль «Фрезерная обработка»;
- Ядро, контролирующее работу всех компонентов;

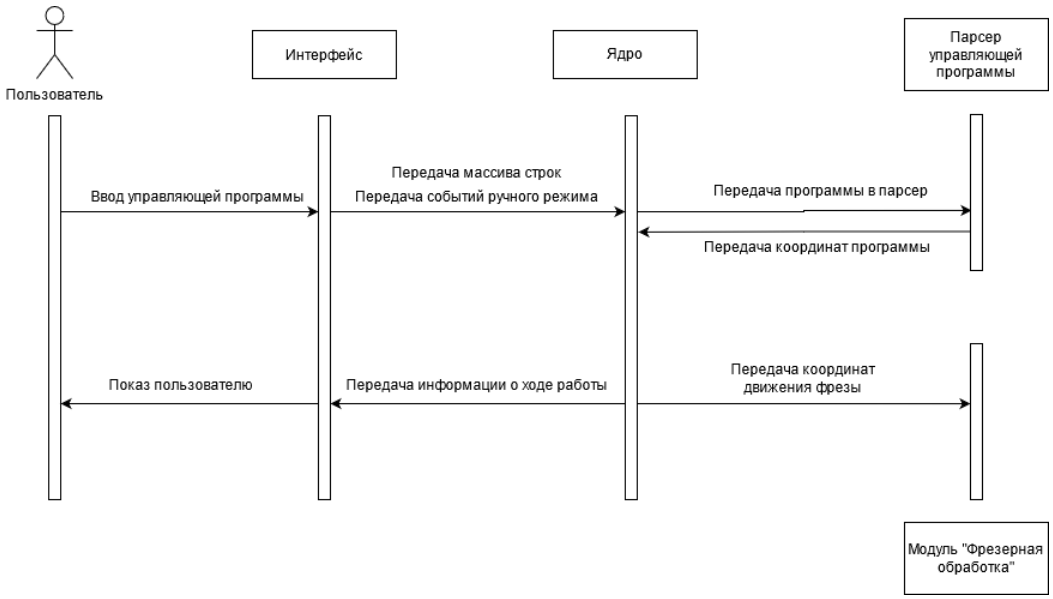

Рисунок 10 – Диаграмма работы приложения

3.2. АЛГОРИТМ РАБОТЫ МОДУЛЯ «ПАРСЕР»

Программы, написанные на G-кодах для токарно-фрезерных станков, могут иметь разный формат записи и количество команд. Также, существуют команды, которые используются только в пультах управления определенных фирм (Sinumerik, Haas, Fanuc).

В данной работе для моделирования работы станка с ЧПУ, было взято несколько команд перемещения фрезы, а также несколько управляющих команд. Впоследствии, данный список может расширяться.

Список управляющих команд указан в п. 1.3.2.

Так как существуют команды круговой интерполяции, одна строка файла управляющей программы может генерировать массив координат. Для этого необходимо сначала интерпретировать файл в промежуточный формат строк, а затем сгенерировать массив координат.

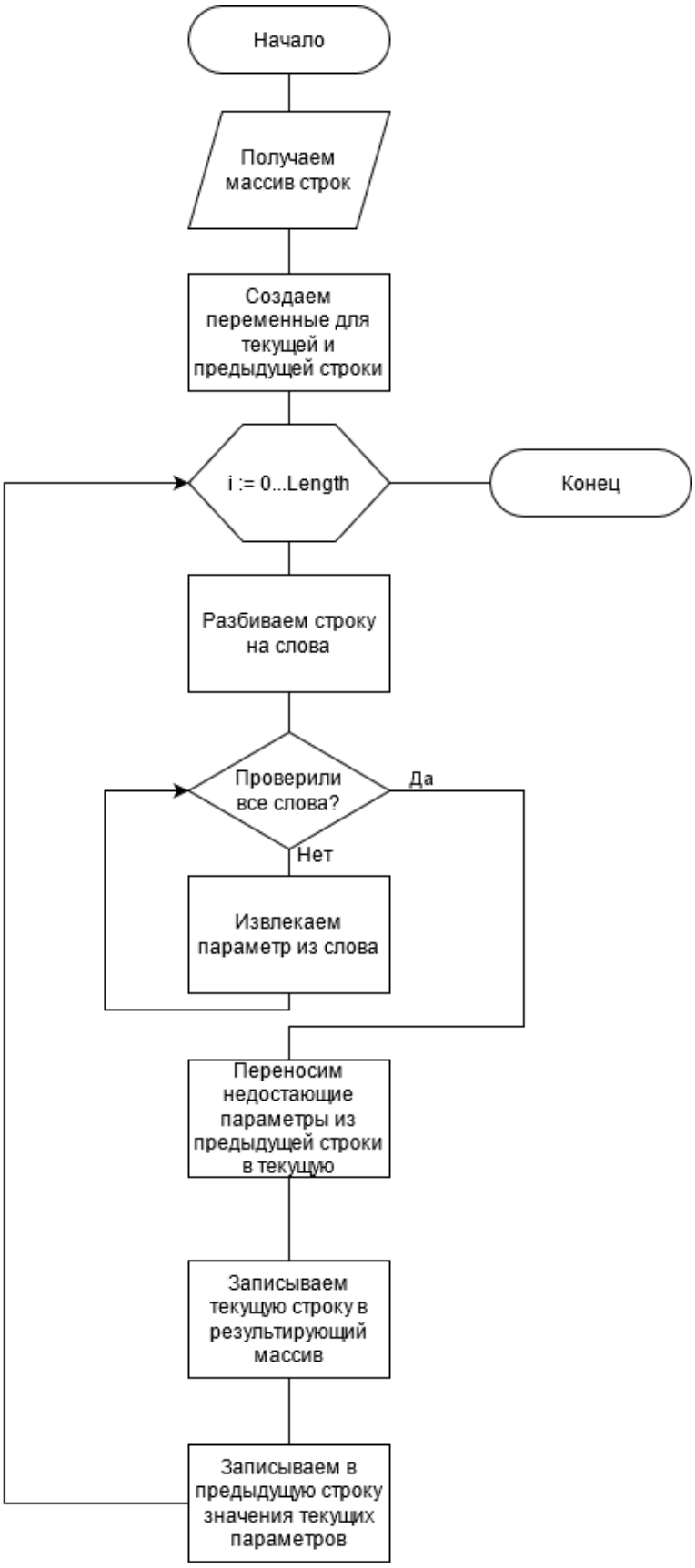

Рисунок 11 – Блок-схема работы интерпретатора файла в промежуточные команды

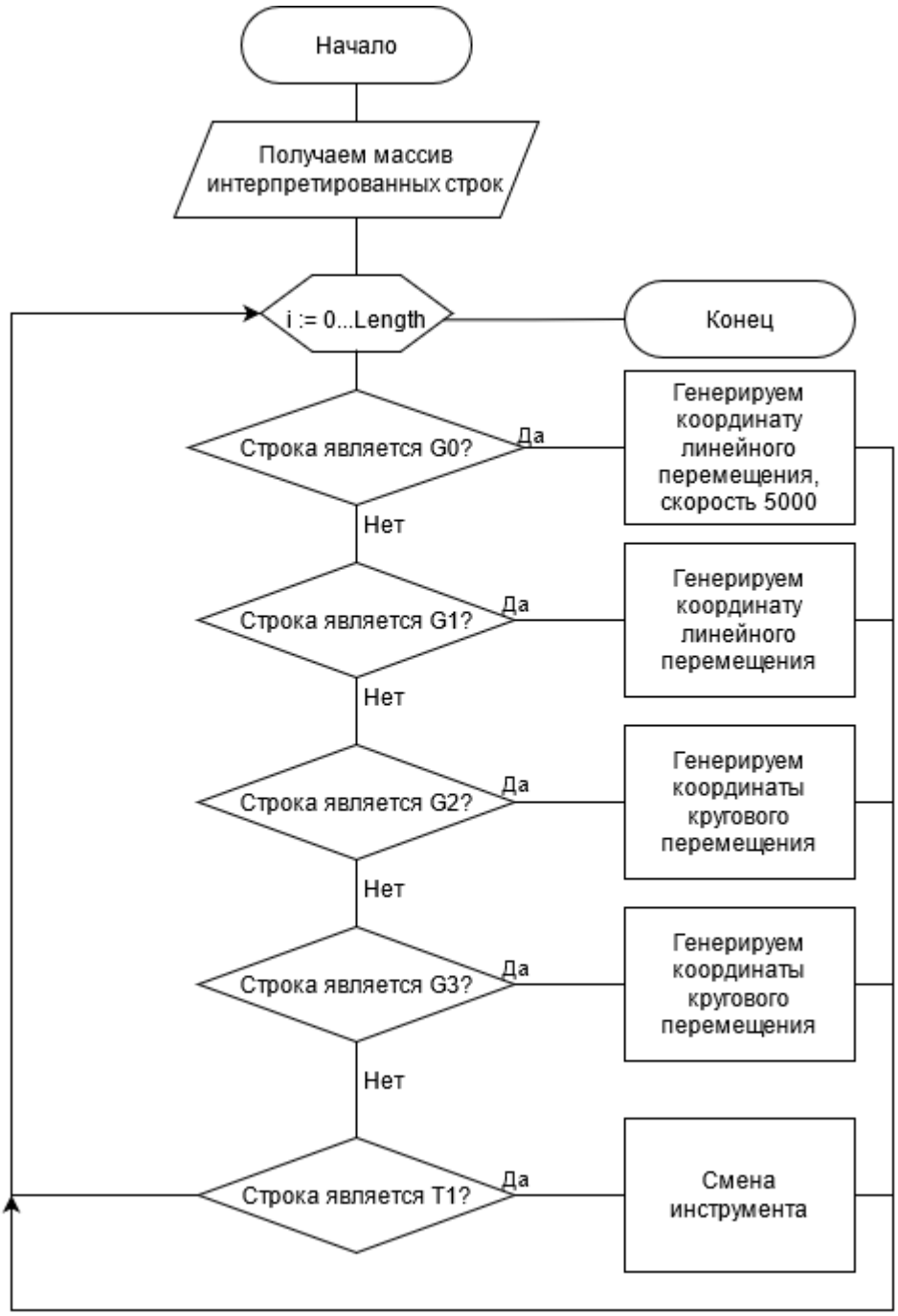

Рисунок 12 – Блок-схема работы парсера по преобразованию промежуточных команд в конечные координаты

3.3. АЛГОРИТМ РАБОТЫ МОДУЛЯ «ФРЕЗЕРНАЯ ОБРАБОТКА», РЕАЛИЗОВАННЫЙ С ПОМОЩЬЮ MARCHING CUBES

3.3.1. Принцип работы алгоритма Marching Cubes

Основная проблема состоит в том, чтобы сформировать приближение к изоповерхности через скалярное поле, дискретизированное на прямоугольной трехмерной сетке. Учитывая одну ячейку сетки, определяемую ее вершинами и скалярными значениями в каждой вершине, необходимо создать плоские фасеты, которые наилучшим образом представляют изоповерхность через эту ячейку сетки. Изоповерхность может не проходить через ячейку сетки, она может отрезать любую из вершин или проходить через нее любым из ряда более сложных способов. Каждая возможность будет характеризоваться количеством вершин, которые имеют значения выше или ниже изоповерхности. Если одна вершина находится выше изоповерхности, а соседняя вершина находится ниже изоповерхности, то мы знаем, что изоповерхность разрезает ребро между этими двумя вершинами. Положение, в котором он обрезает край, будет линейно интерполировано, отношение длины между двумя вершинами будет таким же, как отношение значения изоповерхности к значениям в вершинах ячейки сетки.

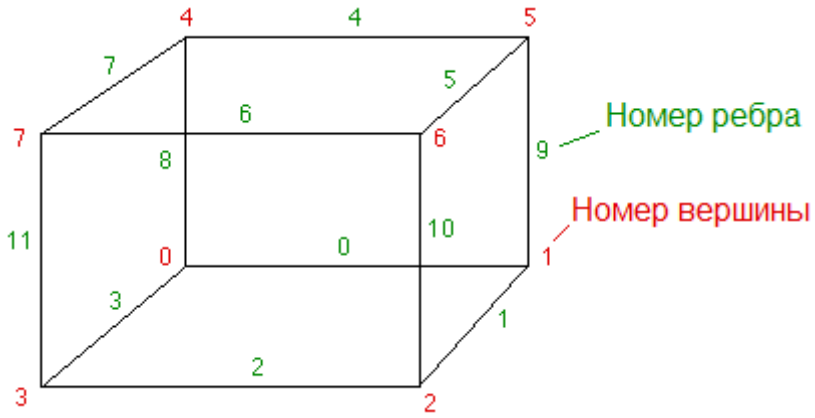

Рисунок 13 – Пример единичного вокселя

Представим, что, значение в вершине 3 ниже значения изоповерхности, а все значения во всех других вершинах были выше значения изоповерхности. Тогда нам необходимо создать треугольную грань, которая пересекает ребра 2, 3 и 11. Точное положение вершин треугольной грани зависит от отношения значения изоповерхности к значениям в вершинах 3-2, 3-0, 3-7 соответственно.[16]

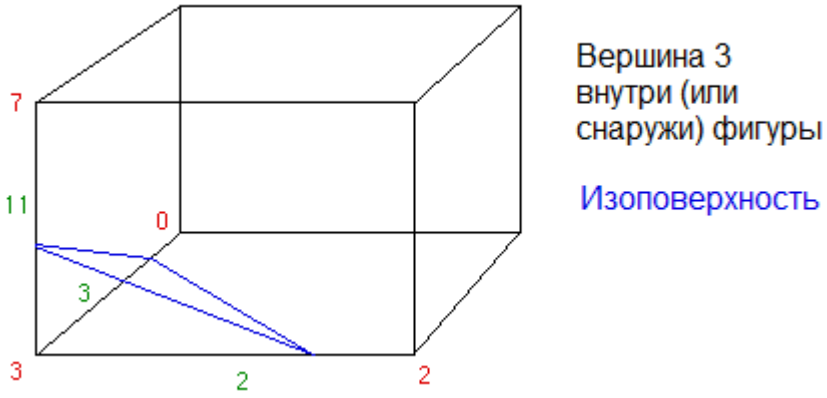

Рисунок 14 – Пересечение клетки (вокселя) изоповерхностью

Существует несколько способов ставить точки пересечения изоповерхности с ребрами клетки:

– Ставить посередине ребра. В таком случае, генерируемая модель будет выглядеть угловатой, но это можно исправить, изменив размер клетки;
– Рассчитывать точку пересечения с помощью интерполяции. Это позволяет снизить количество клеток и получить вполне приемлемое разрешение.

Тем не менее, даже используя интерполяцию, необходимо поддерживать достаточное количество клеток.

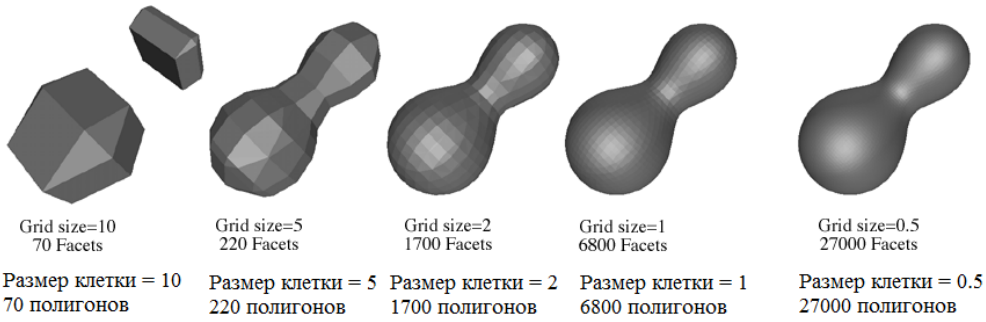

Рисунок 15 – Улучшение детализации при увеличении количества клеток

В нашем случае, изоповерхностью будет являться цилиндрическая или конусная поверхность фрезы. Сетка клеток, из которых состоит заготовка, должна обновляться с заданной периодичностью.

3.3.2. Реализация алгоритма в Unity

Большинство реализаций данного алгоритма были сделаны на C++ или Python. Тем не менее, существует несколько реализаций Marching Cubes в Unity. Все они используются для процедурной генерации ландшафта местности с заданными параметрами, поэтому для полноценной реализации необходимо внести ряд изменений.[19]

В данной работе за основу реализации алгоритма был выбран проект «Marching-Cubes-Terrain» от пользователя Eldemarkki (https://github.com/Eldemarkki/Marching-Cubes-Terrain). Данный проект позволяет вырезать из сетки вокселей различные фигуры, что концептуально похоже на процесс фрезерования и обработки детали.[15]

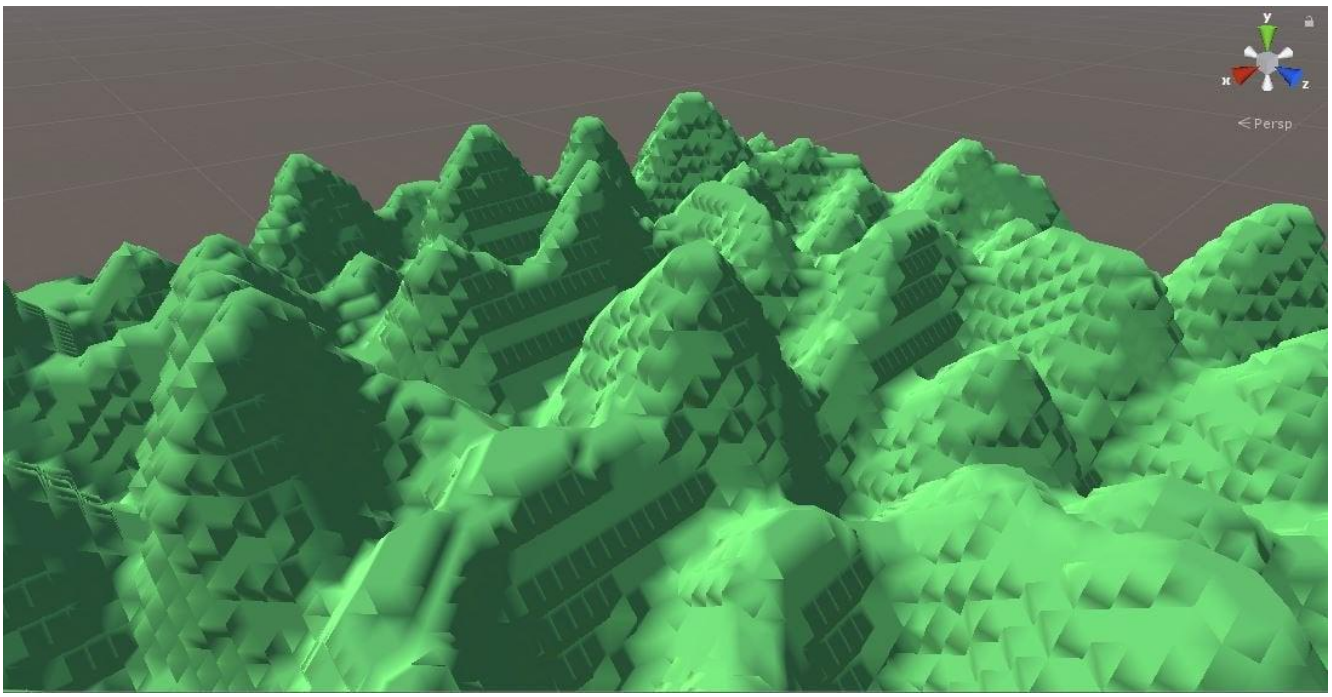

Рисунок 16 – Выпуклости и впадины, созданные с помощью Marching Cubes

3.3.3. Алгоритм работы модуля «Фрезерная обработка»

В существующем проекте необходимо сделать ряд значительных изменений:

- точки ландшафта генерируются процедурно. Необходимо, чтобы в самом начале был сгенерирован конечный массив точек заготовки;
- ландшафт не имеет толщины и является плоской поверхностью. Необходимо, чтобы заготовка была объемной;
- изменение ландшафта осуществляется путем пускания луча из курсора мыши. Необходимо, чтобы заготовку менял объект «Фреза» (цилиндр).

3.3.4. Алгоритм работы модуля «Фрезерная обработка»

Обработчик должен осматривать все клетки, которые находятся в пределах границ резца и проверять, входят ли они в его границы.

Таким образом, в плоскости XY нам необходимо проверять все точки, отвечающие условию:

$$\forall(x, y) \rightarrow (x - x_c)^2 + (y - y_c)^2 \leq R^2, \text{где } (x_c, y_c) - \text{ центр фрезы}$$

В плоскости фрезы нам необходимо проверять точки:

$$\forall z \rightarrow z_c - \frac{h}{2} \leq z \leq z_c + \frac{h}{2}, \text{где } z_c - \text{ центр фрезы; } h - \text{длина фрезы}$$

У каждой точки, отвечающей данному условию вызывается функция SetVoxelDataCustom, которая передает обработчику массива клеток точку пересечения этой клетки с изоповерхностью фрезы. Обработчик преобразовывает меш для каждой клетки и повторяет цикл заново.

3.4. АРХИТЕКТУРА ОБРАБОТЧИКА ИНТЕРФЕЙСА

Необходимо, чтобы станок работал в 3 режимах: ручной режим, режим ручного ввода данных MDI и автоматический режим.[17]

3.4.1. Ручной (толчковый) режим.

В этом режиме осуществляется перемещение исполнительных органов станка при помощи ручного генератора импульсов, который похож на пульт дистанционного управления или при помощи специальных маховиков на панели УЧПУ. Оператор станка может задавать шаг и направление перемещения при помощи специальных переключателей.

Для реализация данного режима необходимы следующие элементы интерфейса:

- кнопки выбора оси перемещения;
- кнопки выбора направления (вперед-назад);
- кнопка возврата в нулевую точку детали;
- кнопка включения ускоренного перемещения;
- переключатель оборотов шпинделя;
- переключатель подачи шпинделя.

3.4.2. Режим ручного ввода данных MDI.

Режим ручного ввода данных MDI позволяет оператору ввести и выполнить один или несколько кадров, не записанных в памяти СЧПУ. Обычно этот режим используется для ввода отдельных G- и M-кодов, например для смены инструмента или включения оборотов шпинделя. Введенные команды и слова данных после выполнения или сброса удаляются.

Для реализация данного режима необходимы следующие элементы интерфейса:

- поле ввода команды;
- кнопка добавления строки;
- кнопка удаления строки;
- кнопка запуска управляющей программы;
- кнопка паузы управляющей программы;
- кнопка остановки управляющей программы.

3.4.3. Автоматический режим.

Позволяет выполнять программу обработки прямо из компьютера или другого внешнего устройства, не записывая ее в память системы. Обычно в этом режиме выполняются УП большого размера, которые не могут поместиться в памяти СЧПУ.

Для реализация данного режима необходимы следующие элементы интерфейса:

- кнопка загрузки программы;
- поле просмотра загруженной программы;
- кнопка запуска управляющей программы;
- кнопка паузы управляющей программы;
- кнопка остановки управляющей программы.

3.5. ВЫВОД

В главе 3 было произведено проектирование приложения.

Был подобран необходимый алгоритм генерации и редактирования 3D модели и описана структура этого алгоритма. Была описана существующая реализация данного алгоритма на Unity и определен ряд требований к изменению данной реализации.

Были описаны методы преобразования файлов управляющих программ станков с ЧПУ. Были составлены блок-схемы алгоритма парсинга этих управляющих команд из строкового формата в формат координат, которые передаются приложению для перемещения резца.

Также были сформированы необходимые функции для корректного составления пользовательского интерфейса для 3 режимов работы станка: ручной, ручной ввод данных и автоматический.

4. РЕАЛИЗАЦИЯ

4.1. РЕАЛИЗАЦИЯ ИНТЕРФЕЙСА

При запуске приложения пользователю сразу открывается окно с рабочей областью станка.

Интерфейс управления инструментом находится справа [20]. На нем расположена область переключения режимов работы (область 1) на рисунке 17:

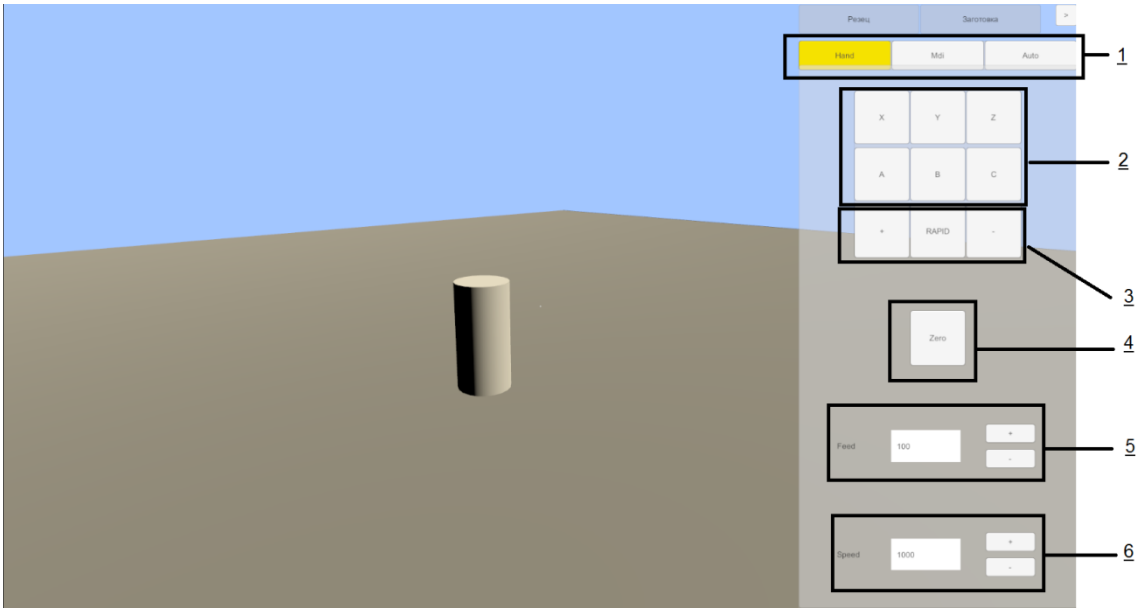

Рисунок 17 – область смены режима и панель ручного режима.

4.1.1. Интерфейс ручного режима

Скриншот панели ручного режима представлен на рисунке 16.

На рисунке 17 выделены следующие области:

1 – переключение режимов работы;

2 – выбор оси;

3 – кнопка ускоренного перемещения и кнопки выбора направления перемещения;

4 – кнопка возврата инструмента в нулевую точку;

5 – установка подачи инструмента;

6 – установка скорости вращения шпинделя.

Для перемещения инструмента в ручном режиме необходимо выбрать ось, установить значения подачи и вращения шпинделя и долгими нажатиями на кнопки «+» и «-» подвести инструмент в искомую точку, как это показано на рисунке 18.

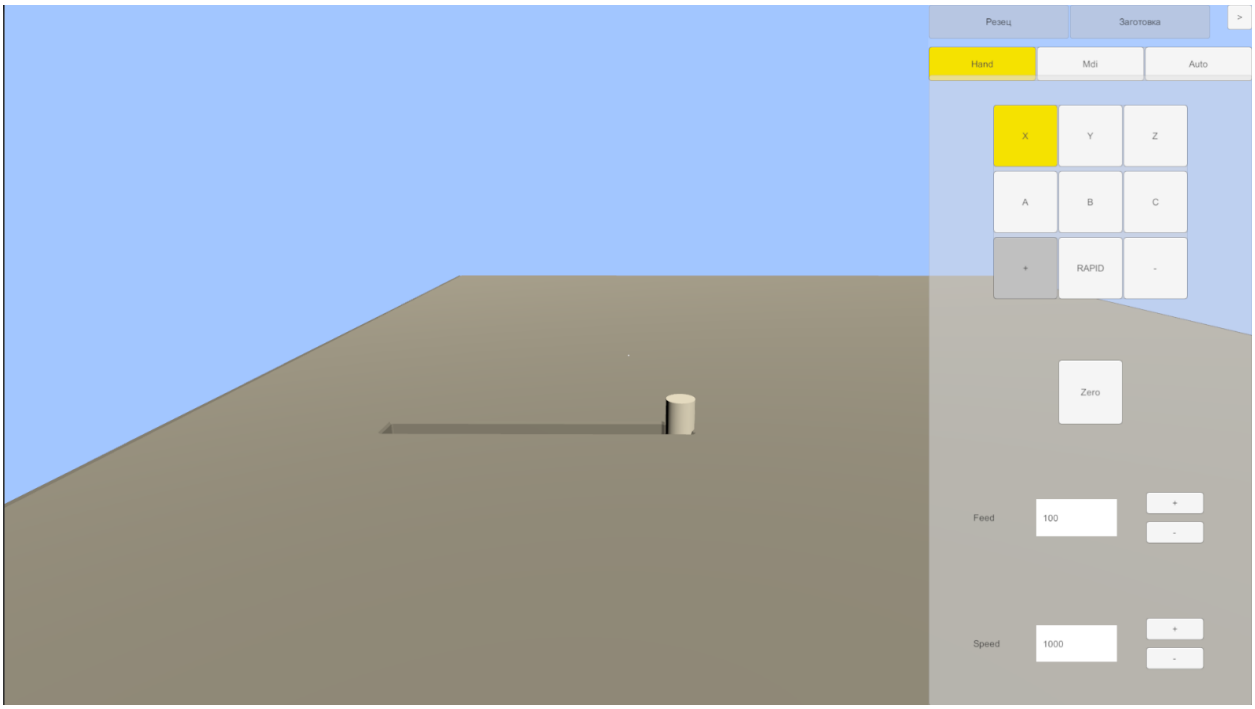

Рисунок 18 – Ручной режим, процесс работы

4.1.2. Интерфейс режима ручного ввода данных

Скриншот панели ручного ввода данных представлен на рисунке 19.

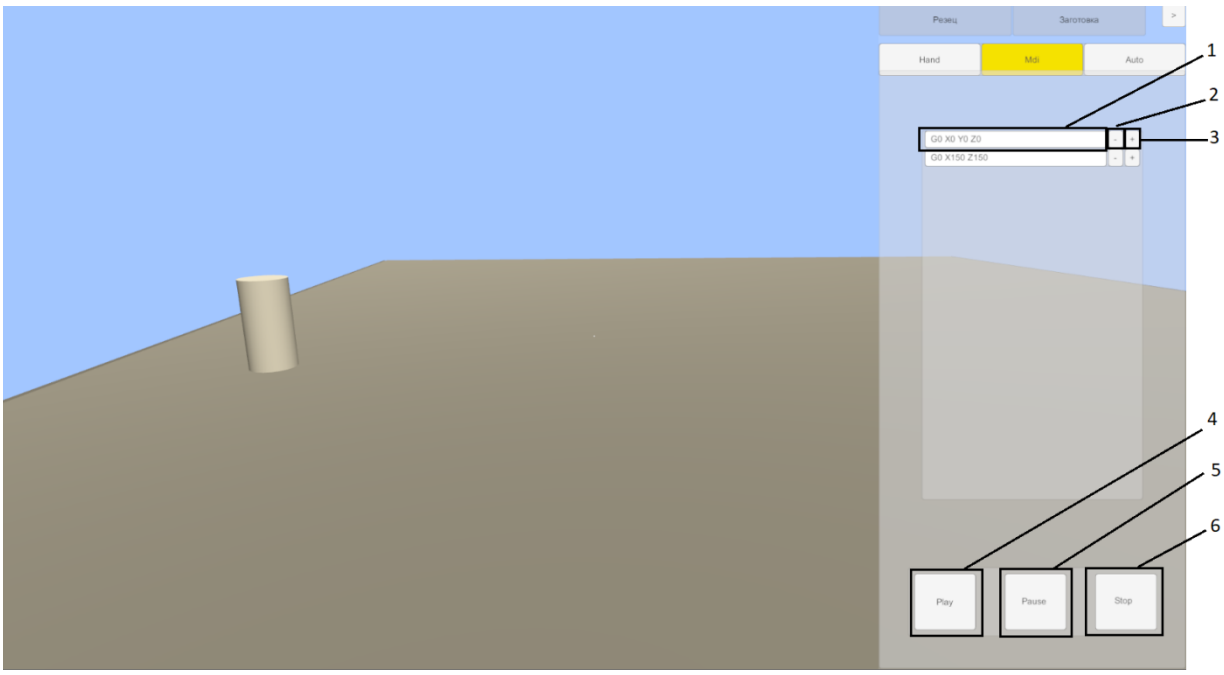

Рисунок 19 – Режим ручного ввода данных

На рисунке 19 выделены следующие области:

1 – поле ввода управляющей команды;

2 – кнопка удаления текущего поля ввода. Если удалить все поля, то создастся одно пустое поле;

3 – кнопка добавления поля ввода после активного. Таким образом, можно вставить строку как в конец программы, так и в середину;

4 – кнопка запуска выполнения введенной управляющей программы;

5 – кнопка паузы;

6 – остановка выполнения программы.

4.1.3. Интерфейс автоматического режима

Скриншот панели автоматического режима представлен на рисунке 20.

Рисунок 20 – Панель автоматического режима

На рисунке 20 выделены следующие области:

1 – кнопка загрузки программы. По нажатию на данную кнопку открывается окно вызова OpenFileDialog (рисунок 21), которое позволяет выбрать файл следующих расширений:

– .txt;

– .doc;

– .nc;

– .gcode.

После выбора файла пользователем, программа данный файл считывает данный файл, удаляет пустые строки и выводит его в поле 2.

2 – поле строк программы. Строки выводятся в формате обычных кнопок, так как это позволяет осуществлять быструю подсветку выполняемой строки во время выполнения программы (см. рисунок 22).

Кнопки старта программы, паузы и остановки аналогичны тем же кнопкам в режиме ручного ввода данных, за исключением того, что программа поступает не из полей ввода, а из файла.

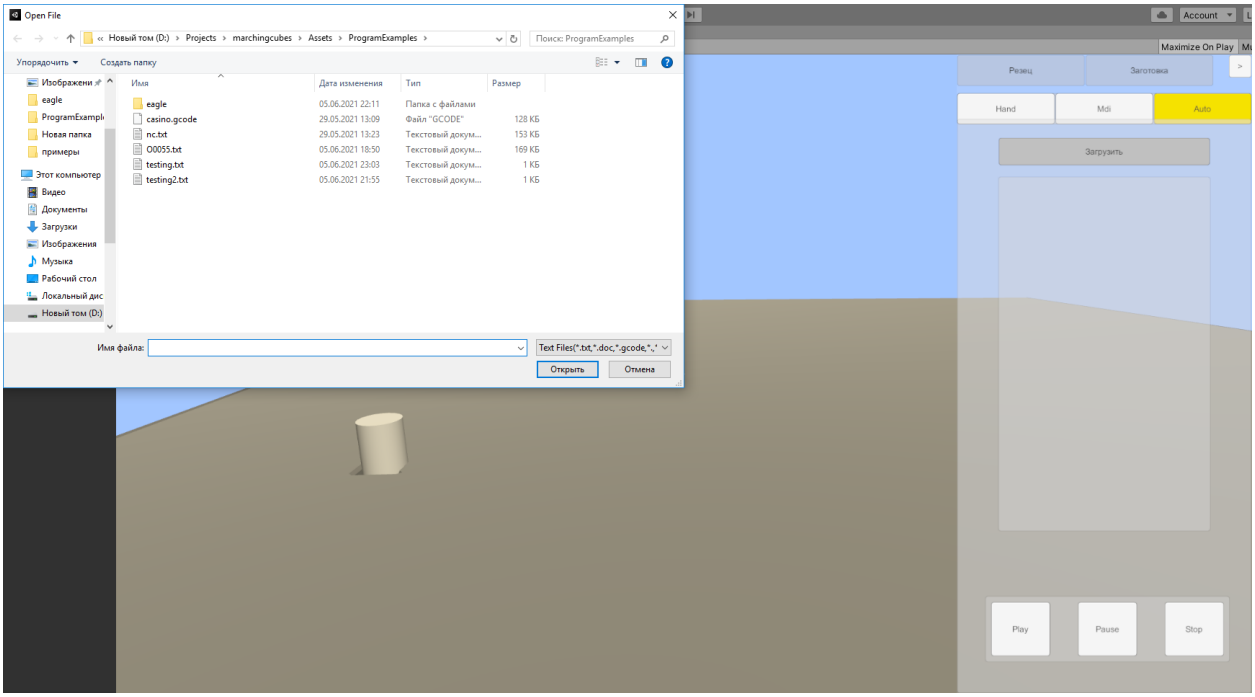

Рисунок 21 – Процесс выбора программы

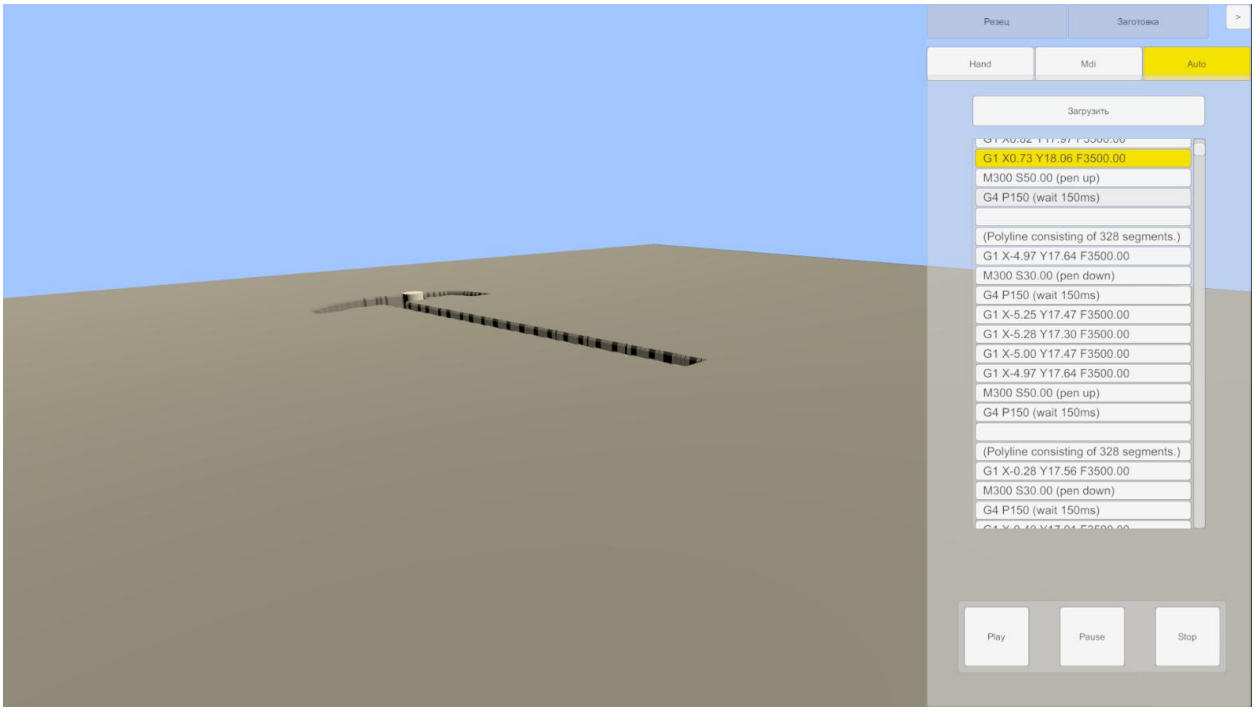

Рисунок 22 – Подсветка линий программы во время обработки

4.2. РЕАЛИЗАЦИЯ АЛГОРИТМА ОБРАБОТКИ

Логика приложения написана на языке C#.

Приложение состоит из одной сцены.

На сцене находятся следующие объекты (рисунок 23):

- Main Camera;
- Directional Light;
- Canvas;
- EventSystem;
- VoxelBillet;
- CutterCylinder;
- SceneController.

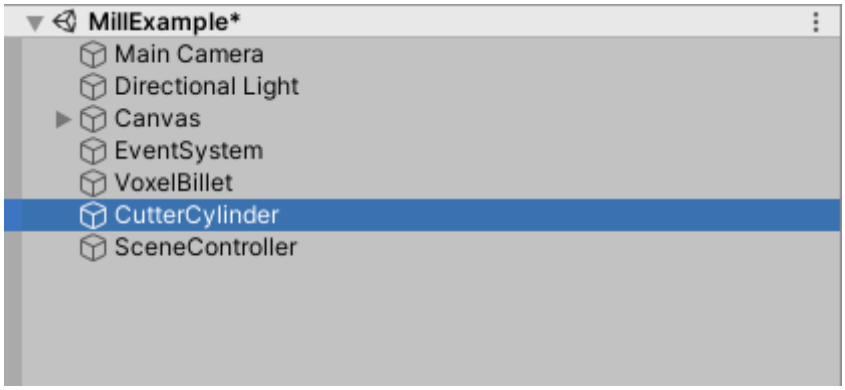

Рисунок 23 – Объекты на сцене

Назначения объектов:

- Main Camera – отображает на дисплей пользователя происходящий процесс обработки. Также имеет компонент PlayerCamera.cs, который позволяет перемещаться по сцене при нажатой клавише Ctrl кнопками W,A,S,D, а также вращать камеру мышью.
- Directional Light – обеспечивает освещение заготовки;
- Canvas – пользовательский интерфейс с панелями управления;

– EventSystem – отвечает за обработку событий пользовательского интерфейса;
– VoxelBillet – объект заготовки, построенный из вокселей. Имеет компонент ChunkUpdate, который каждый кадр проверяет, не нужно ли поменять меш заготовки;
– CutterCylinder – объект резца. Имеет компонент CutterDeformer, который отвечает за текущие границы резца и взаимодействие с ChunkUpdate;
– SceneController – контроллер сцены, передающий команды от Canvas к CutterCylinder и обеспечивающий перемещение резца во всех 3 режимах работы.

1. Запуск и инициализация приложения

При запуске в приложении инициализируются объекты типа VoxelWorld и SceneController.

VoxelWorld запускает создание заготовки в HeightmapWorldGenerator. В WorldSettings.cs можно указать размер клетки, который будет использован при расчёте. Чем меньше размер клетки, тем больше клеток будет создано, следовательно, тем более детализированным будет процесс обработки. Однако, если будет указан слишком маленький размер клетки, приложение будет обрабатывать слишком долго, поэтому, необходимо соблюдать баланс между качеством и производительностью.

Обработка размера ChunkSize и создание меша указаны в листинге 1.

Листинг 1. – Создание меша.

```
public override JobHandleWithData<IMesherJob>
CreateMesh(VoxelDataStore voxelDataStore, VoxelColorStore
voxelColorStore, int3 chunkCoordinate)
        {
            if (!voxelDataStore.TryGetDataChunk(chunkCoordinate, out
VoxelDataVolume<byte> boundsVoxelData))
            {
```

```
                return null;
            }

            if (!voxelColorStore.TryGetDataChunk(chunkCoordinate, out
VoxelDataVolume<Color32> boundsVoxelColors))
            {
                return null;
            }

            NativeCounter vertexCountCounter = new
NativeCounter(Allocator.TempJob);

            int voxelCount = VoxelWorld.WorldSettings.ChunkSize.x *
VoxelWorld.WorldSettings.ChunkSize.y *
VoxelWorld.WorldSettings.ChunkSize.z;
            int maxLength = 15 * voxelCount;

            NativeArray<MeshingVertexData> outputVertices = new
NativeArray<MeshingVertexData>(maxLength, Allocator.TempJob);
            NativeArray<ushort> outputTriangles = new
NativeArray<ushort>(maxLength, Allocator.TempJob);

            MarchingCubesJob marchingCubesJob = new MarchingCubesJob
            {
                VoxelData = boundsVoxelData,
                VoxelColors = boundsVoxelColors,
                Isolevel = Isolevel,
                VertexCountCounter = vertexCountCounter,

                OutputVertices = outputVertices,
                OutputTriangles = outputTriangles
            };

            JobHandle jobHandle = marchingCubesJob.Schedule();

            JobHandleWithData<IMesherJob> jobHandleWithData = new
JobHandleWithData<IMesherJob>
            {
                JobHandle = jobHandle,
                JobData = marchingCubesJob
            };

            return jobHandleWithData;
        }
```

Далее VoxelWorld запускает ChunkUpdater, который каждый кадр проверяет, не входит ли объект CutterCylinder в массив клеток и не нужно ли изменять воксели.

Далее запускается SceneController. Он последовательно инициализирует панели режимов (ручного, ручного ввода данных и автоматического). Все события, приходящие с этих панелей, приходят в SceneController, где обработчик вызывает соответствующий метод в CutterCylinder. Так, запуск программы в автоматическом режиме запускает поток перемещения резца по сцене. В случае, если резец при этом касается сетки вокселей заготовки, эта сетка меняется.

2. Обработка заготовки

Обработка заготовки происходит каждый кадр в методе EditBillet() у объекта CutterDeformer. Тело метода указано в листинге 2.

Листинг 2 – метод EditBillet()

```
    private void EditTerrain()
    {
        int buildModifier = -1;

        currentPositionBoundsInt = new int3((int)
(transform.position.x - cutterRadius / 2), (int) (transform.position.y
- cutterHeight / 2f), (int) (transform.position.z - cutterRadius /
2));
        BoundsInt queryBounds = new
BoundsInt(currentPositionBoundsInt.ToVectorInt(), new Vector3Int((int)
cutterRadius, (int) cutterHeight, (int) cutterRadius));
        //TerrainDeformer.RedrawDebug(queryBounds);
        voxelWorld.VoxelDataStore.SetVoxelDataCustom(queryBounds,
                (voxelDataWorldPosition, voxelData) =>
                {
                    float distance =
math.distance(voxelDataWorldPosition.xz, currentPositionBoundsInt.xz);
                    //if (distance <= range)
                    if (distance <= cutterRadius)
                    {
                        float modificationAmount = buildModifier;
                        float oldVoxelData = voxelData / 255f;
                        return (byte)math.clamp((oldVoxelData -
modificationAmount) * 255, 0, 255);
                    }

                    return voxelData;
```

```
            });
    }
```

Переменная buildModifier в данном случае не используется, однако она показывает, что мы удаляем части заготовки, а не создаем новые.

Переменная queryBounds типа BoundsInt хранит текущее положение резца и его границы, то есть, параллелепипед, описанный вокруг резца. Эта переменная позволяет определить все воксели (клетки), которые нужно проверить в в методе SetVoxelDataCustom.

После этого в методе происходит вызов voxelWorld.VoxelDataStore.SetVoxelDataCustom. Первым параметром передается queryBounds для дальнейшего определения проверяемых вокселей, а вторым – делегат метода проверки этих вокселей. В данном случае, мы проверяем все воксели, которые лежат внутри цилиндра по формулам, указанным в п. 3.3.4.

В данном случае, плоскость цилиндра лежит не в плоскости XY, а в плоскости XZ. Это происходит из-за разницы в системах координат: в станках используется правая система координат, а в Unity – левая. Таким образом, ось X остается неизменной, а оси Y и Z меняются местами.

3. Перемещение в автоматическом режиме

Для перемещения в автоматическом режиме необходимо преобразовывать текстовый файл в набор координат.

Парсинг текстового файла начинается с преобразования в промежуточный формат Line, в котором указан тип команд, координата, номер резца и скорости подачи. Метод парсинга указан в листинге 3.

Листинг 3 – преобразование текстового файла в список Line

```
public struct Line
{
    public int N;
    public string Text;
```

```
    public InstructionType type;
    public int typeNum;


    public Vector3 targetCoordinate;
    public bool xChanged, yChanged, zChanged; //флаг задания
координаты
    public float u, v, w;
    public float a, b, c;
    public bool aChanged, bChanged, cChanged;

    public float i, j, k;
    public bool centerChanged; //флаг смены i, j или k
    public float R, F, S;
    public bool rChanged, fChanged, sChanged; //флаг задания радиуса
дуги, скорости или оборотов
    public bool tChanged;

    public int T, D, H;
}


public static List<Line> GetLineListFromProgram(string[] text)
    {
        var lines = new List<Line>();

        var line = GetStartCoordinate();
        var previousLine = line;

        for (int i = 0; i < text.Length; i++)
        {
            //убираем комментарии из строки
            if (text[i].IndexOf(";") != -1)
                text[i] = text[i].Remove(text[i].IndexOf(";"));
            if (text[i].IndexOf("/") != -1)
                text[i] = text[i].Remove(text[i].IndexOf("/"));

            if (text[i].Contains("("))
            {
                while (text[i].Contains("("))
                {
                    var openBracket = text[i].IndexOf('(');
                    var closeBracket = text[i].IndexOf(')');
                    if (closeBracket != -1)
                        text[i] = text[i].Remove(openBracket,
closeBracket - openBracket + 1);
                    else
                        text[i] = text[i].Remove(openBracket);
                }
                if (text[i].Contains(")") && !text[i].Contains("("))
                    text[i] = text[i].Replace(")", String.Empty);
```

```
            }
            text[i] = text[i].Trim();
            if (text[i] == "\r" || text[i] == "")
                continue;


            line = ParseLine(text[i]);

            //проверяем, строка g/m-код или просто строка с
параметрами
            if (line.type == InstructionType.UnknownType)
                line = CheckType(line, previousLine, i); //определяем
тип инструкции, если он не указан явно
            line.targetCoordinate = CheckChangedCoordinates(line,
previousLine); //если какая-то из координат не поменялась, то для нее
оставляем предыдущее значение

            lines.Add(line);
            previousLine = line;
        }
        return lines;
    }
```

Дальше преобразуем список промежуточного формата в список координат и номера резца, которым мы доезжаем в эту координату (для того, чтобы потом можно было менять резцы). Определим конечный формат данных MoveInfo:

Листинг 4 – формат данных MoveInfo:

```
public struct MoveInfo
{
    public Vector3 TargetPoint; //точка, в которую едем
    public float Speed; //скорость
    public int CutterNumber; //номер резца в таблице магазина
    public int LineNumber; //линия программы (нужна для подсветки)

    public MoveInfo(Vector3 point, float speed, int cutter, int
lineNumber)
    {
        TargetPoint = point;
        Speed = speed;
        CutterNumber = cutter;
        LineNumber = lineNumber;
    }
}
```

Метод преобразования данных в промежуточном формате Line в конечный формат MoveInfo указан в листинге 5.

Листинг 5 – метод преобразования строк в массив MoveInfo.

```
    public static List<MoveInfo> TransformLinesToInfo(List<Line>
interpretedLines)
    {
        currentPosition = Vector3.zero;
        currentSpeed = 0f;
        currentCutterNumber = 1;
        currentLineNumber = 0;

        moves = new List<MoveInfo>();

        foreach (var interpretedLine in interpretedLines)
        {
            if (interpretedLine.type == InstructionType.GType)
            {
                switch (interpretedLine.typeNum)
                {
                    case 0: //G0 - моментальное движение в указанную
точку
                        RapidMove(interpretedLine.targetCoordinate);
                        break;
                    case 1: //G1 - линейное перемещение в указанную
точку (иногда с указанной скоростью)
                        if (interpretedLine.fChanged)
                        {

LinearMove(interpretedLine.targetCoordinate, interpretedLine.F);
                            currentSpeed = interpretedLine.F;
                        }
                        else

LinearMove(interpretedLine.targetCoordinate, currentSpeed);
                        break;
                    case 2: //G2 - перемещение по дуге радиуса R/с
центром в точке (i, j, k) в указанную точку по часовой стрелке
                        if (interpretedLine.rChanged)
                        {
                            ArcMove(interpretedLine.targetCoordinate,
interpretedLine.R, RoundDirection.Clockwise, interpretedLine.F);
                        }
                        else
```

```
                        {
                            //var centerPoint = new
Vector3(interpretedLine.i, interpretedLine.j, interpretedLine.k);
                            var centerPoint = new
Vector3(interpretedLine.i, interpretedLine.k, interpretedLine.j);
//разные СКО
                            ArcMove(interpretedLine.targetCoordinate,
centerPoint, RoundDirection.Clockwise, interpretedLine.F);
                        }

                    case 3: //G3 - перемещение по дуге радиуса R / с
центром в точке (i, j, k) в указанную точку против часовой стрелки
                        if (interpretedLine.rChanged)
                        { //если указан радиус дуги
                            ArcMove(interpretedLine.targetCoordinate,
interpretedLine.R, RoundDirection.CounterClockwise,
interpretedLine.F);                         }
                        else
                        { //если указана центральная точка дуги
                            var centerPoint = new
Vector3(interpretedLine.i, interpretedLine.k, interpretedLine.j);
//разные СКО
                            ArcMove(interpretedLine.targetCoordinate,
centerPoint, RoundDirection.CounterClockwise, interpretedLine.F);
                        }
                }
            }
            currentLineNumber++;
        }
        return moves;
    }
```

После преобразования необходимо передать полученный список MoveInfo в резец CutterDeformer.cs и начать перемещение:

Листинг 6 – метод запуска потока перемещения

```
public void StartProgramMoving(List<MoveInfo> moves)
    {
        StartCoroutine(StartCutting(moves));
    }

private IEnumerator StartCutting(List<MoveInfo> moves)
    {
        var currentIndex = 0;
        var currentState = 0f;
        var delta = 0.1f;
```

```
        var yieldReturn = new WaitForSeconds(Time.deltaTime);
        Vector3 currentPosition = moves[0].TargetPoint;

        while (currentIndex < moves.Count - 1)
        {
            var length = (moves[currentIndex].TargetPoint *
cutterRadius -
                          moves[currentIndex + 1].TargetPoint *
cutterRadius).sqrMagnitude;
            if (length > 10 * cutterRadius * cutterRadius)
                delta = 0.01f;
            else delta = 0.1f;

            currentPosition =
Vector3.Lerp((moves[currentIndex].TargetPoint - minimumCoord) *
cutterRadius, (moves[currentIndex + 1].TargetPoint) * cutterRadius,
currentState);
            transform.position = currentPosition;
            EditTerrain();

            currentState += delta;
            if (currentState > 1f)
            {
                currentState = 0f;
                currentIndex++;

OnLineExecuting?.Invoke(moves[currentIndex].LineNumber);
            }

            yield return yieldReturn;
        }
    }
```

Таким образом, мы перемещаем резец по полученным координатам и вызываем каждый кадр метод EditTerrain, который редактирует заготовку, если нужно.

4.3. ВЫВОДЫ

В главе 4 была описана реализация эмулятора. Были приведены листинги методов для реализации функционала приложения. Была описана библиотека для реализации алгоритма Marching Cubes, используемого в

приложении и то, как она используется. Также был представлен интерфейс приложения с описанием всех функциональных окон.

5. ТЕСТИРОВАНИЕ

Для тестирования данного приложения необходимо проверить, насколько корректно оно отрисовывает введенную программу и то, насколько алгоритм загружает рабочую машину во время выполнения нарезания.

5.1. ТЕСТИРОВАНИЕ ВЫПОЛНЕНИЯ АЛГОРИТМА

За основу выполнения была взята простая программа с сайта примеров тестовых программ, которая позволяет проверить основные перемещения инструмента фрезерного станка [23].

Листинг 7 – тестовая управляющая программа

```
N40 G90 G00 X0 Y0
N50 G01 X-10 Y-20 R8          (P1)
N60 G01 X-50 R10              (P2)
N70 Y10                       (P3)
N80 X-19.97 Y25.01            (P4)
N90 G03 X7.97 Y38.99 R18      (P5)
N100 G01 X30 Y50              (P6)
N110 G91 X10.1 Y-10.1         (P7)
N120 G90 G02 X59.9 Y20.1 R14  (P8)
N130 G01 X70 Y10              (P9)
N140 Y-20 R10                 (P10)
N150 X50                      (P11)
N160 G03 X30 R10              (P12)
N170 G01 X10 R8               (P13)
N180 X0 Y0
```

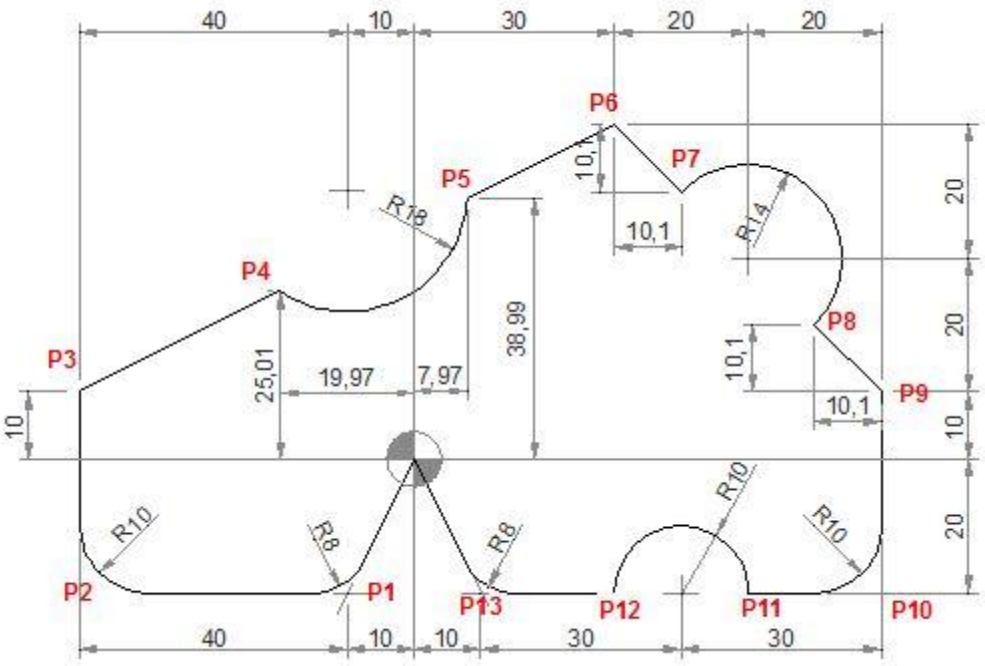

Рисунок 24 – предполагаемый результат выполнения

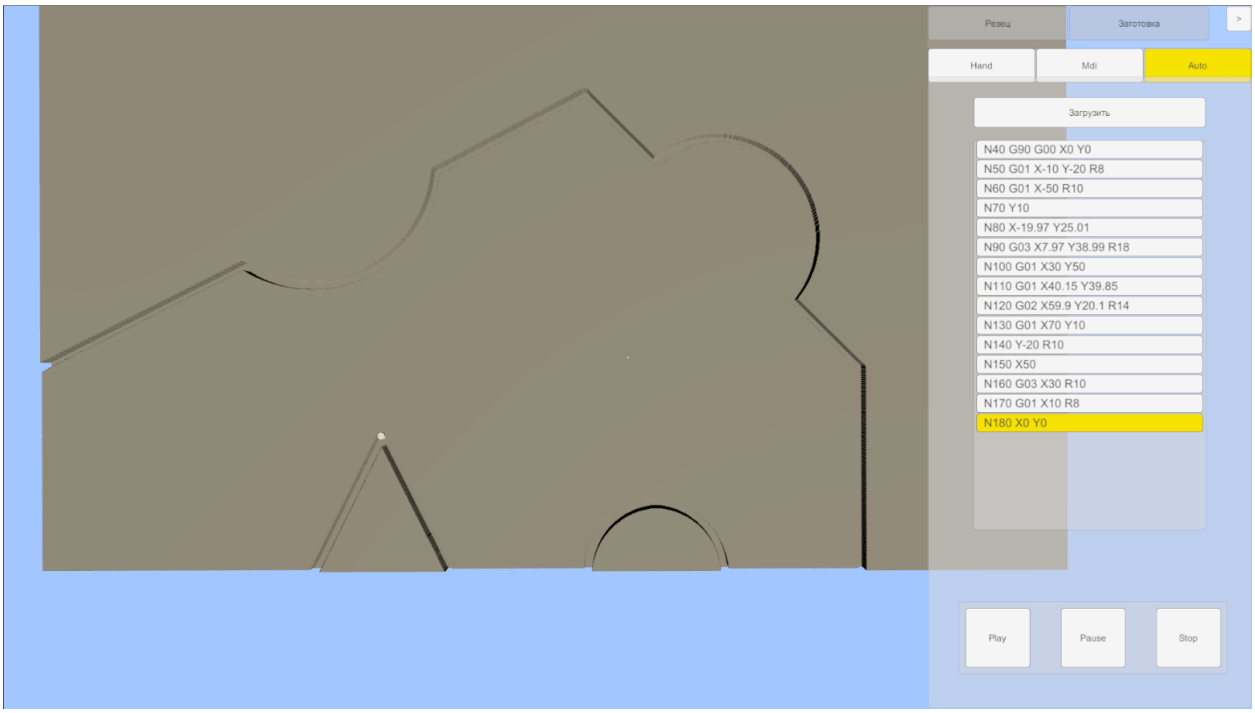

Рисунок 25 – Результат выполнения управляющей программы.

Таким образом, результат выполнения отвечает параметрам предполагаемой заготовки. Исключения составляют точки P1, P2, P10 и P13, которые получились без закруглений из-за других размеров резца, но это

будет исправлено в дальнейшем при добавлении редактирования размеров резца. В целом координаты определены правильно.

5.2. ПРОИЗВОДИТЕЛЬНОСТЬ АЛГОРИТМА

Для тестирования производительности был использован встроенный Unity Profiler, который позволяет посмотреть кадровую частоту и оценить затраты на разные аспекты работы приложения.

Данный метод измерения имеет существенный недостаток: сам профайлер нагружает систему и не позволяет точно определить нагруженность. Кроме того, для визуального оценивания был открыт редактор сцены, который тоже потребляет значительную часть ресурсов.

В качестве тестируемой программы была выбрана программа обработки на 3200 строк файлов, которая многократно проходит по одной и той же траектории, затрудняя вычисления алгоритма. Измерения были проведены в начале и конце выполнения программы

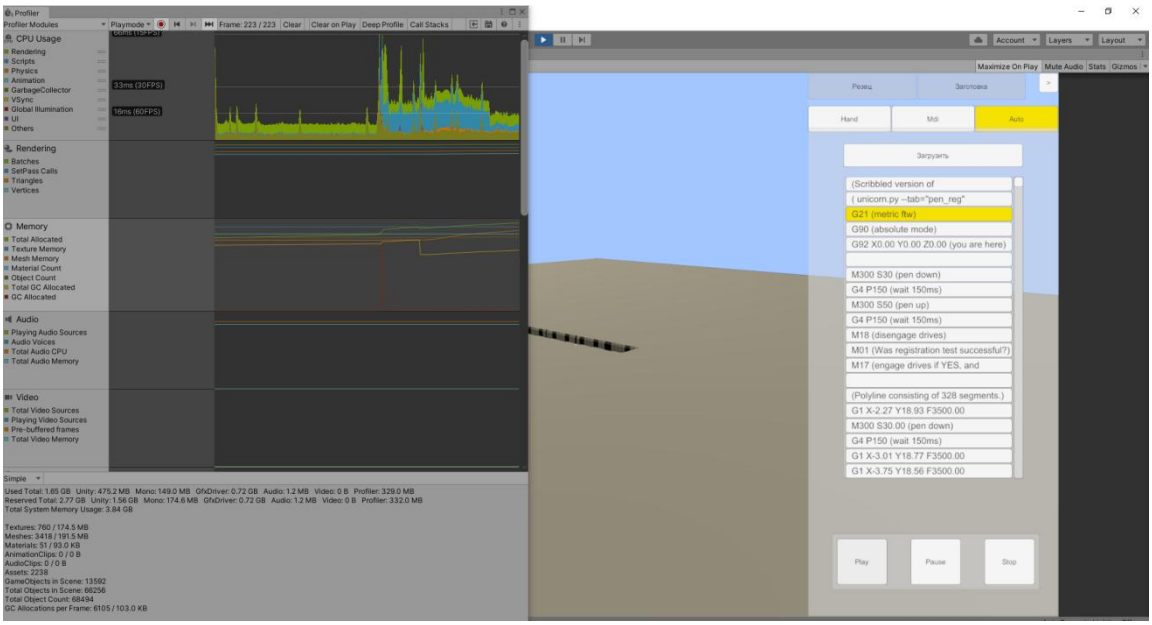

Рисунок 26 – Измерения, проведенные в начале работы выполнения программы

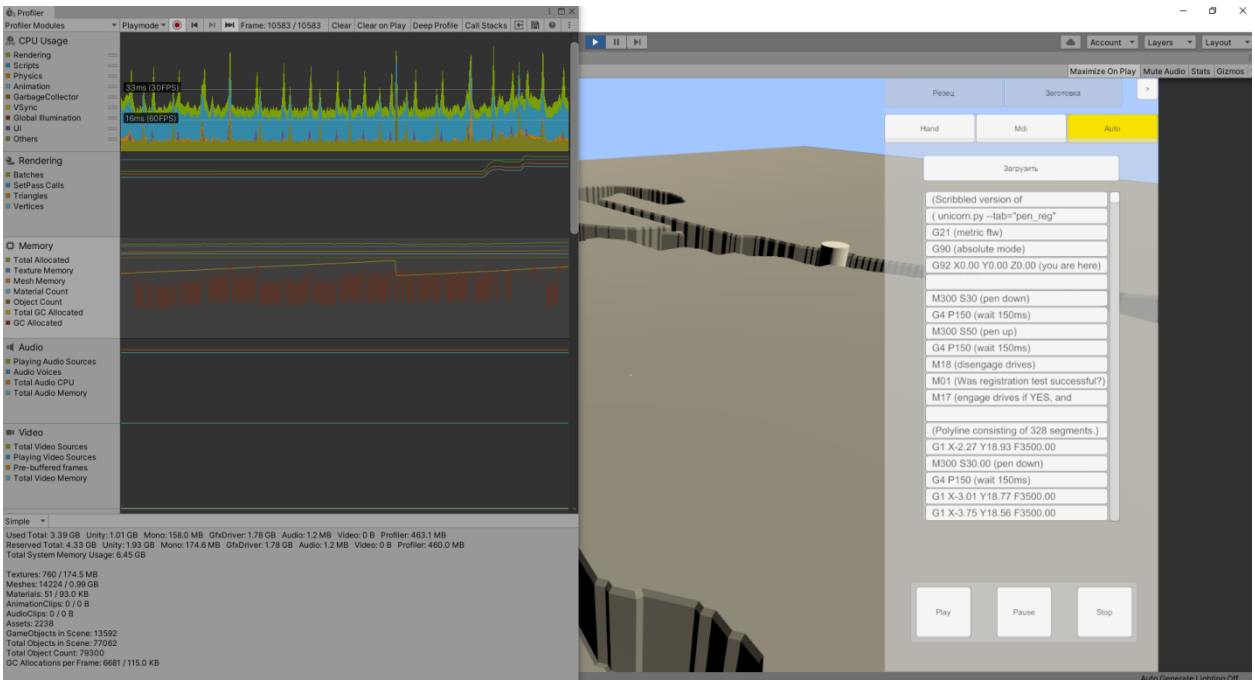

Рисунок 27 – Измерения, проведенные в конце выполнения работы программы

Как видно из показаний профайлера, количество потребляемой памяти повысилось с 1.65 ГБ до 3.39 ГБ. Тем не менее, несмотря на открытые инструменты измерения производительности, просадки кадровой частоты замечены не были. Без открытых инструментов измерения и в собранном проекте эти результаты будут лучше, однако можно сделать вывод о том, что приложение требует оптимизации в плане выделения памяти: при попытке сымитировать очень большие программы возможны утечки памяти. Возможно, необходимо вручную вызывать сборку мусора.

5.3. ВЫВОД

На основании анализа выполнения программы были сделаны выводы о точности и производительности работы приложения. Была оценена точность работы приложения. Разработаны планы по оптимизации приложения.

6. ВОПРОСЫ, ТРЕБУЮЩИЕ ДОРАБОТКИ

Несмотря на то, что приложение работает корректно и на данном этапе разработки отвечает основным требованиям пользователя, необходимо провести ряд изменений для комфортной работы пользователя:

6.1. ДОБАВЛЕНИЕ АНИМИРОВАННОЙ МОДЕЛИ СТАНКА

На данный момент уже существует модель трехосевого фрезерного станка, процесс работы на которой также необходимо визуализировать. Однако, на ней отсутствуют материалы и текстуры. К тому же, необходимо записать соответствующие анимации перемещения осей станков, учитывая корректное перемещение и скорость обработки.

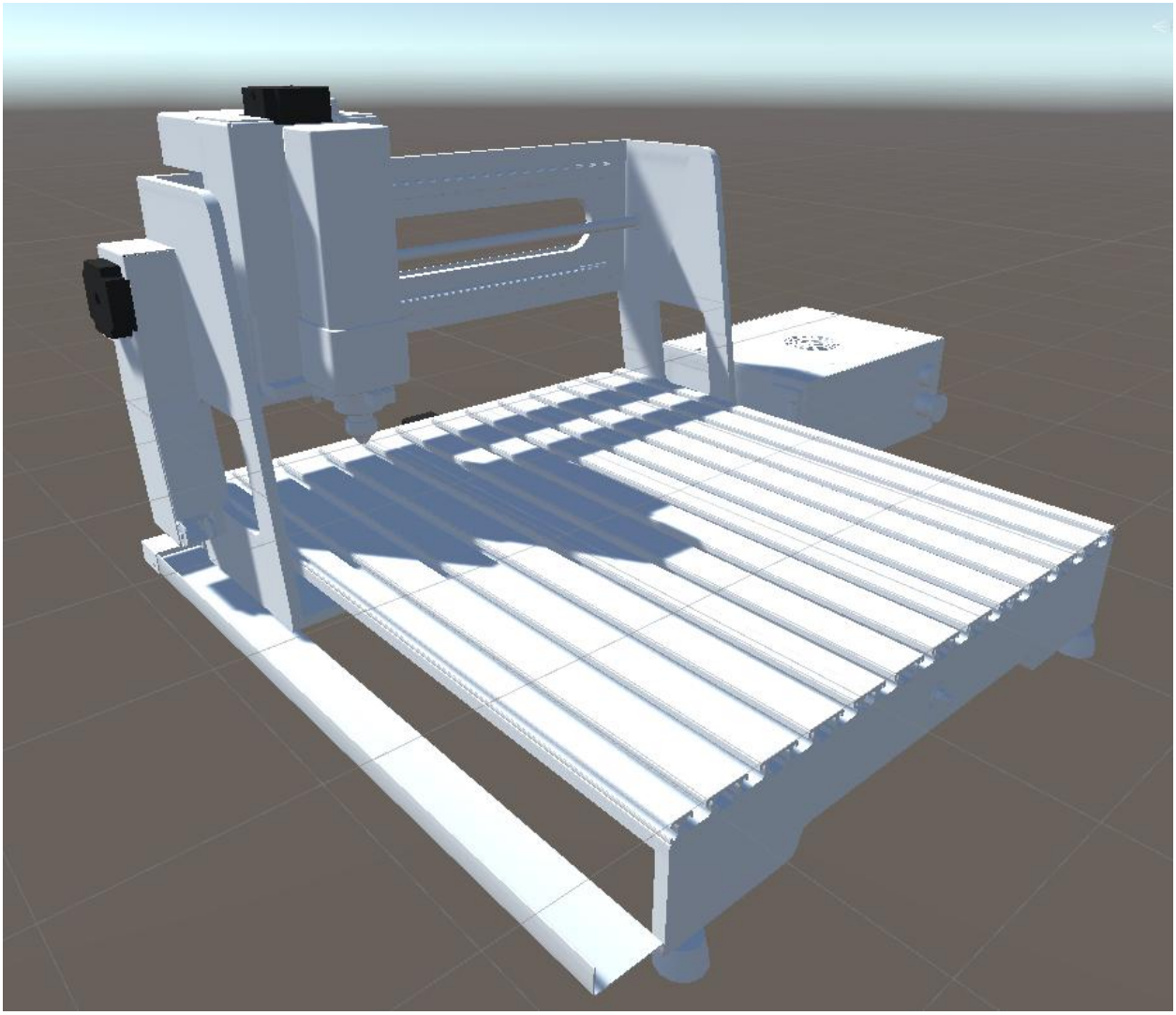

Рисунок 28 – Модель фрезерного станка «Микрон»

6.2. НАПИСАНИЕ ШЕЙДЕРОВ ДЛЯ БОЛЕЕ КОРРЕКТНОГО ОТОБРАЖЕНИЯ ОБРАБОТАННОЙ ДЕТАЛИ

Несмотря на корректную генерацию точек заготовки, материал заготовки не позволяет правильно отобразить освещение редактируемой области заготовки. Данную проблемы помогают исправить шейдеры, которые могут отражать свет по-разному, в зависимости от положения точки в пространстве.

Однако, данный проект является только прототипом будущего проекта, для которого еще не выбран тип компьютерной графики: URP или HDRP. Данные конвейеры компьютерной графики не поддерживают шейдеры друг

друга, а значит, его написание должно производиться на одном из последних этапов.

6.3. ИЗМЕНЕНИЕ ПАРАМЕТРОВ ИНСТРУМЕНТА, ДЕТАЛИ И НУЛЕВОЙ ТОЧКИ

Для того, чтобы можно было правильно работать на станке, необходимо менять:

- размер заготовки по всем 3 осям;
- размеры инструмента – радиус и высота;
- форма инструмента – конус или цилиндр;
- смещение нулевой точки детали – команды G54-G59.

Это является одной из первоочередных задач на данный момент.

6.4. ВВЕДЕНИЕ ВОЗМОЖНОСТИ НАКЛОНА СТОЛА/ИНСТРУМЕНТА (5-ОСЕВОЙ ОБРАБОТКИ)

Для введения возможности наклона инструмента, необходимо дополнить условие метода проверки точек, лежащих внутри резца, а также отредактировать парсинг файла управляющей программы в автоматическом режиме. Данное приложение поддерживает возможность такого расширения.

6.5. ВВЕДЕНИЕ ИНСТРУМЕНТОВ ИЗМЕРЕНИЯ ПАРАМЕТРОВ ОБРАБОТАННОЙ ЗАГОТОВКИ

Точность работы алгоритма на данный момент можно проверить только в режиме редактирования приложения. Для того, чтобы сделать это в запущенном приложении, необходимо создать инструмент измерения расстояния между двумя точками.

Также, можно создать несколько уроков по измерению обработанной детали с моделированием реального измерительного оборудования. Однако,

данная задача выходит за пределы выпускной квалификационной работы.

7. ЗАКЛЮЧЕНИЕ

По итогам выпускной квалификационной работы было разработано приложение для эмуляции процесса фрезерной обработки в соответствии с требованиями заказчика. Благодаря обзору литературы и аналогов были сделаны выводы по концепции будущего приложения. После чего были определены требования для будущей системы, учитывающие интересы групп пользователей. Был произведен анализ существующих алгоритмов генерации 3D-моделей и выбран оптимальный алгоритм. Приложение было реализовано, протестировано и отлажено. Также определены аспекты для дальнейшей реализации и улучшения приложения.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Юсупова, Ф. Э. Симуляторы в образовательном процессе / Ф. Э. Юсупова, М. Об. К. Солижонова // Вопросы науки и образования. 2018. №10 (22). URL: https://cyberleninka.ru/article/n/simulyatory-v-obrazovatelnom-protsesse/. Дата обращения: 01.04.2021.

2. Россия – мировой центр разработки САПР. Вы об этом знали? / Хабр - https://habr.com/ru/post/473694/. Дата обращения: 25.03.2021.

3. Емелин, А. Г. Геометрическое моделирование технологического процесса фрезерной обработки на станках с ЧПУ : специальность 05.13.12 «Системы автоматизации проектирования (по отраслям)» : автореферат диссертации на соискание ученой степени кандидата технических наук / А.Г. Емелин – Екатеринбург, 2004. – 18 с.

4. Степанов, Д. Д. Вопросы разработки обучающих симуляторов крановых операций в морском порту в среде Unity / Д.Д. Степанов // Системный анализ и логистика: журнал.: выпуск No3(25), ISSN 2007-5687. – СПб.: ГУАП. -2020 –с. 25-32. РИНЦ

5. Enhanced simulation of the milling process for a novel roughing and finishing tool / R. Grabowski // Garbsen : TEWISS Verlag – 2019. – 10.15488/9944

6. A forward closed-loop virtual simulation system for milling process considering dynamics processing-machine interactions / Chen W, Tong Z, Huo D, Zhong W and Jiang X // Springer London – 2019 – oai:eprint.ncl.ac.uk:258813

7. Kupriyanov, A. V. Basic terms and functions of CAD/CAM/CAE systems / A. V. Kupriyanov // Молодежь. Общество. Современная наука, техника и инновации. – 2019. – № 18. – с. 256-257..

8. Сазонова, А. О. Классификация и место CAM-систем в системах автоматизированного проектирования / А. О. Сазонова, А. А. Дроздов // Master's Journal. – 2014. – No 2. – P. 34-42.

9. Нулевая точка станка и направления перемещений – http://planetacam.ru/college/learn/4-1/. Дата обращения: 25.03.2021.

10. G- и M-коды – http://planetacam.ru/college/learn/5-1/. Дата обращения: 03.02.2021.

11. Как работает 5-осевой фрезерный станок с ЧПУ. Устройство станка с ЧПУ 5 осей – https://3dtool.ru/stati/kak-rabotaet-5-osevoy-frezernyy-stanok-s-chpu-ustroystvo-stanka-s-chpu-5-osey/. Дата обращения: 01.04.2021.

12. Unity - Manual: Unity User Manual 2020.3 (LTS) – https://docs.unity3d.com/Manual/index.html. Дата обращения: 01.04.2021.

13. Colin S., On Vertex-Vertex Meshes and Their Use in Geometric and Biological Modeling – http://algorithmicbotany.org/papers/smithco.dis2006.pdf. Дата обращения: 03.03.2021.

14. Процедурная генерация трёхмерных моделей / Хабр – https://habr.com/ru/post/194620/. Дата обращения: 02.02.2021.

15. Marching Squares, a Unity C# Tutorial – https://catlikecoding.com/unity/tutorials/marching-squares/. Дата обращения: 30.03.2021

16. Marching Cubes 3d Tutorial – BorisTheBrave.Com – https://www.boristhebrave.com/2018/04/15/marching-cubes-3d-tutorial/. Дата обращения: 30.03.2021.

17. Трохова, Т.А. Компьютерное моделирование динамических объектов с использованием Unity / Т.А. Трохова // Агротехника и энергообеспечение. 2018. №1 (18). URL: https://cyberleninka.ru/article/n/kompyuternoe-modelirovanie-dinamicheskih-obektov-s-ispolzovaniem-sredy-razrabotki-kompyuternyh-igr-unity Дата обращения: 12.03.2021.

18. Unity: процедурное редактирование Mesh / Блог компании Plarium / Хабр – https://habr.com/ru/company/plarium/blog/443870/. Дата обращения: 02.02.2021.

19. Галкин, Н. С. Создание трехмерной модели местности в unity 3D / Н. С. Галкин, Е. А. Ромин // Инновационные технологии: теория, инструменты, практика. – 2014. – № 1. – С. 311-316.

20. Организация интерфейса в Unity с UI Canvas / Хабр – https://habr.com/ru/post/472770/. Дата обращения: 30.03.2021.

21. Unity3D: архитектура игры, ScriptableObjects, синглтоны / Хабр – https://habr.com/ru/post/414361/. Дата обращения: 01.04.2021

22. Как писать функциональные требования / Блог компании Retail Rocket / Хабр – https://habr.com/ru/company/retailrocket/blog/431572/. Дата обращения: 30.03.2021

23. Simple G Code Example Mill - G code Programming for Beginners - Helman CNC – http://www.helmancnc.com/simple-g-code-example-mill-g-code-programming-for-beginners/. Дата обращения: 27.05.2021