

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по направлению подготовки
09.03.04 Программная инженерия

Направленность (профиль) Промышленная разработка программного обеспечения

Студента Назарова Алексея Ивановича шифр 150341/п

Институт приборостроения, автоматизации и информационных технологий

Тема выпускной квалификационной работы

«Разработка подсистемы имитации движения летательного аппарата для
программного имитатора закабинного пространства»

Студент

14.06.19  А. И. Назаров

Руководитель

15.06.19  А. И. Фролов

Нормоконтроль

14.06.19  А.Ю. Ужаринский

Зав. кафедрой
программной инженерии

20.06.19  А.И. Фролов

Орёл 2019

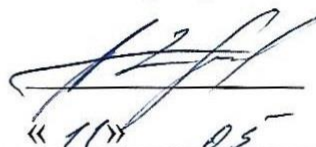
МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Институт приборостроения, автоматизации и информационных технологий
Кафедра программной инженерии
Направление 09.03.04 Программная инженерия
Направленность (профиль) Промышленная разработка программного обеспечения

УТВЕРЖДАЮ:

Зав. кафедрой

 А.И. Фролов
«10» 05 2019г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студента Назарова Алексея Ивановича

шифр 150341/п

1 Тема ВКР «Разработка подсистемы имитации движения летательного аппарата для программного имитатора закабинного пространства»

Утверждена приказом по университету от «20» ноября 2018г. № 2-3635

2 Срок сдачи студентом законченной работы «20» июня 2019г.

3 Исходные данные к работе

Теоретический материал; информация о предметной области;

4 Содержание пояснительной записки (перечень подлежащих разработке вопросов)

Анализ задачи и формулировка требований к разработке подсистемы имитации движения летательного аппарата для программного имитатора закабинного пространства;

Определение спецификаций к подсистеме имитации движения летательного аппарата;

Проектирование подсистемы имитации движения летательного аппарата;

Реализация, тестирование и отладка подсистемы имитации движения летательного аппарата.

5 Перечень демонстрационного материала

Презентация, отображающая основные этапы и результаты выполнения ВКР

Дата выдачи задания

« 11 » мая 2019 г.

Руководитель



(подпись)

А. И. Фролов

Задание принял к исполнению



(подпись)

А. И. Назаров

КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов работы	Сроки выполнения этапов работы	Примечание
Анализ задачи и формулировка требований к разработке подсистемы имитации движения летательного аппарата для программного имитатора закабинного пространства	11.05.19 – 15.05.19	
Определение спецификаций к подсистеме имитации движения летательного аппарата	16.05.19 – 21.05.19	
Проектирование подсистемы имитации движения летательного аппарата	22.05.19 – 30.05.19	
Реализация, тестирование и отладка подсистемы имитации движения летательного аппарата	31.05.19 – 9.06.19	
Оформление ВКР	10.06.19 – 20.06.19	

Студент



(подпись)

А. И. Назаров

Руководитель



(подпись)

А. И. Фролов

АННОТАЦИЯ

ВКР 131 с., 23 рис., 2 табл., 8 источников, 1 прил.

ИМИТАТОР ЗАКАБИННОГО ПРОСТРАНСТВА, МОДЕЛИРОВАНИЕ ПРОЛЕТА, КУБИЧЕСКИЙ СПЛАЙН, ЕСТЕСТВЕННЫЙ ТРЕХГРАННИК, ЛЕТАТЕЛЬНЫЙ АППАРАТ, КАСАТЕЛЬНЫЙ ВЕКТОР, ВЕКТОР ГЛАВНОЙ НОРМАЛИ, ВЕКТОР БИНОРМАЛИ, ТАНГАЖ, КРЕН, КУРС, СКОРОСТЬ ЛЕТАТЕЛЬНОГО АППАРАТА, УСКОРЕНИЕ ЛЕТАТЕЛЬНОГО АППАРАТА.

Выпускная квалификационная работа на тему «Разработка подсистемы имитации движения летательного аппарата для программного имитатора закабинного пространства».

В первой главе ВКР проводится анализ задачи разработки подсистемы имитации движения летательного аппарата для программного имитатора закабинного пространства, рассматриваются общие сведения об имитаторе закабинного пространства. Изучается предложенная математическая модель с последующими выводами о её функциональных особенностях и возможностях.

Во второй главе определяются спецификации к подсистеме имитации движения летательного аппарата. Обосновывается выбор подхода к разработке, разрабатывается функционал подсистемы. На основе рассмотренной математической модели формулируется методика расчета полетных данных.

В третьей главе приводится описание структуры и компонентов подсистемы, разрабатываются структуры данных и алгоритмы, используемые в процессе реализации. Подробно рассматривается проектирование пользовательского интерфейса с созданием макетов двух уровней точности.

В четвертой главе описывается процесс реализации математического ядра, с демонстрацией диаграммы компонентов. Дается описание графического пользовательского интерфейса, используемого для создания полетного задания и демонстрации пролета летательного аппарата по заданному маршруту. Также дается описание методов поиска и устранения возникших ошибок в подсистеме.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 АНАЛИЗ ЗАДАЧИ И ФОРМУЛИРОВКА ТРЕБОВАНИЙ К РАЗРАБОТКЕ ПОДСИСТЕМЫ ИМИТАЦИИ ДВИЖЕНИЯ ЛЕТАТЕЛЬНОГО АППАРАТА ДЛЯ ПРОГРАММНОГО ИМИТАТОРА ЗАКАБИННОГО ПРОСТРАНСТВА	9
1.1 Общие сведения об имитаторе закабинного пространства	9
1.2 Анализ задачи моделирования пролета летательного аппарата	13
1.2.1 Общая постановка задачи	13
1.2.2 Анализ математической модели	14
1.3 Формулировка функциональных и эксплуатационных требований к подсистеме имитации движения летательного аппарата	19
2 ОПРЕДЕЛЕНИЕ СПЕЦИФИКАЦИЙ К ПОДСИСТЕМЕ ИМИТАЦИИ ДВИЖЕНИЯ ЛЕТАТЕЛЬНОГО АППАРАТА	23
2.1 Выбор подхода к разработке	23
2.2 Разработка функциональных спецификаций	23
2.3 Методика расчета полетных данных	25
2.4 Разработка информационных спецификаций	27
2.5 Разработка поведенческих спецификаций	28
3 ПРОЕКТИРОВАНИЕ ПОДСИСТЕМЫ ИМИТАЦИИ ДВИЖЕНИЯ ЛЕТАТЕЛЬНОГО АППАРАТА	32
3.1 Проектирование структуры подсистемы	32
3.2 Проектирование компонентов подсистемы	35
3.3 Проектирование структур данных	52
3.4 Проектирование алгоритмов	55
3.4.1 Алгоритм вычисления коэффициентов кубического сплайна	55
3.4.2 Алгоритм определения орт естественного трехгранника	60

3.4.3 Алгоритм решения систем линейных уравнений методом Гаусса	66
3.5 Проектирование пользовательского интерфейса	69
4 РЕАЛИЗАЦИЯ, ТЕСТИРОВАНИЕ И ОТЛАДКА ПОДСИСТЕМЫ ИМИТАЦИИ ДВИЖЕНИЯ ЛЕТАТЕЛЬНОГО АППАРАТА	74
4.1 Реализация компонентов подсистемы	74
4.2 Реализация пользовательского интерфейса	75
4.3 Тестирование и отладка.	80
ЗАКЛЮЧЕНИЕ	84
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	85
ПРИЛОЖЕНИЕ А - ЛИСТИНГ КОДА	86
УДОСТОВЕРЯЮЩИЙ ЛИСТ № 150341 НА ДЕМОНСТРАЦИОННЫЙ МАТЕРИАЛ	129
ИНФОРМАЦИОННО-ПОИСКОВАЯ ХАРАКТЕРИСТИКА ДОКУМЕНТА НА ЭЛЕКТРОННОМ НОСИТЕЛЕ	130

ВВЕДЕНИЕ

Плохие условия видимости являются одной из основных причин авиационных происшествий и катастроф при выполнении маловысотного полёта и взлётно-посадочных операций.

Повышение безопасности при пилотировании на малых высотах, заходе на посадку, выполнении посадки на малооборудованные аэродромы, на необорудованные посадочные площадки в условиях ограниченного пространства, в сложных метеоусловиях и в условиях ограниченной видимости, низкой освещённости может обеспечиваться за счет применения современного бортового оборудования.

Информационно-измерительные комплексы современных летательных аппаратов могут включать в себя датчики, принцип работы которых основаны на различных принципах (камеры видимого диапазона, лазерные локаторы, радиолокационные станции, камеры инфракрасных диапазонов). В настоящее время в целях повышения информативности получаемых пилотом данных перспективным считается использование одновременно нескольких датчиков, причем не только как самостоятельных и альтернативных, а с применением техники комплексирования получаемых изображений, обеспечивающей получение новой информации и повышение эффективности поиска препятствий.

Разработка таких информационно-измерительных комплексов осложняется невозможностью одновременного получения реальных данных ото всех датчиков до их изготовления, установки на летательном аппарате и проведения летных испытаний.

Для решения обозначенной проблемы применяют специальные программные средства, позволяющие имитировать закабинное пространство и получать модельные изображения и сигналы разноспектральных датчиков для последующего использования в процессах отработки алгоритмов обработки, комплексирования и анализа информации.

Имитатор закабинного пространства представляет собой программную систему, включающую: имитатор местности, имитатор пролета и блоки генерации изображений и сигналов разноспектральных датчиков.

Имитатор местности с объектами, расположенными на ней, реализует 3D-модель окружающей обстановки в области моделирования, обеспечивает:

- конструирование в интерактивном режиме подстилающей поверхности заданного типа(ов);
- расположение на подстилающей поверхности и задание параметров различных объектов;
- определение параметров окружающей среды (освещенности и погодных условий, включая температуру, туман, осадки);
- генерацию трехмерной модели окружающей местности.

Данные от имитатора местности поступают на вход имитатора пролета. Модель местности в имитаторе пролета используется для задания на ней ключевых точек пролета. По заданным ключевым точкам генерируется детальная модель пролета (набор точек с характеристиками положения летательного аппарата с периодом до 40 мс).

На основании модели местности и детальной модели пролета в каждой точке пролета в зависимости от заданных в настройках положений датчиков генерируются изображения и выходные сигналы датчиков.

Программный имитатор закабинного пространства в текущей версии позволяет моделировать 5 датчиков модуля переднего обзора (камера видимого диапазона, инфракрасные камеры SWIR и LWIR диапазонов, лазерный локатор, радиолокатор переднего обзора) и 3 датчика модуля нижнего обзора (камера видимого диапазона, инфракрасная камера SWIR диапазона, радиолокационная станция зондирования подстилающей поверхности).

Целью данной выпускной квалификационной работы является разработка подсистемы имитации движения летательного аппарата для программного имитатора закабинного пространства.

Для достижения поставленной цели необходимо решить следующие задачи:

1) Проанализировать задачи и формулировки требований к разработке подсистемы имитации движения летательного аппарата для программного имитатора закабинного пространства;

2) Определить спецификации к подсистеме имитации движения летательного аппарата;

3) Спроектировать подсистему имитации движения летательного аппарата;

4) Реализовать, протестировать и отладить подсистему имитации движения летательного аппарата.

1 АНАЛИЗ ЗАДАЧИ И ФОРМУЛИРОВКА ТРЕБОВАНИЙ К РАЗРАБОТКЕ ПОДСИСТЕМЫ ИМИТАЦИИ ДВИЖЕНИЯ ЛЕТАТЕЛЬНОГО АППАРАТА ДЛЯ ПРОГРАММНОГО ИМИТАТОРА ЗАКАБИННОГО ПРОСТРАНСТВА

1.1 Общие сведения об имитаторе закабинного пространства

В рамках научно-исследовательской, опытно-конструкторской и технологической работ необходима разработка и создание всепогодного и всесезонного комплекса для обеспечения поисково-спасательных операций, проводимых с помощью летательных аппаратов в условиях Арктики. Комплекс для обеспечения поисково-спасательных операций (КОПСО) представляет собой комплекс информационно-измерительных средств, обеспечивающих повышение безопасности полётов вертолётов и информационного обеспечения поисково-спасательных операций, таких как:

- поиск и обнаружение потерпевших бедствие в прибрежных морских районах и на побережье;
- наведение наземных поисково-спасательных сил на объекты поиска;
- десантирование спасательных групп посадочным способом в сложных метеорологических условиях.

КОПСО включает в себя следующие составные части: автономный источник питания (электрический рекуператор), лазерно-телевизионный модуль, радиолокационную станцию переднего обзора, радиолокационную станцию зондирования подстилающей поверхности и аппаратуру управления и комплексной (АУК) обработки информации.

Для обеспечения проведения комплексной отладки, предварительных и приемочных испытаний опытных образцов КОПСО, а также для проведения приемосдаточных испытаний серийных образцов используется технологический стенд комплексной настройки и проверки (ТСКН).

Процессы комплексной отладки, предварительных и приемочных испытаний опытных образцов КОПСО, приемосдаточных испытаний серийных образцов с использованием ТСКН, сопряжены с необходимостью проведения

большого количества тестовых запусков в части отладки и проверки корректности функционирования программного обеспечения аппаратуры управления и комплексной обработки информации. В условиях отсутствия (до проведения большого количества летных испытаний) достаточного количества исходных данных, получаемых с измерительных блоков КОПСО, необходимо обеспечить генерацию подобных изображений в режиме имитации пролета в определенной местности с заданными параметрами.

С целью обеспечения такой возможности в состав ТСКН входит имитатор закабинного пространства (ИЗП), предназначенный для моделирования измерительной информации (генерации файлов различного формата) от разноспектральных датчиков, входящих в состав аппаратуры КОПСО, при пролете летательного аппарата (ЛА) над участком местности с заданными характеристиками при заданных условиях.

На основе анализа требований технического задания на составную часть научно-исследовательской и опытно-конструкторской работы (НИОКТР) предлагается схема программной системы ИЗП, приведённая на рисунке 1.1.1.

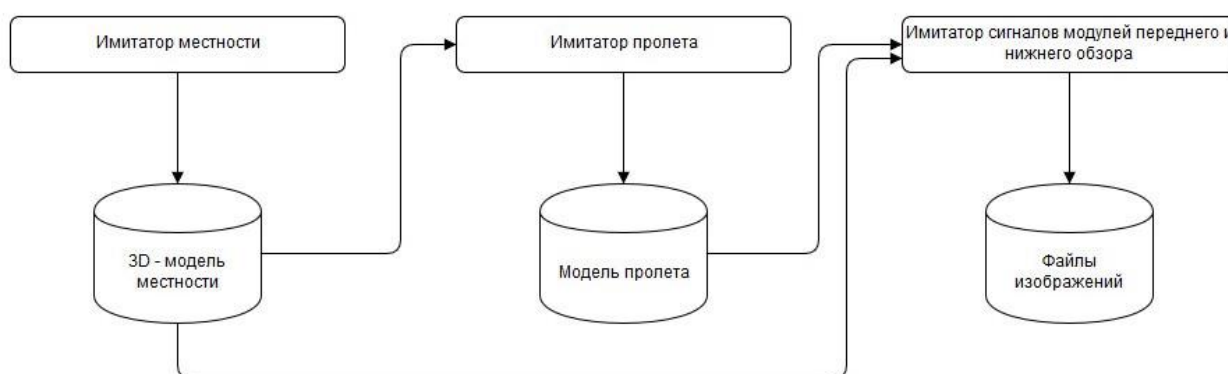


Рисунок 1.1.1 – Схема взаимодействия программ программной системы ИЗП

Имитатор местности с объектами, расположенными не ней, реализующий 3D-модель окружающей обстановки в области наблюдения, должен обеспечить:

- конструирование в интерактивном режиме подстилающей поверхности заданного типа(ов);
- расположение на подстилающей поверхности и задание параметров различных объектов;

- определение параметров окружающей среды (освещенности и погодных условий, включая температуру, туман, осадки);
- генерацию трехмерной модели окружающей местности.

Исходя из этого, имитатор должен включать в себя редактор и базу данных (БД) окружающей местности.

Редактор представляет собой компьютерную программу, в которой пользователю предоставляется возможность добавлять, удалять и редактировать подстилающие поверхности заданных типов и объекты с определёнными характеристиками.

Объект может представлять собой примитив (куб, сфера и т.п.) либо композицию примитивов в случае моделирования сложного объекта, например опоры ЛЭП. В специфике поставленной задачи, высокий уровень детализации не нужен, поэтому достаточно оперировать заданными топологиями объектов.

Основные характеристики примитива объекта:

- форма;
- размер(ы);
- текстуры поверхности (граней).

Наложение текстур необходимо для дальнейшей генерации изображений (извлечение необходимых моделируемых физических параметров) в различных спектрах. Цвет текстуры (градиент интенсивности) для инфракрасного диапазона задаёт температуру определённой области объекта. Также текстура может иметь альфа-канал (степень прозрачности). Данная характеристика может быть полезна для физических расчётов, связанных с освещённостью.

Объекты находятся на определённых типах подстилающих поверхностей:

- поле;
- лес;
- кустарник;
- болото;
- скальный грунт;
- водная поверхность (характеризуется и волнением и глубиной).

Выделяется два вида покрытия подстилающей поверхности:

- ледяная поверхность (характеризуется толщиной);
- снежный покров (характеризуется толщиной).

Подстилающая поверхность тоже представляет собой примитив типа «поверхность» с определенной геометрией (топологией) и с заданной «текстурой».

Общая структура данных для хранения геометрии 3D-модели окружающего пространства может быть представлена в виде воксельной модели, т. к. по сравнению с полигональной, в специфике уровня детализации, является более эффективной по времени обращения к элементу (полю) структуры, по затрачиваемому количеству памяти (видеопамяти).

Также пользователь в редакторе может задавать температурные и погодные характеристики окружающего пространства.

Элементы 3D-модели окружающего пространства, с которыми оперирует пользователь, добавляются в БД, т. к. это позволит более гибко оперировать объектами, хранить данные в удобном и защищённом месте, гибко масштабировать разрабатываемую систему в будущем. Планируется выбрать не классическую, реляционную, технологию построения БД, а объектно-ориентированную, т. к. требуется хранить большое количество разноплановой информации (например, численные характеристики, саму 3D-модель (геометрию), текстуру и т.д.)

Для разработки редактора планируется использовать готовые решения с открытым исходным кодом с последующей модификацией под требования к подсистеме имитации местности ИЗП.

Данные от имитатора местности (модель, элементы БД) поступают на вход имитатора полета. Модель местности в имитаторе полета используется для задания на ней полетного задания. На основании полетного задания и программной физической модели движения ЛА имитатор полета генерирует модель полета, содержащую полетную информацию в соответствии с требованиями технического задания:

- положение в пространстве (3 координаты);
- проекции вектора скорости на оси системы координат;
- ориентация (3 угла: курс, крен, тангаж).

Подсистемы имитации сигналов модулей переднего и нижнего обзора в качестве входной информации получают модель пролета и модель окружающей местности. На основании информации о характеристиках датчиков, их выходных изображениях, точках подвеса датчиков и модели пролета строятся двухмерные представления видимой области модели окружающей местности с учетом условий видимости, и параметров движения генерируются изображения в различных спектрах и сохраняются во внешнюю память.

Полученные серии файлов будут использоваться для настройки, проверки и отладки аппаратуры управления и комплексной обработки информации, а именно алгоритмов и программного обеспечения обработки данных датчиков. В ходе выполнения работ по настройке, проверке и отладке АУК полученные посредством ИЗП файлы будут последовательно подаваться на вычислительное устройство ТСКН АУК для обработки, а также на инструментальный персональный компьютер ТСКН АУК для оценки количества и качества входной информации.

1.2 Анализ задачи моделирования пролета летательного аппарата

1.2.1 Общая постановка задачи

Задача моделирования пролета летательного аппарата предполагает решение двух подзадач:

1) Инициализация параметров пролёта ЛА, отвечающего за первичную инициализацию параметров пролёта ЛА, инициализацию ПО, которое производит компьютерное моделирование пролёта. Задание параметров пролета осуществляется путем расстановки на трехмерном представлении местности ключевых точек траектории пролета с возможностью задания в ключевых точках макроопераций (например, кружение, разворот по заданному радиусу и т.д.).

2) Моделированием процесса пролёта ЛА, отвечающего за имитацию физических процессов пролёта ЛА, основываясь на данных задания на пролет и 3D-модели окружающего пространства. По заданным в интерактивном режиме ключевым точкам с использованием интерполяции кубическими сплайнами строится детальная траектория пролета с расчетом всех определенных в техническом задании параметров пролета.

1.2.2 Анализ математической модели

Математическая модель пролета летательного аппарата определяется в пояснительной записке программного имитатора закабинного пространства.

На основании данного документа летательный аппарат рассматривается как абсолютно твердое тело, следовательно, ограничимся кинематикой пролета, т.е. будем имитировать движение центра масс C этого летательного аппарата.

Пусть в пространстве с выбранной земной системой координат $O_0X_0Y_0Z_0$ заданы n точек $P_1(x_1, y_1, z_1), \dots, P_n(x_n, y_n, z_n)$, через которые должна пройти траектория центра масс C , рассматривая как годограф радиус-вектора $r(t) = (r_1(t), r_2(t), r_3(t))$, где независимый аргумент t интерпретируется как время. При этом скорость $v(t) = r'(t)$, а ускорение $a(t) = r''(t)$. Так как движение любого реального объекта характеризуется не только непрерывной траекторией, но и непрерывным ускорением, то интерполировать скалярные функции $r_i(t)$ необходимо функциями, имеющими непрерывную производную второго порядка. Этому условию, в частности, отвечают кубические сплайны. Считая известным время «прибытия» t_j центра масс C в точку $P_j(x_j, y_j, z_j)$ (выбор значений $t_1 < t_2 < \dots < t_n$ будет описан далее), получим, что

$$r_1(t) = \begin{cases} p_1(t) \equiv a_{1,0} + a_{1,1}t + a_{1,2}t^2 + a_{1,3}t^3, & t \in [t_1, t_2], \\ p_2(t) \equiv a_{2,0} + a_{2,1}t + a_{2,2}t^2 + a_{2,3}t^3, & t \in [t_2, t_3], \\ \vdots \\ p_{n-1}(t) \equiv a_{n-1,0} + a_{n-1,1}t + a_{n-1,2}t^2 + a_{n-1,3}t^3, & t \in [t_{n-1}, t_n], \end{cases} \quad (1)$$

$$r_2(t) = \begin{cases} p_1(t) \equiv b_{1,0} + b_{1,1}t + b_{1,2}t^2 + b_{1,3}t^3, & t \in [t_1, t_2], \\ p_2(t) \equiv b_{2,0} + b_{2,1}t + b_{2,2}t^2 + b_{2,3}t^3, & t \in [t_2, t_3], \\ \vdots \\ p_{n-1}(t) \equiv b_{n-1,0} + b_{n-1,1}t + b_{n-1,2}t^2 + b_{n-1,3}t^3, & t \in [t_{n-1}, t_n], \end{cases} \quad (2)$$

$$r_3(t) = \begin{cases} p_1(t) \equiv c_{1,0} + c_{1,1}t + c_{1,2}t^2 + c_{1,3}t^3, & t \in [t_1, t_2], \\ p_2(t) \equiv c_{2,0} + c_{2,1}t + c_{2,2}t^2 + c_{2,3}t^3, & t \in [t_2, t_3], \\ \vdots \\ p_{n-1}(t) \equiv c_{n-1,0} + c_{n-1,1}t + c_{n-1,2}t^2 + c_{n-1,3}t^3, & t \in [t_{n-1}, t_n], \end{cases} \quad (3)$$

где $4n - 4$ коэффициентов $a_{j,0}, a_{j,1}, a_{j,2}, a_{j,3}$ ($j = 1, 2, \dots, n$) находятся из известных условий кубической сплайн интерполяции:

$$p_k(t_k) = x_k, k = 1, 2, \dots, n - 1 \quad (4)$$

$$p_k(t_{k+1}) = p_{k+1}(t_{k+1}), k = 1, 2, \dots, n - 1 \quad (5)$$

$$p'_k(t_{k+1}) = p'_{k+1}(t_{k+1}), k = 1, 2, \dots, n - 2 \quad (6)$$

$$p''_k(t_{k+1}) = p''_{k+1}(t_{k+1}), k = 1, 2, \dots, n - 2 \quad (7)$$

$$p''_1(t_1) = p''_{n-1}(t_n) = 0 \quad (8)$$

Аналогично для коэффициентов $b_{j,0}, b_{j,1}, b_{j,2}, b_{j,3}$ и $c_{j,0}, c_{j,1}, c_{j,2}, c_{j,3}$. Само вычисление искоемых коэффициентов $a_{j,0}, a_{j,1}, a_{j,2}, a_{j,3}$ (как и коэффициентов $b_{j,0}, b_{j,1}, b_{j,2}, b_{j,3}$ и $c_{j,0}, c_{j,1}, c_{j,2}, c_{j,3}$) проводится стандартными алгоритмами.

Значения t_1, t_2, \dots, t_n выбираются в два этапа. При получении списка точек $P_1(x_1, y_1, z_1), \dots, P_n(x_n, y_n, z_n)$ сходу можно утверждать только то, что $t_1 < t_2 < \dots < t_n$.

На первом этапе по заданным n точкам P_j вычисляются $n - 1$ расстояний между соседними точками:

$$\Delta P_j = |P_{j+1} - P_j| = \sqrt{(x_{j+1} - x_j)^2 + (y_{j+1} - y_j)^2 + (z_{j+1} - z_j)^2},$$

$$j = 1, \dots, n - 1 \quad (9)$$

Округляя полученные результаты до целых значений, не меньших их, получаем (с точностью до постоянного множителя с размерностью $[t]/[L]$) временные промежутки для j -го участка траектории:

$$\Delta t_j = \Delta P_j, j = 1, \dots, n - 1 \quad (10)$$

и сами значения t_j :

$$t_{j+1} = t_j + \Delta t_j, j = 1, \dots, n - 1 \quad (11)$$

Далее вычисляем коэффициенты сплайна и считаем 1-й этап завершенным.

На 2-м этапе для каждого -го участка траектории вычисляем скорость:

$$v_j(t) = |v_j(t)| = |r'_j(t)|, j = 1, \dots, n - 1 \quad (12)$$

и ее максимальное (на этом участке) значение:

$$V_j = \max_{[t_j, t_{j+1}]} v_j(t), j = 1, \dots, n - 1 \quad (13)$$

Затем из полученных значений выбираем наибольшее:

$$v_{\max} = \max\{V_1, V_2, \dots, V_{n-1}\} \quad (14)$$

Вычисляем «корректирующий» множитель:

$$M = \frac{v_{\max}}{V_{\max}} \quad (15)$$

(где $V_{\max} = 250 \text{ км/час} \approx 0.0694 \text{ км/с}$ – максимальная допустимая скорость летательного аппарата) и новые значения t_j по формуле

$$t_{j+1} = t_j + M \cdot \Delta t_j, j = 1, \dots, n - 1 \quad (16)$$

После этого повторно вычисляем коэффициенты сплайна и завершаем 2-й этап.

Обозначим через L траекторию центра масс C (т.е. пространственная кривая L – годограф найденной выше векторной функции $r(t)$). Из дифференциальной геометрии известно, что каждой точке C кривой L соответствуют три вектора – $\tau(t), n(t), b(t)$ (исходящих из C и образующих сопровождающий трехгранник кривой L), где

$\tau(t)$ – орт касательного вектора $T = r'(t)$,

$n(t)$ – орт вектора главной нормали $N = (r'(t) \times r''(t)) \times r'(t)$,

$b(t)$ – орт вектора бинормали $B = r'(t) \times r''(t)$,

где \times - знак векторного произведения.

Сопровождающий трехгранник кривой L можно связать с твердым телом (летательный аппарат),двигающимся вдоль траектории L . Для этого плоскость симметрии летательного аппарата совмещаем с спрямляющей плоскостью $Стb$, а

поперечную ось летательного аппарата с линией пересечения нормальной (Cbn) и соприкасающейся (Ctn) плоскостей, т.е. направляем параллельно вектору главной нормали n .

В качестве связанной системы координат $OXYZ$ возьмем $Otnb$, направив продольную ось OX по направлению касательного вектора ($OX \uparrow \tau$), нормальную ось OY вдоль вектора бинормали ($OY \uparrow b$), а поперечную ось OZ вдоль вектора главной нормали ($OZ \uparrow n$). При этом в нормальной системе координат $OX_gY_gZ_g$ вертикальная ось $OY_g \uparrow k$, а горизонтальная плоскость $OX_gZ_g \parallel O_0ij$ [пусть $OX_g \uparrow i$, тогда $OZ_g \uparrow j$].

Угол курса ψ , по определению равный углу между осью OX_g нормальной системы координат и проекцией продольной оси OX на горизонтальную плоскость OX_gZ_g нормальной системы координат, можно найти как угол между вектором i и проекцией τ_{Oxy} вектора τ на плоскость $OX_gZ_g \parallel Oij$. Если в земной системе координат вектор $\tau = [\tau_1, \tau_2, \tau_3]^T$, то вектор $\tau_{Oxy} = [\tau_1, \tau_2, 0]^T$, поэтому угол курса ψ однозначно определяется из системы

$$\begin{cases} \cos(\psi) = \frac{\tau_1}{\sqrt{\tau_1^2 + \tau_2^2}} = \frac{\tau_1}{\sqrt{1 - \tau_3^2}} \\ \sin(\psi) = \frac{\tau_2}{\sqrt{\tau_1^2 + \tau_2^2}} = \frac{\tau_2}{\sqrt{1 - \tau_3^2}} \end{cases} \quad (17)$$

Сам угол курса ψ может быть вычислен с помощью функции

$$\text{atan2}(y, x) = \begin{cases} \text{arctg}\left(\frac{y}{x}\right), x > 0 \\ \text{arctg}\left(\frac{y}{x}\right) + \pi, x < 0 \text{ и } y \geq 0 \\ \text{arctg}\left(\frac{y}{x}\right) - \pi, x < 0 \text{ и } y < 0 \\ \text{sign}(y) \cdot \frac{\pi}{2}, x = 0 \text{ и } y \neq 0 \\ \text{неопределено}, x = 0 \text{ и } y = 0 \end{cases} \quad (18)$$

$$\psi = \text{atan2}(\tau_2, \tau_1) \quad (19)$$

Угол тангажа θ по определению, равен углу между продольной осью OX и горизонтальной плоскостью OX_gZ_g нормальной системы координат. Так как $OX \uparrow \tau$, а $OX_gZ_g \parallel Oij$, то

$$\vartheta = (\tau, Cij) = (\tau, \tau_{Cxy}) = \frac{\pi}{2} - (\tau, k) \quad (20)$$

Следовательно:

$$\sin \vartheta = \cos(\tau, k) = \tau \cdot k = \tau_3 \quad (21)$$

$$\vartheta = \arcsin(\tau_3) \quad (22)$$

Угол крена γ , по определению равен углу между поперечной осью OZ и осью OZ_g нормальной системы координат, смещенной в положение, при котором угол курса равен нулю. В то же время угол крена γ равен углу между нормальной осью Oy_1 и вертикальной плоскостью, проходящей через ось Ox_1 .

В результате рассмотрения сил, действующих на летательный аппарат во время «правильного виража» (т.е. разворота по окружности в горизонтальной плоскости с постоянной скоростью полета V и углом крена γ без скольжения) выводиться формула

$$\operatorname{tg} \gamma(t) = \frac{v^2(t)}{g \cdot R(t)}, \quad (23)$$

где $R(t)$ – радиус кривизны траектории летательного аппарата (в момент времени t);

$v(t) = r'(t)$ – скорость летательного аппарата.

Для плоской кривой L , являющейся годографом векторной функции $r(t) = (x(t), y(t), z(t))$, кривизна $k(t)$ выражается формулой

$$k(t) = \frac{x'y'' - x''y'}{(x'^2 + y'^2)^{\frac{3}{2}}}, \quad (24)$$

которую можно переписать в виде

$$k(t) = \frac{x'y'' - x''y'}{|v(t)|^3} \quad (25)$$

При этом числитель этой дроби совпадает с третьей координатой векторного произведения $r'(t) \times r''(t)$, поэтому его можно получить с помощью смешанного произведения:

$$x'y'' - x''y' = r'(t)r''(t)k \quad (26)$$

Учитывая также, что $v^2(t) = |v(t)|^2$, а $k(t) = \frac{1}{R(t)}$, мы можем для траектории L векторной функции $r(t) = (x(t), y(t), h)$ в горизонтальной плоскости переписать в виде

$$\operatorname{tg} \gamma(t) = \frac{r'(t)r''(t)k}{g \cdot |r'(t)|} \quad (27)$$

При переходе к пространственным траекториям необходимо заменить скорость $v(t)$ на ее горизонтальную составляющую - $v(t) \cdot \cos \vartheta(t)$. Тогда получим формулу

$$\operatorname{tg} \gamma(t) = -\frac{r'(t)r''(t)k}{g \cdot |r'(t)|} \cos^2 \vartheta(t), \quad (28)$$

как частный случай (при $\vartheta(t) \equiv 0$). Таким образом, угол крена рассчитывается по формуле:

$$\gamma(t) = -\operatorname{arctg} \left(\frac{r'(t)r''(t)k}{g \cdot |r'(t)|} (1 - \tau_3^2) \right). \quad (29)$$

На основании данной математической модели имеются возможности:

- расчета маршрута движения летательного аппарата, построенного на основе некоторой последовательности опорных точек;
- расчета проекций векторов скорости на генеральные оси координат;
- расчета проекций векторов ускорения на генеральные оси координат;
- расчета характеристик поворота летательного аппарата (курс, крен, тангаж).

1.3 Формулировка функциональных и эксплуатационных требований к подсистеме имитации движения летательного аппарата

Полётное задание состоит из набора входных данных (пунктов), определяемых пользователем. Порядковый номер пункта определяет последовательность выполнения (пункт с меньшим номером выполняется раньше). В каждом пункте полётного задания задаётся набор данных из строк таблицы 1.3.1.

Описание ряда пилотажных действий (типа равномерный прямолинейный полёт, полёт с разворотом в горизонтальной плоскости, полёт с набором высоты,

полёт со снижением и т. п.) не требует дополнительных параметров. Пилотажные действия «зависание с разворотом вокруг своей оси», «полёт по кругу перед посадкой» могут быть описаны макрооперациями.

Таблица 1.3.1 – Класс «Полетное задание», входные данные

№	Элемент данных	Источник данных	Единица измерения (формат представления)	Характеристики (описания)
1	Координаты на поверхности	Ручной ввод	(X, Z) в метрах	Проекция положения ЛА на поверхность земли
2	Высота	Ручной ввод	метр	От подстилающей поверхности

Результатом имитации пролёта являются наборы данных, структура которых представлена в таблице 1.3.2. Интервал генерации полетной информации при имитации — 20 мс.

Таблица 1.3.2 – Класс «Полетная информация», выходные данные

№	Элемент данных	Единица измерения (формат представления)
1	Положение	Координаты (X,Y,Z), метр
2	Проекции скоростей на генеральные оси координат (v_x , v_y , v_z)	м/с
3	Курс (ψ)	градус
4	Тангаж (ϑ)	градус
5	Крен (γ)	градус

В техническом задании на составную часть НИОКТР по теме «Разработка имитатора закабинного пространства для применения в составе технологического стенда комплексной настройки и проверки комплекса для обеспечения поисково-спасательных операций, проводимых с помощью летательных аппаратов в условиях Арктики» в качестве требований к составу и параметрам технических средств указана минимальная конфигурация IBM-совместимого персонального компьютера, а именно:

- процессор с частотой от 2 ГГц;
- ОЗУ от 4 ГБ, ПЗУ от 50 ГБ;
- видеоадаптер NVIDIA или AMD от 2 ГБ ОЗУ.

Для определения рекомендуемой конфигурации проведен предварительный анализ вычислительной и емкостной сложности процессов обработки данных в ИЗП. Проанализированы рекомендуемые аппаратные и программные конфигурации систем 3D-моделирования (Autodesk 3ds Max 2017, Punch Home & Landscape Design, Realtime Landscaping Architect) и систем имитации полета (FlightGear, FLYIT).

Выбор рекомендуемой конфигурации по результатам анализа обосновывается следующими факторами:

- представление трехмерной модели окружающей местности размером не менее 10×10 км с высокой и средней степенью детализации с применением воксельного (Voxel) моделирования;
- необходимость многократного повторения процессов имитации пролета и генерации изображений, проведения серии экспериментов с различными параметрами в ограниченное время;
- наличие развитых и доступных средств разработки для обработки данных на графическом адаптере, наличие программного интерфейса (API) для реализации параллельных процессов обработки графической информации.

Рекомендуемые требования к составу и характеристикам технических и программных средств для функционирования ИЗП:

- процессор: характеристики не ниже Intel core i5;

- оперативная память: 16 Гб;
- жесткий диск для хранения файлов быстрого доступа: SSD 320 Гб;
- основной жесткий диск: 1 Тб;
- видеоадаптер: NVIDIA GeForce GTX 1060;
- операционная система: Linux.

2 ОПРЕДЕЛЕНИЕ СПЕЦИФИКАЦИЙ К ПОДСИСТЕМЕ ИМИТАЦИИ ДВИЖЕНИЯ ЛЕТАТЕЛЬНОГО АППАРАТА

2.1 Выбор подхода к разработке

На основании проведенного анализа установлено, что в качестве среды разработки программного обеспечения ИЗП используется межплатформенная среда разработки Unity. Языки разработки в указанной среде – C# и JavaScript (модификация).

Выбирая между этими двумя языками, следует остановиться на C#, т.к. модификация JavaScript весьма существенна, она отличается от «классического» JavaScript, что может привести к сложностям в процессе реализации, C# же остался без изменений (за исключением некоторых особенностей, связанных с его целевой операционной системой), он является весьма развитым за счет того, что базируется на .Net Framework.

C# является объектно-ориентированным языком, следовательно имеет смысл придерживаться объектно-ориентированного подхода к разработке, что является достаточно удачным решением, т.к. в разрабатываемой подсистеме явно выделяются многие объекты и взаимосвязи между ними, например:

- летательный аппарат;
- сопровождающий трехгранник;
- кубический сплайн;
- элемент полетного задания;
- элемент полетной информации.

2.2 Разработка функциональных спецификаций

Для описания функциональных спецификаций подсистемы имитации движения летательного аппарата воспользуемся диаграммой вариантов использования (use case diagram) языка UML, разработанная диаграмма представлена на рисунке 2.2.1.

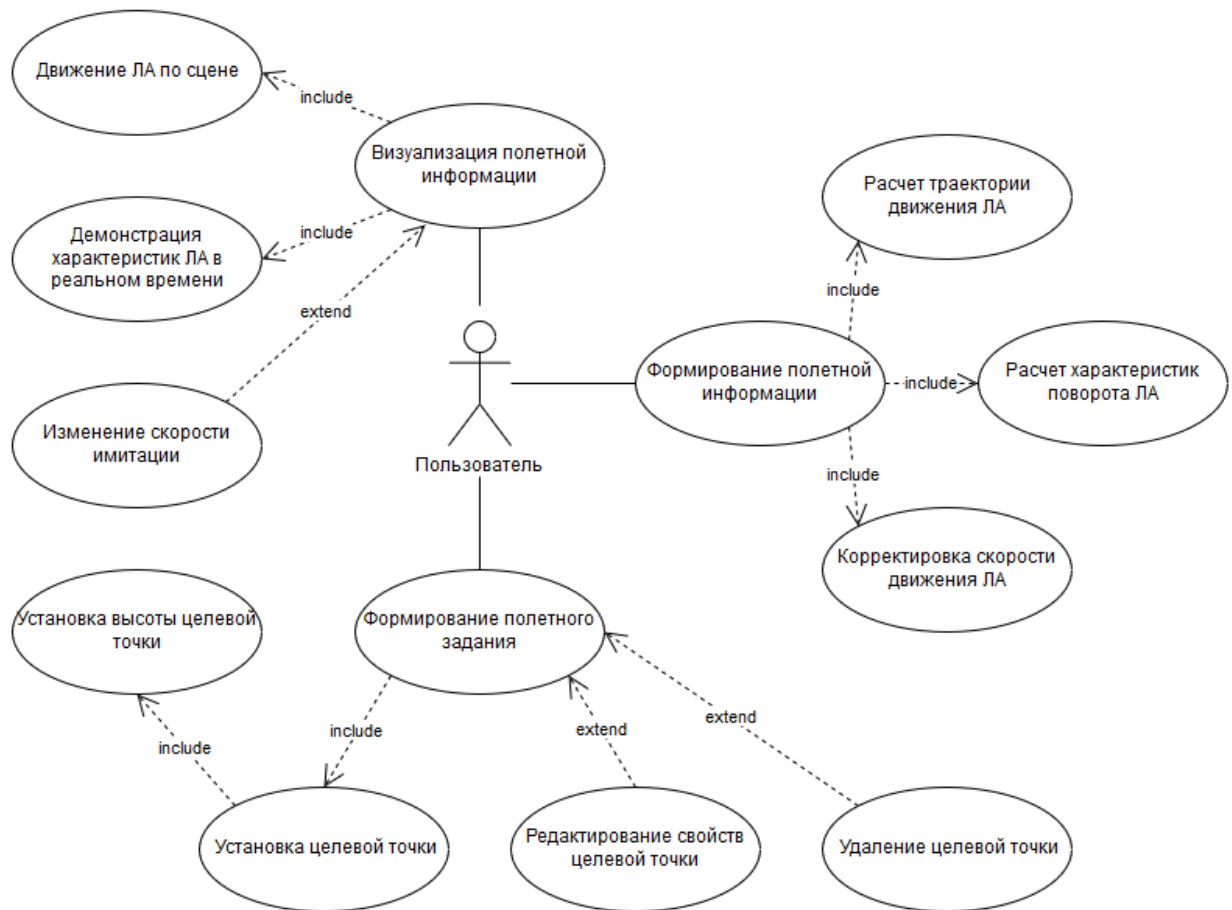


Рисунок 2.2.1 – Диаграмма вариантов использования

Целевой пользователь при использовании подсистемы имитации движения летательного аппарата имеет следующие возможности:

1) Сформировать полетное задание. Расположить на сцене последовательность целевых точек, через которые летательному аппарату необходимо пролететь.

2) Сформировать полетную информацию. Получить полный список характеристик летательного аппарата в каждый момент времени имитации пролета. Полетная информация будет храниться в виде текстового файла, каждая строка которого содержит отдельную временную точку.

3) Визуализировать полетную информацию. На основе полученного массива данных полетной информации возможно смоделировать пролет летательного аппарата в реальном времени.

Формирование полётного задания предполагает:

- 1) Обязательное действие пользователя по установке целевой точки как проекции на горизонтальную плоскость.
- 2) Обязательное действие пользователя по установке высоты целевой точки.
- 3) Пользователь имеет возможность изменить характеристики целевой точки после установки.
- 4) Пользователь имеет возможность удалить целевую точку.

Формирование полетной информации предполагает:

- 1) Расчет траектории движения летательного аппарата.
- 2) Расчет характеристик поворота летательного аппарата (курс, тангаж, крен).
- 3) Корректировку скорости летательного аппарата, т.к. в процессе имитации движения летательный аппарат может превысить максимальную допустимую скорость, поэтому необходима корректировка.

Визуализация полетной информации предполагает:

- 1) Анимацию движения летательного аппарата по сцене.
- 2) Демонстрацию характеристик летательного аппарата в каждый момент времени.
- 3) Пользователь имеет возможность изменять скорость имитации движения летательного аппарата.

2.3 Методика расчета полетных данных

На основе вышеописанной математической модели, разработана методика расчета полетных данных (рисунок 2.3.1), которая содержит следующие стадии:

- 1) Расчет коэффициентов кубического сплайна. На основе списка узловых точек получаем кубический сплайн, который рассматривается как годограф радиус-вектора, содержащий маршрут движения летательного аппарата. На данной стадии время принимается как аккумулярующее значение, получаемое как сумма расстояний между соседними точками.

2) Вычисление скорости. Так как на предыдущем этапе время бралось как сумма между соседними точками, что может быть не корректно для конкретного случая, т.к. на маршруте пролета летательного аппарата могут возникнуть ситуации, когда скорость превышает максимальную. Для выявления данных нарушений необходимо найти максимальную скорость движения летательного аппарата на всем маршруте.

3) Корректировка скорости. Если максимальная скорость выходит за диапазон допустимых скоростей, то необходимо выполнить корректировку, т.е. увеличить время движения летательного аппарата по всему маршруту и рассчитать коэффициенты сплайна повторно, основываясь на новых значениях времени. Если максимальная скорость находится в диапазоне допустимых скоростей, то корректировка не требуется.

4) Определение орт трехгранника Френе. После получения корректного по скорости кубического сплайна, необходимо вычислить орты трехгранника Френе, которые используются для расчета характеристик поворота летательного аппарата. Трехгранник Френе вычисляется для каждой отдельной функции, которые образуют кубический сплайн.

5) Расчет характеристик поворота летательного аппарата. С помощью полученных орт трехгранника Френе выполняется расчет углов поворота летательного аппарата (курса, крена, тангажа).

Основываясь на данной методике, получаем все необходимые полетные данные для каждого момента имитационного времени, которые рассчитываются с помощью кубического сплайна и соответствующих трехгранников Френе.



Рисунок 2.3.1 – Методика расчета полетных данных

2.4 Разработка информационных спецификаций

Выделим в подсистеме следующие информационные объекты:

- 1) Математическое ядро. Сущность, которая содержит основной математический функционал.
- 2) Кусочная функция. Сущность, хранящая функции, используемые для расчета полетной информации на всем маршруте.
- 3) Элемент кусочной функции. Функция, используемая для расчета полетной информации на отдельном временном промежутке.
- 4) Трехгранник Френе. Назначается каждому элементу кусочной функции, используется для расчета характеристик поворота летательного аппарата.
- 5) Параметрическое уравнение. Используется для параметрического представления уравнения.
- 6) Позиция точки в пространстве. Сущность, хранящая значения координат точки в пространстве.
- 7) Математические операции и функции. Содержит в себе допустимые к использованию в подсистеме математические операции и функции, такие как, сложение, вычитание, деление, умножение, синус, косинус и т.д.
- 8) Полетная информация. Хранит в себе основные характеристики летательного аппарата в момент времени, к характеристикам относятся:

- позиция в пространстве;
- модельное время;
- угловая скорость;
- курс;
- крен;
- тангаж.

9) Отладочный функционал. Процедуры и функции, которые понадобятся на этапе реализации, для ускорения процесса поиска и устранения ошибок.

10) Элемент дерева операций. Используется для представления математических функций в программе.

11) Содержимое элемента дерева операций. Содержит информационную нагрузку на каждый отдельный элемент дерева операций, может хранить: вещественные значения, имена переменных, математическую операцию или математическую функцию.

Для демонстрации взаимосвязи между информационными объектами разработаем контекстную диаграмму классов, которая представлена на рисунке 2.4.1.

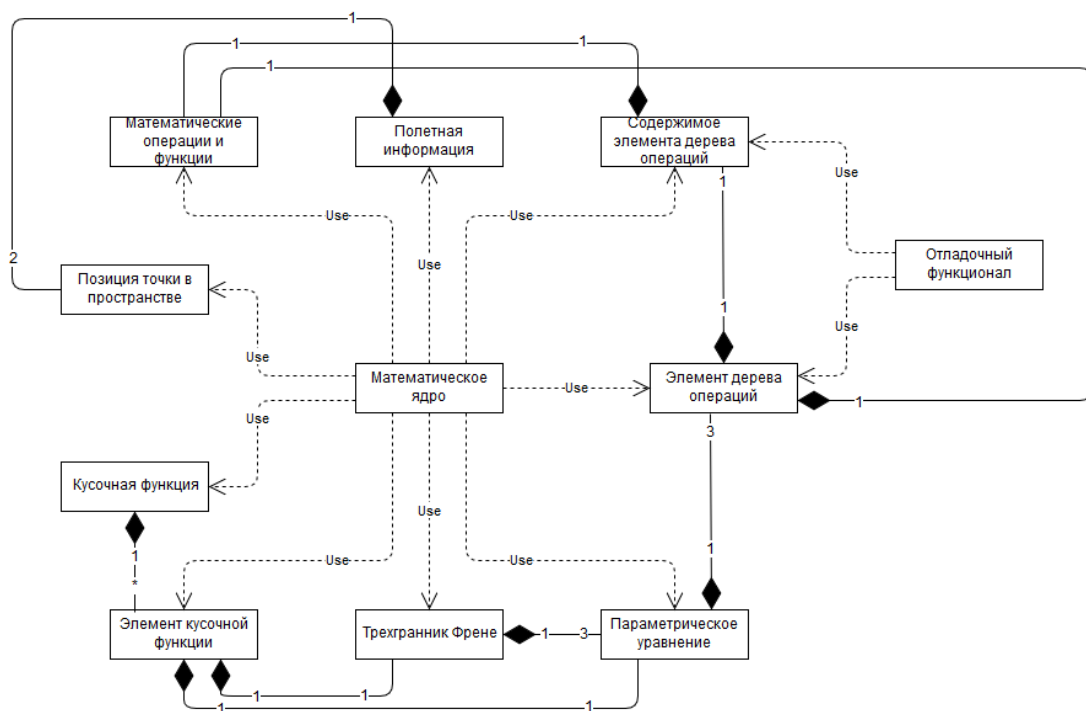


Рисунок 2.4.1 – Контекстная диаграмма классов

Центром всей системы является математическое ядро, оно выполняет необходимые расчеты и формирует конечный результат в виде списка объектов полетной информации. На вход математическому ядру поступает список точек в пространстве, далее формируется кусочная функция, которая состоит из некоторого количества элементов кусочной функции. В свою очередь элемент кусочной функции содержит элемент кубического сплайна (кубическая парабола), представляемая в параметрической форме, и трехгранник Френе (сопровождающий трехгранник) для конкретного участка маршрута, т.е. для конкретной кубической параболы. Параметрические уравнения хранятся с помощью деревьев операций. Отладочный функционал работает над уравнениями в системе, т.е. способствует отладке математических операций.

2.5 Разработка поведенческих спецификаций

Для структуризации поведения системы на определенные действия пользователя необходимо предоставить диаграмму состояний подсистемы имитации движения летательного аппарата, разработанная диаграмма состояний представлена на рисунке 2.5.1.

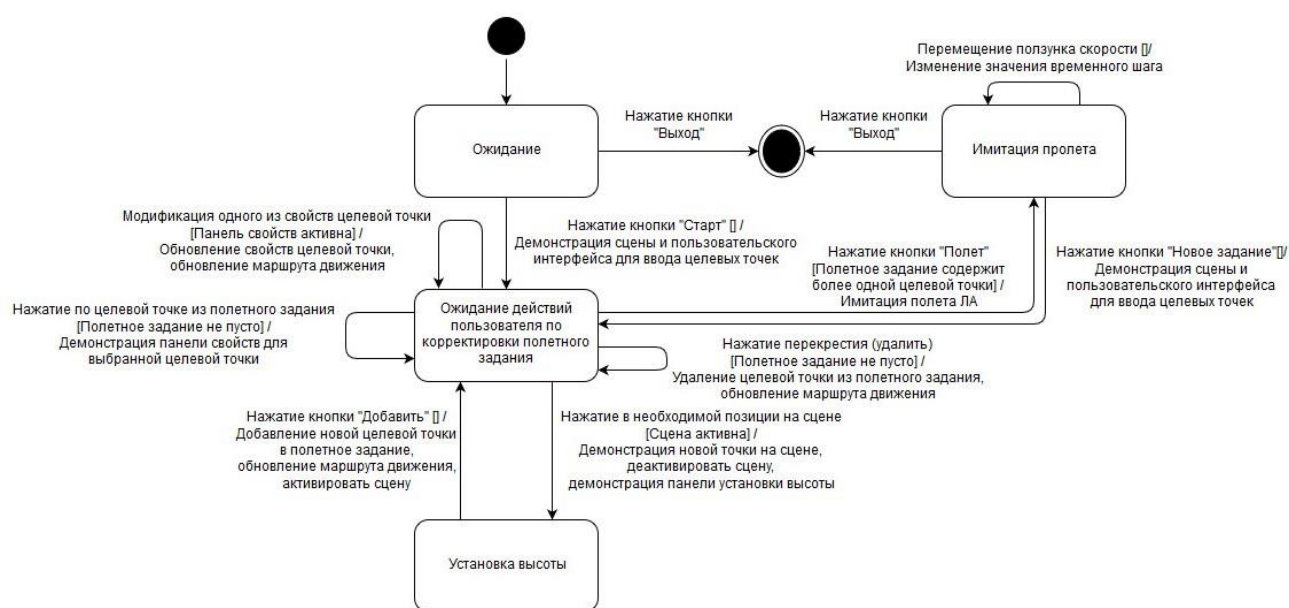


Рисунок 2.5.1 – Диаграмма состояний

Диаграмма содержит следующие состояния:

1) Ожидание. Состояние подсистемы после запуска приложения, пользовательский интерфейс содержит две кнопки: «Старт» и «Выход». Кликнув по кнопке «Выход» приложение закроется, и освободятся все занимаемые ресурсы, кликнув по кнопке «Старт» осуществиться переход в состояние «Ожидание действий пользователя по корректировке полетного задания», загрузиться сцена создания полетного задания и отобразиться интерфейс редактирования полетного задания.

2) Ожидание действий пользователя по корректировке полетного задания. Состояние подсистемы при создании и редактировании полетного задания, интерфейс предполагает следующие возможности:

- установка целевой точки как проекции на горизонтальную плоскость, при этом сцена должна быть активна, в результате на сцене отобразиться новая целевая точка на высоте 10 метров от подстилающей поверхности, сцена станет не активной для добавления новых точек, т.к. осуществиться переход в состояние «Установка высоты»;

- модификация (изменение, редактирование) целевой точки из полетного задания, при этом панель свойств должна быть активной, изменения, происходящие на панели свойств, отображаются в полетном задании и предварительном маршруте движения летательного аппарата, изменения применяются по нажатию кнопки «Enter»;

- открытие панели свойств для конкретной целевой точки, при этом полетное задание должно содержать минимум одну целевую точку;

- удаление целевой точки из полетного задания, для этого необходимо нажать кнопку с перекрестием напротив удаляемой целевой точки (полетное задание должно быть не пустое), удаление точки приводит к изменению предварительного маршрута движения летательного аппарата;

- получить полетную информацию для разработанного полетного задания и визуализировать пролет летательного аппарата по полученной полетной информации, при этом полетное задание должно содержать

минимум две целевых точек, подсистема перейдет в состояние «Имитации полета».

3) Установка высоты. Состояние подсистемы при установке высоты для добавляемой целевой точки. После установки необходимой высоты и нажатии на кнопку «Добавить» осуществляется переход в состояние «Ожидание действий пользователя по корректировке полетного задания», в полетное задание добавляется новая целевая точка, предварительный маршрут движения летательного аппарата обновляется, а сцена становится активной для добавления следующей целевой точки.

4) Имитация пролета. Состояние подсистемы во время имитации движения летательного аппарата по полученной полетной информации. Пользователь имеет возможность:

- создать новое полетное задание, нажав кнопку «Новое задание» осуществиться переход в состояние «Ожидание действий пользователя по корректировке полетного задания», загрузиться сцена создания полетного задания и отобразиться интерфейс редактирования полетного задания;

- изменять скорость имитации движения летательного аппарата, для этого необходимо переместить ползунок, значения которого варьируются от 1 до 100, в результате измениться временной шаг получения полетной информации.

3 ПРОЕКТИРОВАНИЕ ПОДСИСТЕМЫ ИМИТАЦИИ ДВИЖЕНИЯ ЛЕТАТЕЛЬНОГО АППАРАТА

3.1 Проектирование структуры подсистемы

Выделим в подсистеме следующие классы и наделим их соответствующими атрибутами и операциями, описание которых будет представлено далее:

1) MathCore. Класс, который содержит основной математический функционал.

2) PiecewiseFunction. Класс, хранящий функции, используемые для расчета полетной информации на всем маршруте.

3) ElementPiecewiseFunction. Класс, хранящий функцию, используемую для расчета полетной информации на отдельном временном промежутке.

4) TrihedronFrene. Класс, назначается каждому элементу кусочной функции, используется для расчета характеристик поворота летательного аппарата.

5) ParametricRepresentation. Класс используется для параметрического представления уравнения.

6) Vector3. Класс, хранящий значения координат точки в пространстве.

7) ElementsEquation. Класс содержит в себе допустимые к использованию в подсистемы математические операции и функции, такие как, сложение, вычитание, деление, умножение, синус, косинус и т.д.

8) FlightInformation. Класс хранит в себе основные характеристики летательного аппарата в момент времени.

9) AuxiliaryFunction. Класс, содержащий процедуры и функции, которые понадобятся на этапе реализации, для ускорения процесса поиска и устранения ошибок.

10) ElementTreeNonlinearEquation. Класс используется для представления математических функций в программе.

11) EquationComponent. Класс содержит информационную нагрузку на каждый отдельный элемент дерева операций, может хранить: вещественные

значения, имена переменных, математическую операцию или математическую функцию.

Для графического отображения объектно-ориентированной модели воспользуемся диаграммой классов, которая демонстрирует структуру классов, атрибутов, методов и взаимосвязь между ними, на рисунке 3.1.1 представлена разработанная физическая диаграмма классов.

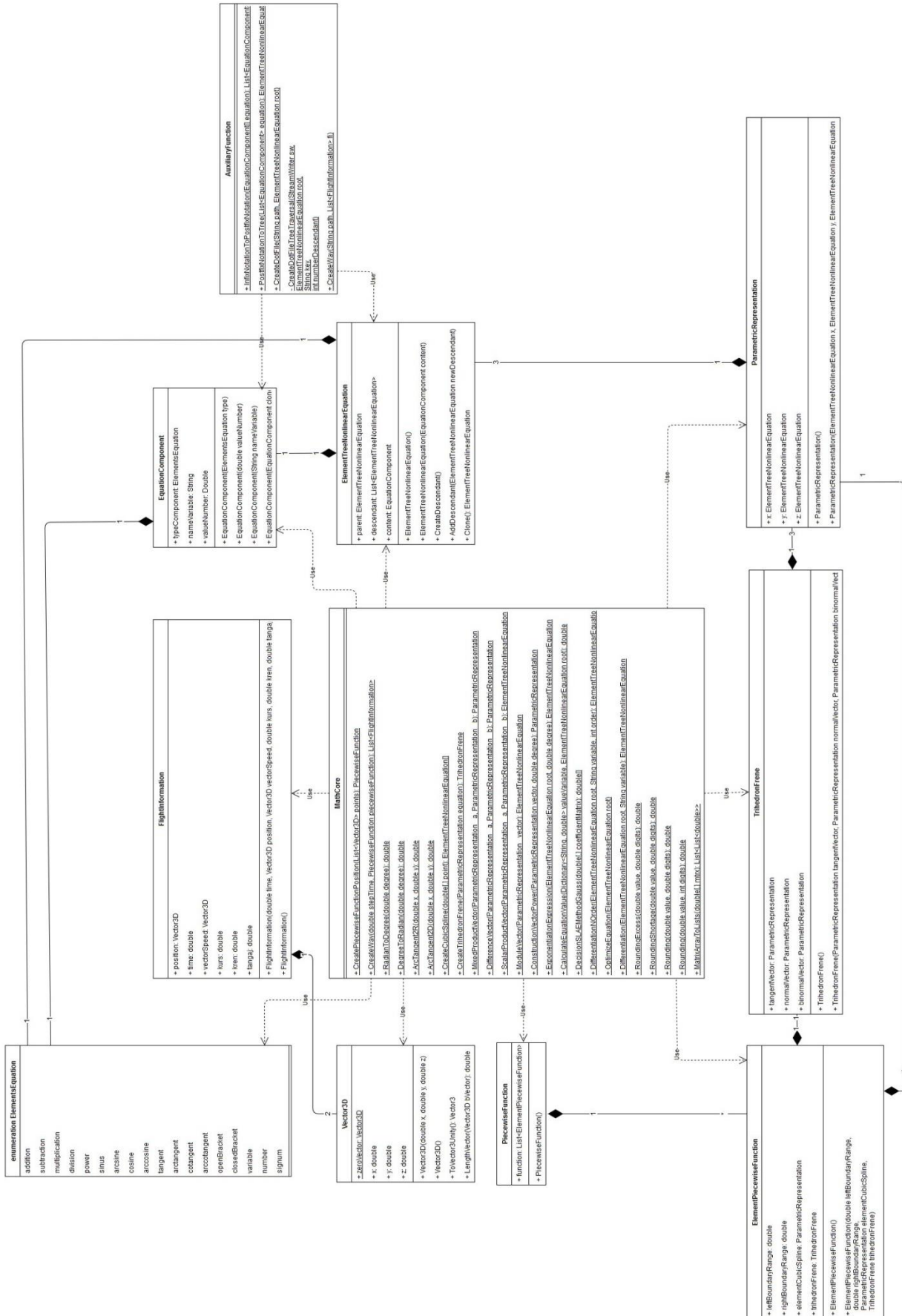


Рисунок 3.1.1 – Физическая диаграмма классов

3.2 Проектирование компонентов подсистемы

Для разработанной диаграммы классов необходимо привести проектные спецификации классов, т.е. описать назначение и содержимое каждого из них. Далее на естественном языке дается описание каждого класса с диаграммы классов, которая представлена на рисунке 3.1.1.

AuxiliaryFunction

Статический класс, созданный специально для отладки промежуточных результатов разработки. Включает в себя следующие методы:

1) `InfixNotationToPostfixNotation`. Статический открытый метод для перевода выражения из инфиксной нотации в постфиксную по алгоритму Эдсгера Дейкстры, который называется «сортировочная станция». Метод принимает один аргумент, это массив экземпляров класса `EquationComponent` (описание данного класса приведено далее), данный массив представляет выражение в инфиксной форме. Главное преимущество постфиксной формы записи выражений заключается в однозначности порядка выполнения операций, благодаря этому отпадает необходимость использования скобок, введение приоритетов и ассоциативности операций. Данный метод будет применяться для определения порядка выполняемых операций в тестовых выражениях, которые будут использоваться на этапе отладки для выявления ошибок в алгоритмах. Метод возвращает список экземпляров `EquationComponent`, который представляет исходное выражение в постфиксной форме.

2) `PostfixNotationToTree`. Статический открытый метод для перевода выражения из постфиксной формы записи в дерево операций. Все математические выражения в программе будут представлены в виде деревьев операций, данная структура данных будет описана далее. Преимуществом деревьев операций является удобство их обработки и изменения выражений, представленных в них. Метод принимает список экземпляров класса `EquationComponent`, который будет получен с помощью метода `InfixNotationToPostfixNotation`, описанного ранее. Возвращает метод ссылку на корень дерева операций, который представлен

экземпляром класса `ElementTreeNonlinearEquation` (описание данного класса приведено далее).

3) `CreateDotFile`. Статический открытый метод для представления дерева операций в виде текстового файла, содержащего описание исходного дерева на языке DOT. DOT – это язык описания графов в понятном для человека виде. Файл, содержащий граф на языке DOT имеют расширения `.gv` или `.dot`, с помощью такого файла посредством специальных программ можно сгенерировать изображение исходного графа, именно это особенность и понадобится на этапе отладки, т.к. будет наглядно виден полученный результат от применения того или иного алгоритма. Метод принимает два параметра:

- строку, представляющую путь к файлу для записи с расширением `.dot` (если файл не существует, то он будет создан, иначе перезаписан);
- ссылка на корень дерева, представленная в виде экземпляра класса `ElementTreeNonlinearEquation`.

Метод ничего не возвращает.

4) `CreateDotFileTreeTraversal`. Статический приватный метод, выполняет прямой обход дерева операций и формирует `dot` файл для текстового представления дерева, вызывается из метода `CreateDotFile`. Метод принимает четыре параметра:

- ссылка на поток записи;
- ссылка на корень дерева, представленная в виде экземпляра класса `ElementTreeNonlinearEquation`;
- строку – ключ, которая используется при формировании уникального имени для каждой вершины дерева;
- целое число, указывающее каким по счету наследником идет конкретная вершина, применяется также при формировании уникального имени каждой вершины дерева.

Метод ничего не возвращает.

5) `CreateWay`. Статический открытый метод для формирования полного отчета по имитации движения летательного аппарата в закабинном пространстве.

Результатом выполнения данного метода является текстовый файл, каждая строка которого хранит полётную информацию в каждую единицу времени. Строка файла имеет следующий формат:

- целое число, представляющее порядковый номер полетной информации;
- вещественное число, представляющее модельное время;
- вещественное число, представляющее позицию летательного аппарата по X координате;
- вещественное число, представляющее позицию летательного аппарата по Y координате;
- вещественное число, представляющее позицию летательного аппарата по Z координате;
- вещественное число, представляющее угловую скорость летательного аппарата по X координате;
- вещественное число, представляющее угловую скорость летательного аппарата по Y координате;
- вещественное число, представляющее угловую скорость летательного аппарата по Z координате;
- вещественное число, представляющее угол курса в радианах;
- вещественное число, представляющее угол крена в радианах;
- вещественное число, представляющее угол тангажа в радианах.

Каждое значение в строке разделяется знаком табуляции. Метод принимает два параметра:

- строку, представляющую путь к файлу для записи с расширением .txt (если файл не существует, то он будет создан, иначе перезаписан);
- список экземпляров класса FlightInformation (описание данного класса приведено далее), в котором хранятся все характеристики летательного аппарата за все время имитации полета.

Метод ничего не возвращает.

ElementPiecewiseFunction

Класс, представляющий элемент кусочно-заданной функции. Кусочно-заданная функция – это функция, которая задана различными формулами на разных интервалах. Записывается в виде:

$$f(x) = \begin{cases} f_0(x), x < x_1 \\ f_1(x), x_1 < x < x_2 \\ \dots \\ f(x), x_n < x \end{cases} \quad (30)$$

Иначе говоря, экземпляр класса `ElementPiecewiseFunction` это одно из выражений вида $f_i(x), x_i < x < x_{i+1}$.

Класс содержит следующие атрибуты:

1) `leftBoundaryRange`. Вещественное число, обозначающее левую границу интервала. Атрибут открыт.

2) `rightBoundaryRange`. Вещественное число, обозначающее правую границу интервала. Атрибут открыт.

3) `elementCubicSpline`. Экземпляр класса `ParametricRepresentation` (описание данного класса приведено далее), который хранит функцию в параметрическом представлении. Атрибут открыт.

4) `trihedronFrene`. Экземпляр класса `TrihedronFrene` (описание данного класса приведено далее), который хранит трехгранник Френе для функции, хранящейся в `elementCubicSpline`. Атрибут открыт.

Класс содержит следующие конструкторы:

1) Открытый конструктор, принимает четыре параметра:

- вещественное число, обозначающее левую границу интервала;
- вещественное число, обозначающее правую границу интервала;
- экземпляр класса `ParametricRepresentation`, который хранит функцию в параметрическом представлении;
- экземпляр класса `TrihedronFrene`, который хранит трехгранник Френе для функции, хранящейся в `elementCubicSpline`.

Конструктор проводит инициализацию соответствующих атрибутов экземпляра.

2) Открытый конструктор, не принимает параметров, применяется только для создания экземпляра класса.

ElementsEquation

Класс-перечисление, содержит: арифметические операции, тригонометрические функции, обратные тригонометрические функции и прочие части, которые могут участвовать в построении формул. Далее перечислены все элементы данного класса:

- 1) addition. Сложение.
- 2) subtraction. Разность.
- 3) multiplication. Произведение.
- 4) division. Деление.
- 5) power. Степень.
- 6) sinus. Синус.
- 7) arcsine. Арксинус.
- 8) cosine. Косинус.
- 9) arccosine. Арккосинус.
- 10) tangent. Тангенс.
- 11) arctangent. Арктангенс.
- 12) cotangent. Котангенс.
- 13) arccotangent. Арккотангенс.
- 14) openBracket. Открывающаяся скобка.
- 15) closedBracket. Закрывающаяся скобка.
- 16) variable. Переменная.
- 17) number. Число.
- 18) signum. Сигнум.

ElementTreeNonlinearEquation

Класс, представляющий узел дерева операций. Все функции в разрабатываемой подсистеме будут храниться в деревьях операций, описание этой структуры данных приведено далее.

Класс содержит следующие атрибуты:

- 1) parent. Экземпляр класса `ElementTreeNonlinearEquation`, ссылается на узел «родителя». Атрибут открыт.
- 2) descendant. Список экземпляров класса `ElementTreeNonlinearEquation`, хранит в себе ссылки на узлы всех «наследников». Атрибут открыт.
- 3) content. Экземпляр класса `EquationComponent` (описание данного класса приведено далее), который хранит информационную составляющую узла. Атрибут открыт.

Класс содержит следующие конструкторы:

- 1) Открытый конструктор, не принимающий параметров, выполняет создание пустого узла и обнуление информационной составляющей.
- 2) Открытый конструктор, принимает один параметр - экземпляр класса `EquationComponent`, выполняет создание узла и инициализирует атрибут content.

Класс содержит следующие методы:

- 1) `CreateDescendant`. Открытый метод, ничего не возвращает, не принимает параметров, выполняет первичную инициализацию аргумента descendant.
- 2) `AddDescendant`. Открытый метод добавления нового «наследника» текущего узла. Принимает один параметр – экземпляр класса `ElementTreeNonlinearEquation`, указывает на узел, который станет наследником. Метод ничего не возвращает.
- 3) `Clone`. Открытый метод, для создания полной копии текущего узла, не принимает параметров, возвращает клон узла.

EquationComponent

Класс, представляющий информационную составляющую узла дерева операций.

Класс содержит следующие атрибуты:

- 1) typeComponent. Экземпляр класса `ElementsEquation`, в котором храниться к какому типу относится хранимая информация (переменная, число, математическая функция, арифметическая операция). Атрибут открыт.

2) `nameVariable`. Опциональный атрибут, представленный строкой, в которой храниться имя переменной, если атрибут `typeComponent` имеет значение `variable`. Атрибут открыт.

3) `valueNumber`. Опциональный атрибут, представленный вещественным числом, в котором храниться константное значение, если атрибут `typeComponent` имеет значение `number`. Атрибут открыт.

Класс содержит следующие конструкторы:

1) Открытый конструктор, принимает один параметр – экземпляр класса `ElementsEquation`, выполняет создание информационной части и инициализирует атрибут `typeComponent` переданным значением.

2) Открытый конструктор, принимает один параметр – вещественное число, выполняет создание информационной части, инициализирует атрибут `typeComponent` значением `number` и присваивает переданное значение атрибуту `valueNumber`.

3) Открытый конструктор, принимает один параметр – строку (имя переменной), выполняет создание информационной части, инициализирует атрибут `typeComponent` значением `variable` и присваивает переданную строку атрибуту `nameVariable`.

4) Открытый конструктор, принимает один параметр – экземпляр класса `EquationComponent`, выполняет создание информационной части и инициализирует атрибуты теми же значениями что и в переданном параметре, т.е. создает копию.

FlightInformation

Класс, представляющий полетную информацию, т.е. характеристики летательного аппарата в единицу времени.

Класс содержит следующие атрибуты:

1) `position`. Экземпляр класса `Vector3D` (описание данного класса приведено далее), в котором храниться позиция летательного аппарата в пространстве. Атрибут открыт.

2) `time`. Вещественное число, хранящее модельное время, т.е. время, в котором летательный аппарат имеет те или иные характеристики. Атрибут открыт.

3) `vectorSpeed`. Экземпляр класса `Vector3D`, в котором хранятся угловые скорости летательного аппарата. Атрибут открыт.

4) `kurs`. Вещественное число, хранящее угол курса летательного аппарата в радианах. Атрибут открыт.

5) `kren`. Вещественное число, хранящее угол крена летательного аппарата в радианах. Атрибут открыт.

6) `tangaj`. Вещественное число, хранящее угол тангажа летательного аппарата в радианах. Атрибут открыт.

Класс содержит следующие конструкторы:

1) Открытый конструктор, принимающий следующие параметры:

- вещественное число, отражающее модельное время;
- экземпляр класса `Vector3D`, отражающий позицию летательного аппарата в пространстве;
- экземпляр класса `Vector3D`, отражающий угловые скорости летательного аппарата;
- вещественное число, отражающее угол курса в радианах;
- вещественное число, отражающее угол крена в радианах;
- вещественное число, отражающее угол тангажа в радианах.

Конструктор выполняет создание экземпляра и последующую инициализацию соответствующих атрибутов.

2) Открытый конструктор, не принимающий параметров, выполняет создание пустого экземпляра класса.

MathCore

Статический класс, хранящий весь базовый функционал математического ядра.

Класс содержит следующие методы:

1) `MatrixArrayToLists`. Открытый статический метод перевода матрицы представленной с помощью двумерного массива, в матрицу, представленную с помощью вложенных списков. Метод принимает двумерный массив вещественных чисел, а возвращает список вложенных списков, элементами которых являются вещественные числа.

2) `Rounding`. Открытый статический метод округления вещественных чисел, принимает следующие параметры:

- вещественно число, которое будет округляться;
- целое число, отражающее необходимое количество знаков в дробной части.

Возвращает округленное исходное число с заданной точностью.

3) `Rounding`. Открытый статический метод округления вещественных чисел, принимает следующие параметры:

- вещественно число, которое будет округляться;
- вещественное число, отражающее необходимую точность.

Возвращает округленное исходное число с заданной точностью.

4) `RoundingShortage`. Открытый статический метод округления вещественных чисел по недостатку, принимает следующие параметры:

- вещественное число, которое будет округляться;
- вещественное число, отражающее погрешность округления.

Возвращает округленное исходное число с заданной погрешностью.

5) `RoundingExcess`. Открытый статический метод округления вещественных чисел по избытку, принимает следующие параметры:

- вещественное число, которое будет округляться;
- вещественное число, отражающее погрешность округления.

Возвращает округленное исходное число с заданной погрешностью.

6) `Differentiation`. Открытый статический метод, выполняющий дифференцирование первого порядка функции, которая представлена с помощью дерева операций. Метод принимает следующие параметры:

- экземпляр класса `ElementTreeNonlinearEquation`, который является корнем дерева операций;
- строка, в которой хранится имя переменной, по которой осуществляется дифференцирование.

Метод возвращает экземпляр класса `ElementTreeNonlinearEquation`, являющийся корнем дерева операция, с помощью которого представлена функция после дифференцирования.

7) `OptimizeEquation`. Открытый статический метод, выполняющий оптимизацию дерева операций. Под оптимизацией в данном случае понимается удаление из дерева незначущих операций, таких как:

- сложение с нулем;
- вычитание нуля;
- умножение на ноль;
- умножение на единицу;
- деление на единицу;
- возведение в первую степень;
- возведение в нулевую степень.

Метод принимает экземпляр класса `ElementTreeNonlinearEquation`, указывающий на вершину дерева операций, возвращает также экземпляр класса `ElementTreeNonlinearEquation`, в котором храниться вершина оптимизированного дерева операций.

8) `DifferentiationNOrder`. Открытый статический метод, выполняющий дифференцирование n-го порядка функции, которая представлена с помощью дерева операций, является оболочкой для метода `Differentiation`. Метод принимает следующие параметры:

- экземпляр класса `ElementTreeNonlinearEquation`, который является корнем дерева операций;
- строка, в которой хранится имя переменной, по которой осуществляется дифференцирование;

- целое число, отражающее необходимый порядок дифференцирования.

Метод возвращает экземпляр класса `ElementTreeNonlinearEquation`, являющийся корнем дерева операция, с помощью которого представлена функция после дифференцирования.

9) `DecisionSLAEMethodGauss`. Открытый статический метод, реализующий алгоритм Карла Фридриха Гауса для решения систем линейных алгебраических уравнений (данный алгоритм описан далее). Метод принимает один параметр – двумерный массив вещественных чисел, в котором храниться расширенная матрица системы. Возвращает метод массив вещественных чисел, которые являются корнями системы уравнений.

10) `CalculateEquationValue`. Открытый статический метод, вычисляющий значение функции при заданных значениях переменных. Функция представляется в виде дерева операций. Метод принимает следующие параметры:

- ассоциативный массив, в котором ключом является строка, хранящая имя переменной, а значение представлено вещественным числом;
- экземпляр класса `ElementTreeNonlinearEquation`, который является корнем дерева операций.

Метод возвращает значение функции при заданных значениях переменных в виде вещественного числа.

11) `ExponentiationExpression`. Открытый статический метод, возводящий выражение в n-ю степень. Метод принимает следующие параметры:

- экземпляр класса `ElementTreeNonlinearEquation`, который является корнем дерева операций;
- вещественное число, которое указывает, в какую степень необходимо возвести выражение.

Метод возвращает экземпляр класса `ElementTreeNonlinearEquation`, являющийся корнем дерева операция, с помощью которого представлена функция после возведения в заданную степень.

12) `ConstructionVectorPower`. Открытый статический метод, возводящий вектор-функцию в n -ю степень. Вектор-функция представляется в параметрическом виде с помощью класса `ParametricRepresentation` (описание данного класса приведено далее). Метод принимает следующие параметры:

- экземпляр класса `ParametricRepresentation`, с помощью которого представлена вектор-функция;
- вещественное число, которое указывает, в какую степень необходимо возвести вектор-функцию.

Метод возвращает экземпляр класса `ParametricRepresentation`, с помощью которой представлена исходная вектор-функция возведенная в заданную степень.

13) `ModuleVector`. Открытый статический метод, используемый для создания выражения расчета модуля вектор-функции. Вектор-функция представляется в параметрическом виде с помощью класса `ParametricRepresentation` (описание данного класса приведено далее). Метод принимает один параметр - экземпляр класса `ParametricRepresentation`, с помощью которого представлена вектор-функция, а возвращает экземпляр класса `ElementTreeNonlinearEquation`, являющийся корнем дерева операция, с помощью которого представлено выражение для расчета модуля исходной вектор-функции.

14) `ScalarProductVector`. Открытый статический метод, используемый для создания выражений расчета скалярного произведения векторов, которые представлены вектор-функциями в параметрическом виде с помощью класса `ParametricRepresentation` (описание данного класса приведено далее). Метод принимает следующие параметры:

- экземпляр класса `ParametricRepresentation`, с помощью которого представлена первая вектор-функция;
- экземпляр класса `ParametricRepresentation`, с помощью которого представлена вторая вектор-функция.

Метод возвращает экземпляр класса `ElementTreeNonlinearEquation`, являющийся корнем дерева операция, с помощью которого представлено выражение для расчета скалярного произведения векторов.

15) `DifferenceVector`. Открытый статический метод, используемый для создания выражений расчета разности векторов, которые представлены вектор-функциями в параметрическом виде с помощью класса `ParametricRepresentation` (описание данного класса приведено далее). Метод принимает следующие параметры:

- экземпляр класса `ParametricRepresentation`, с помощью которого представлена первая вектор-функция;
- экземпляр класса `ParametricRepresentation`, с помощью которого представлена вторая вектор-функция.

Метод возвращает экземпляр класса `ElementTreeNonlinearEquation`, являющийся корнем дерева операция, с помощью которого представлено выражение для расчета разности векторов.

16) `MixedProductVector`. Открытый статический метод, используемый для получения смешанного произведения векторов, которые представлены вектор-функциями в параметрическом виде с помощью класса `ParametricRepresentation` (описание данного класса приведено далее). Метод принимает следующие параметры:

- экземпляр класса `ParametricRepresentation`, с помощью которого представлена первая вектор-функция;
- экземпляр класса `ParametricRepresentation`, с помощью которого представлена вторая вектор-функция.

Метод возвращает экземпляр класса `ParametricRepresentation`, в котором хранится вектор, представляющий результат смешанного произведения.

17) `CreateTrihedronFrene`. Открытый статический метод, создающий трехгранник Френе для конкретной функции, представленной в параметрическом виде (алгоритм получения трехгранника Френе описан далее). Метод принимает один параметр – экземпляр класса `ParametricRepresentation` (описание данного класса приведено далее), в котором хранится исходная функция. Возвращает метод экземпляр класса `TrihedronFrene` (описание данного класса приведено далее), в котором хранится полученный трехгранник.

18) `CreateCubicSpline`. Открытый статический метод, создающий кубический сплайн (алгоритм создания кубического сплайна описан далее). Метод принимает один параметр – двумерный массив вещественных чисел, в котором хранятся опорные точки, в следующем формате $[i, 0] = x$, $[i, 1] = f(x)$. Возвращает метод массив экземпляров класса `ElementTreeNonlinearEquation`, которые указывают на вершины деревьев операций для полученного кубического сплайна.

19) `ArcTangent2D`. Открытый статический метод, используемый для получения угла в градусах, тангенс которого равен отношению двух передаваемых координат. Метод принимает следующие параметры:

- вещественное число, отражающее координату по оси абсцисс;
- вещественное число, отражающее координату по оси ординат.

Метод возвращает вещественное число равное углу в градусах.

20) `ArcTangent2R`. Открытый статический метод, используемый для получения угла в радианах, тангенс которого равен отношению двух передаваемых координат. Метод принимает следующие параметры:

- вещественное число, отражающее координату по оси абсцисс;
- вещественное число, отражающее координату по оси ординат.

Метод возвращает вещественное число равное углу в радианах.

21) `DegreeToRadian`. Открытый статический метод, используемый для перевода значения угла из градусов в радианы. Метод принимает один параметр – вещественное число, отражающее значение угла в градусах, а возвращает вещественное число, отражающее значение угла в радианах.

22) `RadianToDegree`. Открытый статический метод, используемый для перевода значения угла из радиан в градусы. Метод принимает один параметр – вещественное число, отражающее значение угла в радианах, а возвращает вещественное число, отражающее значение угла в градусах.

23) `CreateWay`. Открытый статический метод, формирующий полный отчет по имитации полета летательного аппарата. Метод принимает следующие параметры:

- вещественное число, отражающее временной шаг, через который необходимо получать полетную информацию;

- экземпляр класса `PiecewiseFunction` (описание данного класса приведено далее), в котором храниться кусочная функция для всего маршрута летательного аппарата.

Метод возвращает список экземпляров класса `FlightInformation`, в котором храниться характеристики летательного аппарата за время всей имитации.

24) `CreateWay`. Открытый статический метод, формирующий полный отчет по имитации пролета летательного аппарата. Метод принимает следующие параметры:

- список экземпляров класса `Vector3D` (описание данного класса приведено далее), в котором хранятся опорные точки, которые необходимо посетить летательному аппарату во время имитации пролета;

- вещественное число, отражающее временной шаг, через который необходимо получать полетную информацию;

Метод возвращает список экземпляров класса `FlightInformation`, в котором храниться характеристики летательного аппарата за время всей имитации.

25) `CreatePiecewiseFunctionPosition`. Открытый статический метод, создающий кусочную функцию и трехгранники Френе для всех функций, участвующих в построении маршрута движения летательного аппарата. Метод принимает один параметр - список экземпляров класса `Vector3D` (описание данного класса приведено далее), хранящий опорные точки, которые необходимо посетить летательному аппарату во время имитации пролета, а возвращает экземпляр класса `PiecewiseFunction` (описание данного класса приведено далее), хранящий кусочную функцию.

ParametricRepresentation

Класс, используемый для представления функций в параметрическом виде:

$$\begin{cases} x = x(t), \\ y = y(t), \\ z = z(t). \end{cases} \quad (31)$$

Класс содержит следующие атрибуты:

1) x . Экземпляр класса `ElementTreeNonlinearEquation`, который является вершиной дерева операций, содержащего функцию расчета координаты x , т.е. данный атрибут представляет $x = x(t)$. Атрибут открыт.

2) y . Экземпляр класса `ElementTreeNonlinearEquation`, который является вершиной дерева операций, содержащего функцию расчета координаты y , т.е. данный атрибут представляет $y = y(t)$. Атрибут открыт.

3) z . Экземпляр класса `ElementTreeNonlinearEquation`, который является вершиной дерева операций, содержащего функцию расчета координаты z , т.е. данный атрибут представляет $z = z(t)$. Атрибут открыт.

Класс содержит следующие конструкторы:

1) Открытый конструктор, принимающий следующие параметры:

- экземпляр класса `ElementTreeNonlinearEquation`, содержащий функцию расчета координаты по x ;
- экземпляр класса `ElementTreeNonlinearEquation`, содержащий функцию расчета координаты по y ;
- экземпляр класса `ElementTreeNonlinearEquation`, содержащий функцию расчета координаты по z ;

Конструктор создает экземпляр класса и инициализирует соответствующие атрибуты.

2) Открытый конструктор, не принимающий параметров, выполняет создание пустого экземпляра класса.

PiecewiseFunction

Класс, представляющий кусочно-заданной функцию. Кусочно-заданная функция – это функция, которая задана различными формулами на разных интервалах. Записывается в виде:

$$f(x) = \begin{cases} f_0(x), x < x_1 \\ f_1(x), x_1 < x < x_2 \\ \dots \\ f(x), x_n < x \end{cases} \quad (32)$$

Данный класс содержит один открытый атрибут – `function`, в котором содержится список экземпляров класса `ElementPiecewiseFunction`, т.е. атрибут отражает функции на всех интервалах кусочно-заданной функции.

Также класс содержит один открытый конструктор, который не принимает параметров, конструктор проводит первичную инициализацию атрибута `function`.

TrihedronFrene

Класс, представляющий трехгранник Френе (описание трехгранника Френе и алгоритм его получения представлен далее).

Класс содержит следующие атрибуты:

1) `tangentVector`. Экземпляр класса `ParametricRepresentation`, в котором храниться вектор-функция (в параметрическом виде) касательного вектора. Атрибут открыт.

2) `normalVector`. Экземпляр класса `ParametricRepresentation`, в котором храниться вектор-функция (в параметрическом виде) вектора нормали. Атрибут открыт.

3) `binormalVector`. Экземпляр класса `ParametricRepresentation`, в котором храниться вектор-функция (в параметрическом виде) вектора бинормали. Атрибут открыт.

Класс содержит следующие конструкторы:

1) Открытый конструктор, не принимающий параметров, выполняет создание пустого экземпляра класса.

2) Открытый конструктор, принимающий следующие параметры:

- экземпляр класса `ParametricRepresentation`, содержащий вектор-функцию для касательного вектора;
- экземпляр класса `ParametricRepresentation`, содержащий вектор-функцию для вектора нормали;
- экземпляр класса `ParametricRepresentation`, содержащий вектор-функцию для вектора бинормали;

Конструктор создает экземпляр класса и инициализирует соответствующие атрибуты.

Vector3D

Класс, представляющий позицию точки в пространстве.

Класс содержит следующие атрибуты:

- 1) x. Вещественно число, которое хранит координату x. Атрибут открыт.
- 2) y. Вещественно число, которое хранит координату y. Атрибут открыт.
- 3) z. Вещественно число, которое хранит координату z. Атрибут открыт.
- 4) zeroVector. Экземпляр класса Vector3D, в котором храниться нулевой вектор. Атрибут статический и открытый.

Класс содержит следующие методы:

- 1) ToVector3Unity. Открытый метод, используемый для создание экземпляра класса Vector3 (стандартный класс Unity), метод не принимает параметров, возвращает экземпляр класса Vector3, в котором значения координат x, y, z соответствуют значениям в экземпляре класса Vector3D.

- 2) LengthVector. Открытый метод, используемый для получения длинны вектора, заданного двумя точками. Метод принимает один параметр – экземпляр класса Vector3D, в котором содержатся координаты второй точки. Возвращает метод вещественное число, отражающее длину вектора заданного двумя точками в пространстве.

Класс содержит следующие конструкторы:

- 1) Открытый конструктор, не принимающий параметров, выполняет создание пустого экземпляра класса.
- 2) Открытый конструктор, принимающий следующие параметры:
 - вещественное число, хранящее позицию точки по x координате;
 - вещественное число, хранящее позицию точки по y координате;
 - вещественное число, хранящее позицию точки по z координате;

3.3 Проектирование структур данных

В системе имитации полета летательного аппарата необходимо хранить математические формулы и уравнения, для решения этой задачи воспользуемся деревом операций. Дерево операций отражает структуру вычислений в виде

потоков данных от операндов к результатам, терминальными вершинами (листьями) являются исходные операнды, промежуточными вершинами – операции (доступные операции для разрабатываемого дерева хранятся в классе `ElementsEquation`).

Класс `ElementTreeNonlinearEquation` выступает в роли узла дерева, рассмотрим класс подробнее:

1) Атрибут `parent` является указателем на узел родитель, т.е. является указателем на экземпляр класса `ElementTreeNonlinearEquation`.

2) Атрибут `descendant` содержит список указателей на узлы наследники, т.е. на экземпляры класса `ElementTreeNonlinearEquation`.

3) Атрибут `content` является информационной составляющей узла (представляется в виде экземпляра класса `EquationComponent`), в которой может храниться:

- переменная;
- число;
- операция.

Для хранения используется три атрибута (`typeComponent`, `nameVariable`, `valueNumber`), два из которых опциональны, т.е. могут существовать или не существовать в зависимости от значения первого (`typeComponent`).

Если атрибут `typeComponent` имеет значение функции или оператора из класса перечисления `ElementsEquation` (например `addition`, `division`, `sinus`, `power` и т.д.), то значения `nameVariable` и `valueNumber` считается не существующими и значение информационной составляющей определяется только значением данного атрибута.

Если атрибут `typeComponent` имеет значение `number`, то переменная `valueNumber` содержит операнд, который будет использоваться для вычисления значения. Атрибут `nameVariable` считается не существующим.

Если атрибут `typeComponent` имеет значение `variable`, то переменная `nameVariable` содержит имя переменной, которая будет использоваться для вычисления значения. Атрибут `valueNumber` считается не существующим.

Можно было поступить иным способом, спроектировать обобщенный класс, который будет содержать один атрибут `typeComponent`, а потом создать два класса наследника, содержащие отдельно атрибуты `nameVariable` и `valueNumber`. Но такой подход не даст значительных изменений по объему затраченной памяти, но с другой стороны попытка изменения информационной части приведет к необходимости удаления старого экземпляра класса и созданию нового, т.е. выделению памяти под новый экземпляр класса, данные операции являются весьма трудоемки и занимают больше времени, чем простое присвоение нового значений. Поэтому при выборе между скоростью и объемом затраченной памяти выбирается скорость.

Атрибут `descendant` класса `ElementTreeNonlinearEquation` был преднамеренно представлен в виде списка указателей, а не двух отдельных указателей на левое и правое поддерево, т.к. в дереве в перспективе могут храниться не только унарные/бинарные операции и функции, но и тернарные (на данный момент подобные операции и функции не используются). Если возникнет необходимость в хранении подобных операций/функций, то изменения будут достаточно незначительны, следовательно, дерево получится достаточно гибким для модернизации. На скорость обработки и объем занимаемой памяти подобное решение не повлияет кардинальным образом, т.к. список подлежит индексированию, следовательно, доступ к элементам списка не будет занимать много времени, а для хранения узла понадобится только на четыре байта больше (ссылка на список).

Атрибут `parent`, хранящий указатель на узел родитель позволит быстрее обрабатывать и изменять выражения, хранящиеся в дереве, он занимает четыре байта, но так же при выборе между скоростью обработки и объемом занимаемой памяти предпочтение отдается скорости.

Для взаимодействия с деревом операций используются методы:

- 1) `CreateDescendant`. Метод инициализирует атрибут `descendant`.

2) AddDescendant. Метод добавляет нового наследника узлу, т.е. присваивает новому узлу родителя, а в узел родитель добавляет указатель на новый узел.

На рисунке 3.3.1 представлена схема хранения тестового выражения в разработанном дереве операций.

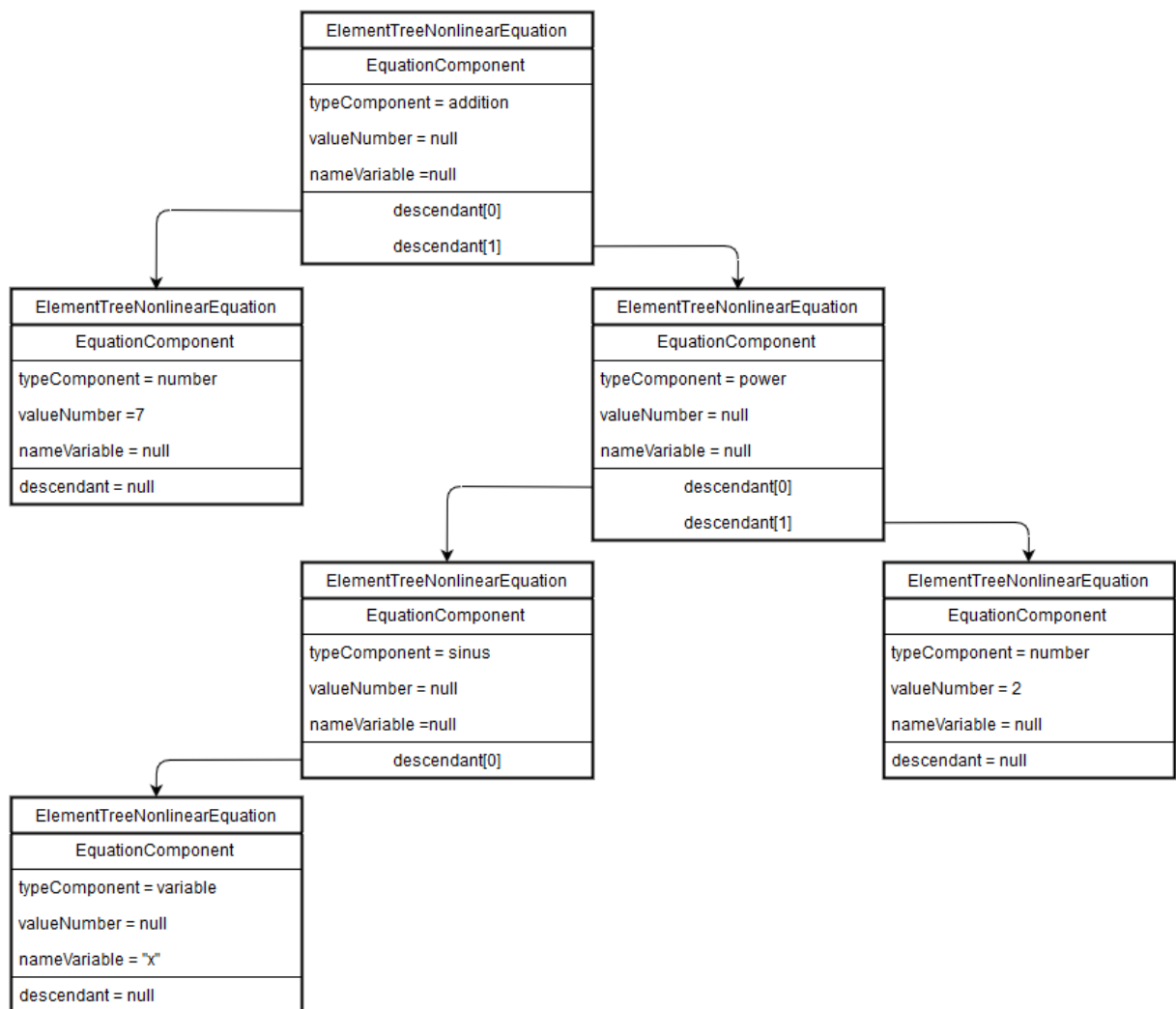


Рисунок 3.3.1 – Пример хранения выражения в дереве операций

3.4 Проектирование алгоритмов

3.4.1 Алгоритм вычисления коэффициентов кубического сплайна

Слово сплайн (англ. spline) означает гибкую линейку, применяемую для проведения гладких кривых через заданные точки на плоскости, форма такого универсального лекала на каждом отрезке описывается кубической параболой.

Сплаины широко применяются в инженерных приложениях, в частности, в компьютерной графике.

Интерполяция – операция приближения функции, заданной в отдельных точках внутри некоторого заданного промежутка. Простейшая задача интерполяции заключается в следующем, на отрезке $[x_0, x_n]$ заданы n точек x_k ($k = 0, 1, 2, \dots, n$), называемые узлами интерполяции, и значения некоторой функции $f(x)$ в этих точках $f(x_0) = y_0, f(x_1) = y_1, \dots, f(x_n) = y_n$. Требуется построить интерполирующую функцию $F(x)$, принимающую в узлах интерполяции те же значения, что и $f(x)$, т.е. $F(x_0) = y_0, F(x_1) = y_1, \dots, F(x_n) = y_n$. Геометрически это означает, что требуется найти некоторую кривую $y = F(x)$ определенного типа, проходящую через заданный набор точек $(x_k, y_k), k = 0, 1, 2, \dots, n$.

Пусть задана табличная функция:

$$(x_k, f(x_k)) \quad k = 1, 2, \dots, n, \quad (33)$$

где x_k – упорядоченные по возрастанию абсциссы узлов интерполирования.

Задача состоит в нахождении коэффициентов кубического сплайна. Обозначим его $s(x)$; сплайн образован $(n - 1)$ кубическими функциями

$$P_k(x_k) = a_k + b_k(x - x_k) + c_k(x - x_k)^2 + d_k(x - x_k)^3, \quad (34)$$

которые определены на отрезке $[x_k, x_{k+1}]$ и которые должны удовлетворять нижеприведённым условиям в узлах интерполирования, состоящими в прохождении графика сплайна через заданные точки и наличие у сплайна определенной гладкости.

Условие прохождения через заданные точки

$$P_k(x_k) = f(x_k), P_k(x_{k+1}) = f(x_{k+1}), k = 1, 2, 3, \dots, n - 1 \quad (35)$$

Условие непрерывности первой производной во внутренних точках

$$\frac{dP_k}{dx}(x_k) = \frac{dP_{k+1}}{dx}(x_k), k = 2, 3, \dots, n - 1 \quad (36)$$

Условие непрерывности второй производной во внутренних точках

$$\frac{d^2P_k}{dx^2}(x_k) = \frac{d^2P_{k+1}}{dx^2}(x_k), k = 2, 3, \dots, n - 1 \quad (37)$$

Кроме этого, наложим на сплайн следующие граничные условия (для однозначного определения его коэффициентов):

На левой границе отрезка $s(x_1) = 0$, т.е.

$$\frac{d^2 P_1}{dx^2}(x_1) = 0 \quad (38)$$

На правой границе отрезка $s(x_n) = 0$, т.е.

$$\frac{d^2 P_{n-1}}{dx^2}(x_{n-1}) = 0 \quad (39)$$

Данные выкладки можно обобщить для сплайна любой степени. С ростом степени будут добавляться условия для производных (для сплайна n степени – n условий, $(n - 1)$ -я производная).

Значения первых производных функции $f(x)$ в узлах интерполирования заранее неизвестны, поэтому выберем коэффициенты b_k таким образом, чтобы обеспечить непрерывность второй производной кубического сплайна во всех внутренних узлах на отрезке интерполирования. Итак, для $k = 1, 2, 3, \dots, n - 1$:

$$a_k = f(x_k) \quad (40)$$

$$c_k = \frac{3f[x_k, x_{k+1}] - b_{k+1} - 2b_k}{h_k} \quad (41)$$

$$d_k = \frac{b_k + b_{k+1} - 2f[x_k, x_{k+1}]}{h_k^2} \quad (42)$$

где

$$f[x_k, x_{k+1}] = \frac{f(x_{k+1}) - f(x_k)}{x_{k+1} - x_k} \quad (43)$$

$$h_k = x_{k+1} - x_k \quad (44)$$

является разделенной разностью первого порядка интерполируемой функции $f(x)$, которая построена по точкам x_k, x_{k+1} .

Из условия непрерывности второй производной сплайна во внутренних узлах интерполирования

$$\frac{d^2 P_k}{dx^2}(x_{k+1}) = \frac{d^2 P_{k+1}}{dx^2}(x_{k+1}), k = 2, 3, \dots, n - 2 \quad (45)$$

получаем, что

$$c_k + 3d_k h_k = c_{k+1} \quad (46)$$

Подставив в полученные уравнения приведенные выражения для коэффициентов c_k и d_k полиномов $P_k(x)$ через неизвестные пока коэффициенты b_k . Тогда получим систему линейных алгебраических уравнений относительно коэффициента b_k , т.е. относительно тангенсов углов наклона кубического сплайна во всех (как внутренних, так и граничных) узлах отрезка интерполирования.

$$\begin{aligned} & \frac{3f[x_k, x_{k+1}] - b_{k+1} - 2b_k}{h_k} + 3h_k \frac{b_k + b_{k+1} - 2f[x_k, x_{k+1}]}{h_k^2} \\ &= \frac{3f[x_{k+1}, x_{k+2}] - b_{k+2} - 2b_{k+1}}{h_{k+1}}, \\ & k = 1, 2, \dots, n-2 \end{aligned} \quad (47)$$

Преобразуем систему уравнений, вычислив коэффициенты при b_k, b_{k+1} и b_{k+2} , получим тогда

$$\alpha_k b_k + \beta_k b_{k+1} + \gamma_k b_{k+2} = \delta_k, \quad (48)$$

где введены следующие обозначения

$$\alpha_k = \frac{1}{h_k} \quad (49)$$

$$\beta_k = 2 \left[\frac{1}{h_k} + \frac{1}{h_{k+1}} \right] \quad (50)$$

$$\gamma_k = \frac{1}{h_{k+1}} \quad (51)$$

$$\delta_k = 3 \left[\frac{f[x_{k+1}, x_{k+2}]}{h_{k+1}} + \frac{f[x_k, x_{k+1}]}{h_k} \right] \quad (52)$$

Заметим, что поскольку разыскиваем сплайн с нулевым наклоном в граничных точках, то $b_1 = 0$ и $b_n = 0$, но данные коэффициенты входят в первое последнее уравнение системы. Поэтому система из $(n-2)$ уравнений относительно неизвестных коэффициентов b_k для $k = 2, 3, \dots, n-1$ выглядит следующим образом:

$$\begin{bmatrix} \beta_1 & \gamma_1 & 0 & 0 & 0 & 0 \\ \alpha_2 & \beta_2 & \gamma_2 & 0 & 0 & 0 \\ 0 & \alpha_3 & \beta_3 & \gamma_3 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \alpha_{n-3} & \beta_{n-3} & \gamma_{n-3} \\ 0 & 0 & 0 & 0 & \alpha_{n-2} & \beta_{n-2} \end{bmatrix} \begin{bmatrix} b_2 \\ b_3 \\ b_4 \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{bmatrix} = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \vdots \\ \delta_{n-3} \\ \delta_{n-2} \end{bmatrix} \quad (53)$$

Матрица системы линейных уравнений является трехдиагональной, все остальные ее коэффициенты, за исключением коэффициентов, стоящих на главной и побочных диагоналях, равны нулю. Решение системы с трехдиагональной матрицей осуществляется за число операций, пропорциональное числу неизвестных. Решив эту систему мы находим коэффициенты b_2, b_3, \dots, b_{n-1} и подставляя их, а также $b_1 = b_n = 0$ в выражение для c_k и d_k при $k = 1, 2, \dots, n - 1$ находим все коэффициенты полиномов $P_k(x)$ составляющих кубический сплайн $s(x)$.

Блок-схема алгоритма вычисления коэффициентов кубического сплайна представлена на рисунке 3.4.1.1.



Рисунок 3.4.1.1 – Схема алгоритма вычисления коэффициентов кубического сплайна

Этапы вычисления коэффициентов кубического сплайна:

- 1) Рассчитать значения $f[x_k, x_{k+1}]$ и h_k для всех отрезков. Выполняется по формулам 43 и 44 соответственно.
- 2) Рассчитать коэффициенты c_k и d_k для всех кубических функций. Выполняется по формулам 41 и 42 соответственно.
- 3) Рассчитать $\alpha_k, \beta_k, \gamma_k$, и δ_k для всех отрезков. Выполняется по формулам 49 – 52 соответственно.
- 4) На основе полученных $\alpha_k, \beta_k, \gamma_k$, и δ_k сформировать смешанную матрицу. Формируем матрицу, представленную формулой 53.
- 5) Решить систему линейных алгебраических уравнений методом Гауса и получить коэффициенты b_k . Алгоритм решения систем линейных алгебраических уравнений рассматривается далее. Решением данной системы будут недостающие коэффициенты $b_2 - b_{n-1}$.
- 6) На основе полученных коэффициентов a_k, b_k, c_k , и d_k сформировать кубические функции. Кубические функции формируются по формуле 34 в виде деревьев операций.

3.4.2 Алгоритм определения орт естественного трехгранника

Репер или трехгранник Френе известный также, как естественный, сопровождающий сопутствующий – ортонормированный репер в трехмерном пространстве, возникающий при изучении бирегулярных кривых, т.е. кривых, у которых первая и вторая производная линейно независимы в любой точке.

Рассмотрим точку M в определенный момент времени (рисунок 3.4.2.1), предполагаем, что нам известна траектория ее движения. Проведя через точку M три прямых:

- касательную к траектории;
- главную нормаль;
- бинормаль.

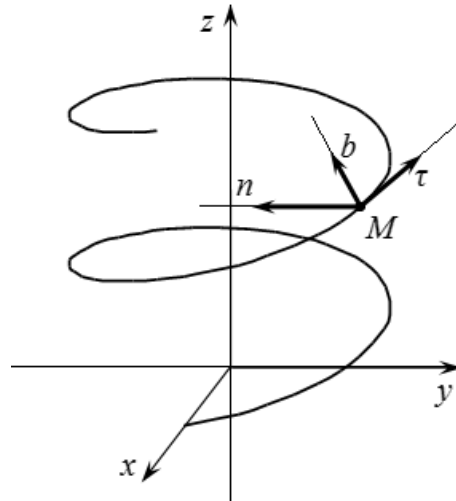


Рисунок 3.4.2.1 – Траектория движения точки М

Главная нормаль перпендикулярна касательной и направлена в сторону мгновенного центра кривизны траектории, бинормаль перпендикулярна касательной и главной нормали. Выберем систему координат с началом в точке М и осями, направленными вдоль этих прямых. Такую систему координат называют естественным трехгранником Френе, а оси такой системы координат называют осями естественного трехгранника.

Пусть $\vec{\tau}$, \vec{n} и \vec{b} – единичные векторы, направленные вдоль касательной, главной нормали и бинормали к траектории, соответственно. Данные векторы являются ортами выбранной системы координат или ортами естественного трехгранника.

Вектор $\vec{\tau}$ направлен вдоль касательной к траектории, поэтому можно выбрать два взаимно противоположных направления. Наиболее удобный способ это направить $\vec{\tau}$ вдоль вектора скорости \vec{v} точки, тогда

$$\vec{\tau} = \frac{\vec{v}}{v} \quad (54)$$

Однако это не всегда можно сделать, встречаются случаи, когда траектория движения заранее известна, а скорость нет.

Направление единичного вектора \vec{n} главной нормали определено однозначно, он направлен перпендикулярно $\vec{\tau}$, в сторону мгновенного центра кривизны траектории.

Вектор бинормали \vec{b} направлен перпендикулярно векторам $\vec{\tau}$ и \vec{n} так, чтобы три вектора $\vec{\tau}$, \vec{n} и \vec{b} образовали правую систему координат:

$$\vec{b} = [\vec{\tau} \times \vec{n}] \quad (55)$$

Считаем, что вектор $\vec{\tau}$ совпадает по направлению с вектором скорости \vec{v} точки, тогда применим следующие формулы (формулы приведены без вывода):

$$\vec{v} = v\vec{\tau} \quad (56)$$

$$\vec{a} = a_{\tau}\vec{\tau} + a_n\vec{n} \quad (57)$$

То есть, в естественном трехграннике с оортами $(\vec{\tau}, \vec{n}, \vec{b})$, скорость имеет одну компоненту:

$$\vec{v} = (v, 0, 0), \quad (58)$$

равную модулю скорости, иными словами, модуль скорости v – это проекция вектора скорости \vec{v} на ось $\vec{\tau}$ естественного трехгранника. Проекция вектора скорости \vec{v} на оси \vec{n} и \vec{b} трехгранника равны нулю.

Ускорение имеет две компоненты:

$$\vec{a} = (a_{\tau}, a_n, 0) \quad (59)$$

Эти компонентами являются касательное и нормальное ускорения:

$$a_{\tau} = \frac{dv}{dt}, \quad (60)$$

$$a_n = \frac{v^2}{\rho} \quad (61)$$

То есть касательное a_{τ} и нормальное a_n ускорения – это проекции вектора ускорения \vec{a} на оси $\vec{\tau}$ и \vec{n} естественного трехгранника. Проекция вектора ускорения \vec{a} на ось \vec{b} равна нулю.

Далее считаем, что есть неподвижная система координат $Oxyz$, материальная точка совершает движение, требуется найти оси естественного трехгранника, т.е. определить проекции орт $\vec{\tau}$, \vec{n} и \vec{b} в системе координат $Oxyz$.

Чтобы определить орты естественного трехгранника, нужно найти компоненты векторов скорости \vec{v} и нормального ускорения \vec{a}_n , применяя следующие формулы:

$$\vec{v} = \dot{\vec{r}} \quad (62)$$

$$\vec{a} = \dot{\vec{v}} \quad (63)$$

$$\vec{a}_\tau = (\vec{a} \cdot \vec{\tau})\vec{\tau} \quad (64)$$

$$\vec{a}_n = \vec{a} - \vec{a}_\tau \quad (65)$$

Далее определяем орты естественного трехгранника:

$$\vec{\tau} = \frac{\vec{v}}{v} \quad (66)$$

$$\vec{n} = \frac{\vec{a}_n}{a_n} \quad (67)$$

$$\vec{b} = [\vec{\tau} \times \vec{n}] \quad (68)$$

При естественном способе задания движения точки нам известна траектория ее движения, поэтому стоит задача по известной траектории, определить орты естественного трехгранника. Если траектория представляет собой простую геометрическую фигуру, то определить векторы $\vec{\tau}$, \vec{n} и \vec{b} можно геометрически.

В общем, и более сложном случае, нужно представить уравнение траектории в параметрическом виде, для этого вводим параметр t . Это можно сделать различными способами, поэтому необходимо выбрать наиболее удобное представление.

Пусть, например, траекторией движения является эллипс, лежащий в плоскости x, y :

$$\begin{cases} \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \\ z = 0 \end{cases} \quad (69)$$

Наиболее удобно параметрическое представление можно получить, если воспользуемся тригонометрической формулой:

$$\cos^2 t + \sin^2 t = 1 \quad (70)$$

Тогда уравнение траектории имеет вид:

$$\begin{cases} x = a \cos t \\ y = b \sin t \\ z = 0 \end{cases} \quad (71)$$

здесь t – параметр.

Это не единственный способ получить параметрическое представление, можно разрешить уравнение эллипса относительно y :

$$y = \pm b \sqrt{1 - \frac{x^2}{a^2}} \quad (72)$$

Применяя эту формулу, получим другое параметрическое представление:

$$\begin{cases} x = t \\ y = \pm b \sqrt{1 - \frac{t^2}{a^2}} \\ z = 0 \end{cases} \quad (73)$$

Далее считаем, что эти параметрические уравнения описывают движение материальной точки, в котором параметр t играет роль времени, тогда, для определения осей трехгранника, можно применить формулы, применяемые для векторного и координатного способов задания движения. Вычисленные, таким образом, скорость и ускорение будут зависеть от выбранного параметрического представления, но геометрические характеристики траектории, такие как орты $\vec{\tau}, \vec{n}, \vec{b}$ и радиус кривизны траектории не зависят от выбранного параметрического представления.

Подводя итоги, чтобы найти орты естественного трехгранника по заданной траектории движения, необходимо представить уравнение траектории в параметрическом виде и применить формулы, применяемые при координатном способе задания движения.

Блок-схема алгоритма определения орт естественного трехгранника представлена на рисунке 3.4.2.2.



Рисунок 3.4.2.2 – Схема алгоритма определения орт естественного трехгранника

Этапы определения орт естественного трехгранника:

1) Дифференцируем исходное уравнение по t и получаем компоненты вектора скорости. Применяя формулу 62 получаем $v_x = \dot{x}, v_y = \dot{y}, v_z = \dot{z}$.

2) Находим квадрат скорости и модуль скорости. Соответственно получаем $v^2 = (\vec{v} \cdot \vec{v}) = v_x^2 + v_y^2 + v_z^2$ и $v = \sqrt{v^2}$.

3) Рассчитываем единичный вектор в направлении касательной к траектории. Применяя формулу 66, получаем касательный вектор $\vec{\tau}$.

4) Дифференцируем компоненты вектора скорости по t и получаем компоненты вектора ускорения. Применяя формулу 63 получаем $a_x = \dot{v}_x, a_y = \dot{v}_y, a_z = \dot{v}_z$.

5) Получаем касательное ускорение. Применяя формулу 60, получаем касательное ускорение.

6) Получаем вектор касательного ускорения и вектор нормального ускорения. Применяя формулы 64 и 65, получаем соответственно вектор касательного ускорения \vec{a}_τ и вектор нормального ускорения \vec{a}_n .

7) Рассчитываем квадрат и модуль вектора нормального ускорения. Соответственно получаем $a_n^2 = (\vec{a}_n \cdot \vec{a}_n) = a_{nx}^2 + a_{ny}^2 + a_{nz}^2$ и $a_n = \sqrt{a_n^2}$.

8) Получаем единичный вектор в направлении главной нормали траектории. Применяя формулу 67, получаем вектор главной нормали \vec{n} .

9) Получаем единичный вектор в направлении бинормали траектории. Применяя формулу 68, получаем вектор главной нормали \vec{b} .

Все вышеописанные действия проводятся над деревьями операций, по итогу выполнения алгоритма имеем три параметрические функции от t для расчета соответствующих векторов трехгранника.

3.4.3 Алгоритм решения систем линейных уравнений методом Гаусса

Одним из наиболее универсальных и эффективных методов решения линейных алгебраических систем является метод Гаусса, состоящий в последовательном исключении неизвестных.

Пусть дана система уравнений

[illegible]

Процесс решения по методу Гаусса состоит из двух этапов, на первом этапе (прямой ход) система приводится к ступенчатому (в частности, треугольному) виду.

Приведенная система имеет ступенчатый вид

[illegible]

где $k \leq n, a_{ij} \neq 0, i = \overline{1, k}$.

Коэффициенты a_{ij} называются главными элементами системы.

На втором этапе (обратный ход) идет последовательное определение неизвестных из этой ступенчатой системы.

Опишем метод Гаусса подробнее.

Прямой ход. Будем считать, что элемент $a_{11} \neq 0$ (если $a_{11} = 0$, то первым в системе запишем уравнение, в котором коэффициент при x_1 отличен от нуля).

Преобразуем исходную систему, исключив неизвестное x_1 во всех уравнениях, кроме первого (используя элементарные преобразования системы). Для этого умножим обе части первого уравнения на $-\frac{a_{21}}{a_{11}}$ и сложим почленно со вторым уравнением системы. Затем умножим обе части первого уравнения на $-\frac{a_{31}}{a_{11}}$ и сложим с третьим уравнением системы. Продолжая этот процесс, получим эквивалентную систему

[illegible]

Здесь $a_{ij}^{(1)}, b_i^{(1)}$ ($i, j = \overline{2, m}$) – новые значения коэффициентов и правых частей, которые получаются после первого шага.

Аналогичным образом, считая главным элементом $a_{22}^{(1)} \neq 0$, исключаем неизвестное x_2 из всех уравнений системы, кроме первого и второго, и так далее. Продолжаем этот процесс, пока это возможно.

Если в процессе приведения исходной системы к ступенчатому виду появляются нулевые уравнения, т.е. равенства вида $0 = 0$, их отбрасывают. Если же появится уравнение вида $0 = b_i$, а $b_i \neq 0$, то это свидетельствует о несовместимости системы.

Второй этап (обратный ход) заключается в решении ступенчатой системы. Ступенчатая система уравнений, вообще говоря, имеет бесчисленное множество решений. В последнем уравнении этой системы выражаем первое неизвестное x_k через остальные неизвестные (x_{k+1}, \dots, x_n) . Затем подставляя значение x_k в предпоследнее уравнение системы и выражаем x_{k-1} через (x_{k+1}, \dots, x_n) ; затем находим x_{k-2}, \dots, x_1 . Придавая свободным неизвестным (x_{k+1}, \dots, x_n) произвольные значения, получим бесчисленное множество решений.

Если ступенчатая система оказывается треугольной, т.е. $k = n$, то исходная система имеет единственное решение. Из последнего уравнения находим x_n , из предпоследнего уравнения x_{n-1} , далее поднимаясь по системе вверх, найдем все остальные неизвестные (x_{n-2}, \dots, x_1) .

На практике удобнее работать не с системой, а с расширенной ее матрицей, выполняя все элементарные преобразования над ее строками. Удобно, чтобы коэффициент a_{11} был равен единице (уравнения переставить местами, либо разделить обе части уравнения на $a_{11} \neq 1$).

На рисунке 3.4.3.1 представлена реализация метода Гаусса на псевдокоде, где расширенная Матрица – это двумерный массив, содержащий расширенную матрицу, а результат – это массив, в котором после завершения работы алгоритма будет храниться решение системы уравнений. Вначале расширенная матрица

приводится к треугольной (ступенчатой) форме (прямой ход), а потом выполняется обратная подстановка (обратный ход).

```

1  for i = 0 to расширеннаяМатрица.колСтрок
2  begin
3      for j = i to расширеннаяМатрица.колСтрок
4      begin
5          if расширеннаяМатрица[j][i] == 0
6          begin
7              следИтерерация;
8          end
9
10         коэффициент = расширеннаяМатрица[j][i];
11
12         for k = i to расширеннаяМатрица.колСтолбцов
13         begin
14             расширеннаяМатрица[j][k] = расширеннаяМатрица[j][k] / коэффициент;
15         end
16     end
17
18
19     for j = i + 1 to расширеннаяМатрица.колСтрок
20     begin
21         if расширеннаяМатрица[j][i] == 0
22         begin
23             следИтерерация;
24         end
25
26         for k = i to расширеннаяМатрица.колСтолбцов
27         begin
28             расширеннаяМатрица[j][k] = расширеннаяМатрица[j][k] - расширеннаяМатрица[i][k];
29         end
30     end
31 end
32 end
33
34 for i = результат.количество to 0
35 begin
36     вспомогательная = расширеннаяМатрица[i][расширеннаяМатрица.колСтолбцов];
37
38     for j = i + 1 to расширеннаяМатрица.колСтолбцов - 1
39     begin
40         вспомогательная = вспомогательная - расширеннаяМатрица[i][j] * результат[j];
41     end
42
43     результат[i] = вспомогательная;
44 end

```

Рисунок 3.4.3.1 – Реализация метода Гаусса на псевдокоде

3.5 Проектирование пользовательского интерфейса

На основе поведенческих спецификаций подсистемы имитации движения летательного аппарата, необходимо разработать макеты графического пользовательского интерфейса для сцены создания полетного задания и сцены имитации движения летательного аппарата.

Wireframes и mockups макеты графического пользовательского интерфейса сцены создания полетного задания представлены на рисунках 3.5.1 и 3.5.2 соответственно.

На данных макетах находится:

- панель, содержащая полетное задание;
- панель свойств целевой точки;
- предварительный маршрут движения;
- кнопка «FLIGHT», клик по которой приведет к генерации полетной информации и переходу на сцену имитации движения летательного аппарата.

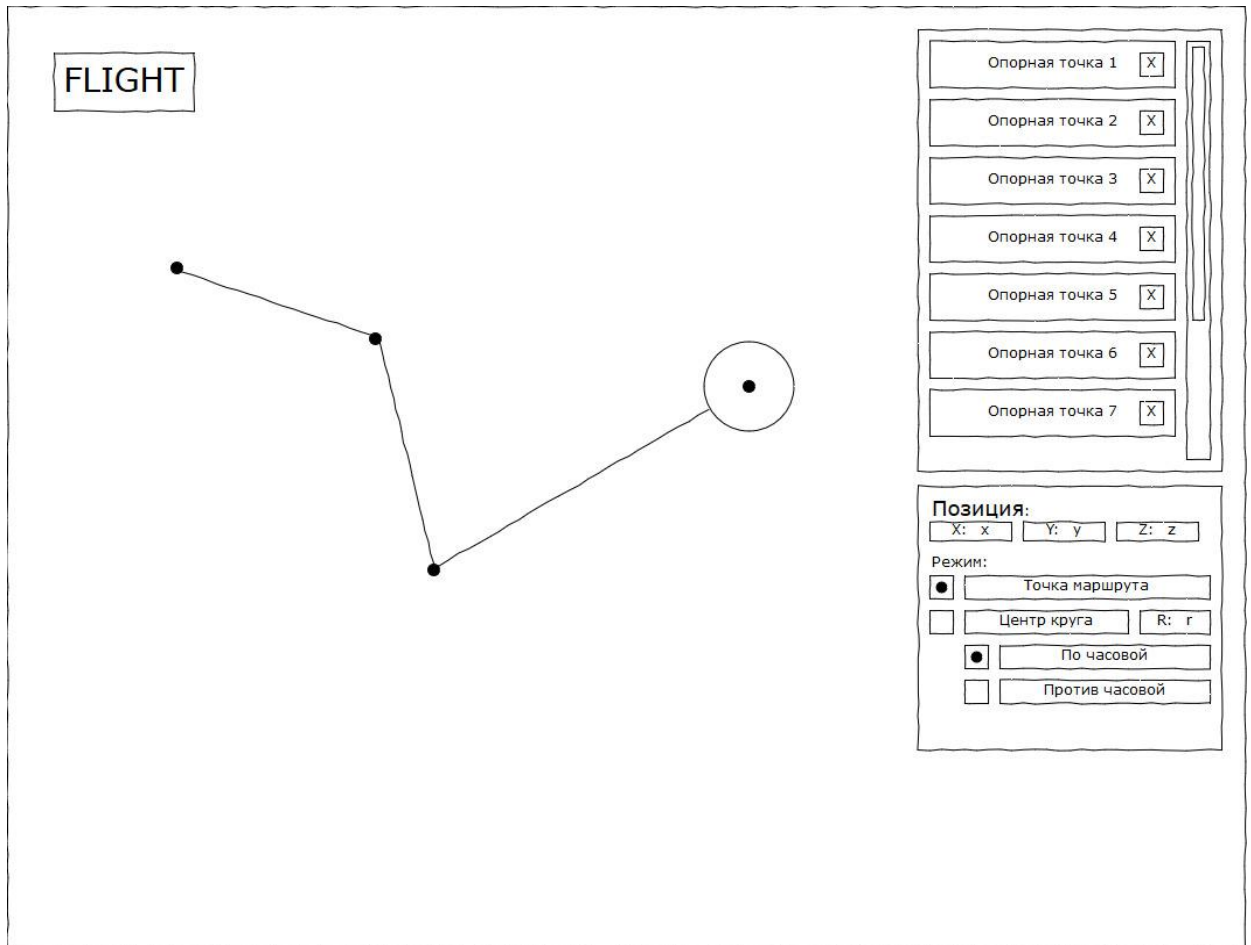


Рисунок 3.5.1 – Wireframes макет графического пользовательского интерфейса сцены создания полетного задания

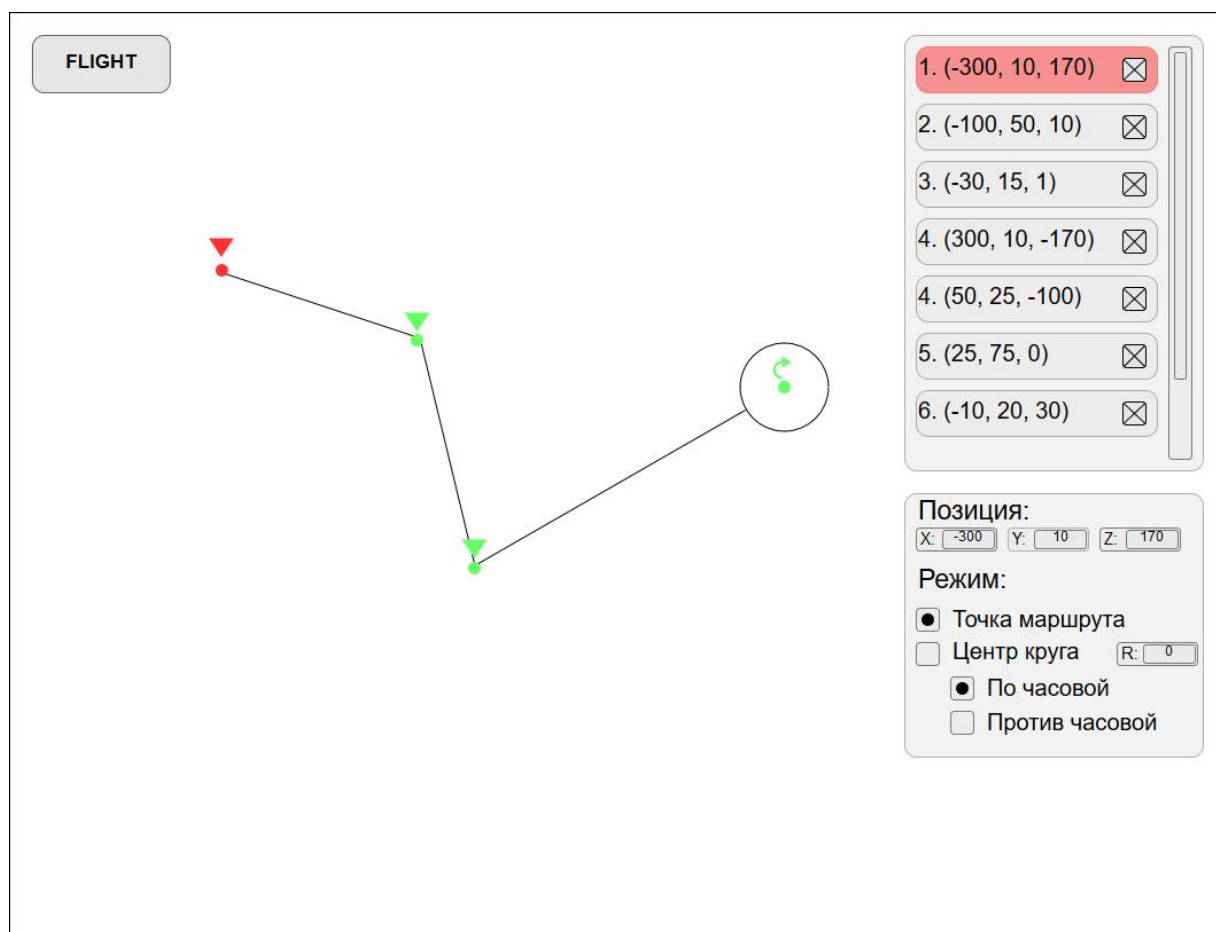


Рисунок 3.5.2 – Mockups макет графического пользовательского интерфейса сцены создания полетного задания

Wireframes и mockups макеты графического пользовательского интерфейса сцены имитации пролета летательного аппарата, представлены на рисунках 3.5.3 и 3.5.4 соответственно.

На данных макетах находится:

- scroll bar для изменения скорости имитации;
- check box включающий/выключающий отображение полного маршрута движения летательного аппарата;
- кнопка «Новое задание», клик по которой приведет к созданию нового полетного задания;
- кнопка «Выход», клик по которой приведет к закрытию приложения;
- координаты летательного аппарата в данный момент времени;
- проекции скоростей на генеральные оси координат;

- время имитации;
- характеристики поворота летательного аппарата (курс, тангаж, крен);
- летательный аппарат с трех точек наблюдения;
- летательный аппарат.

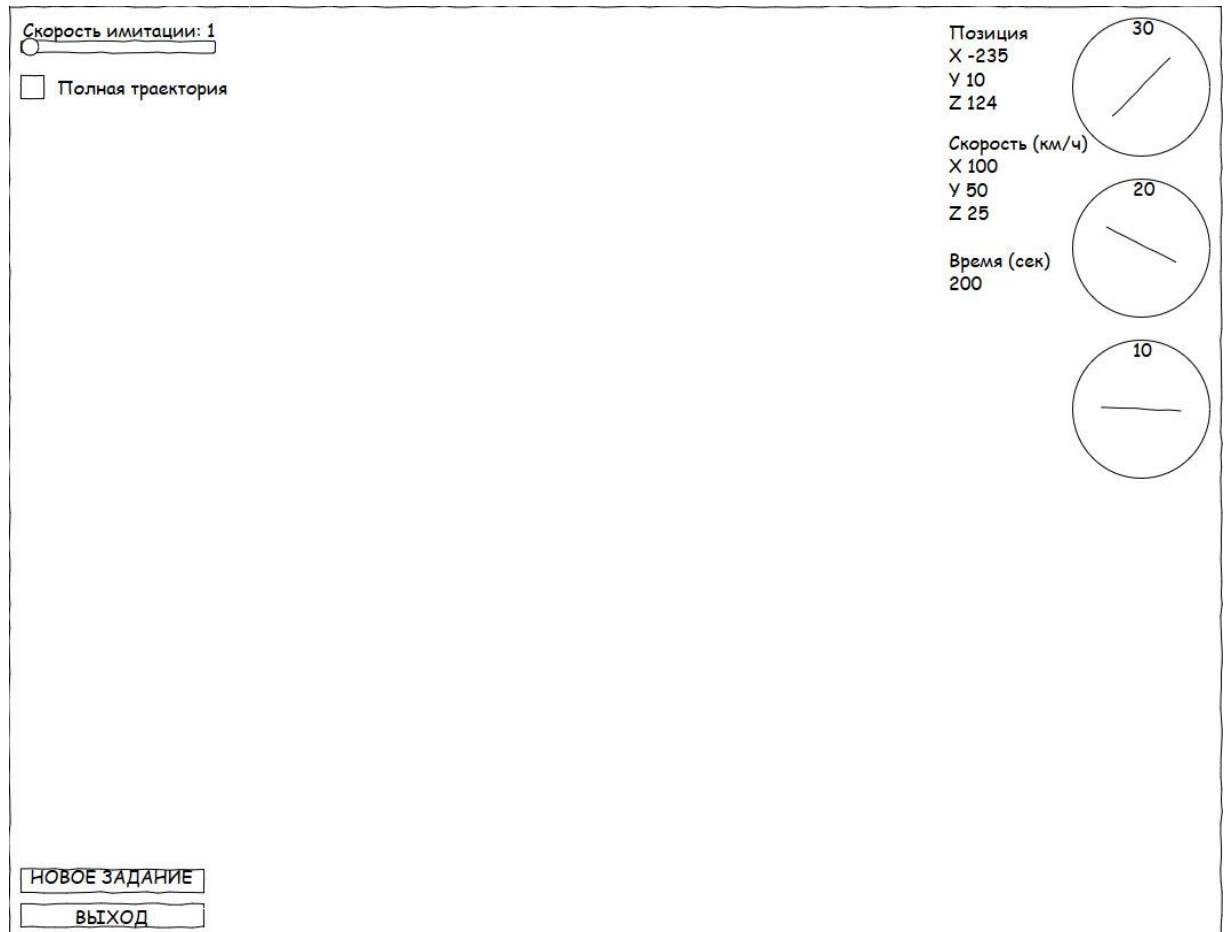


Рисунок 3.5.3 – Wireframes макет графического пользовательского интерфейса сцены имитации полета

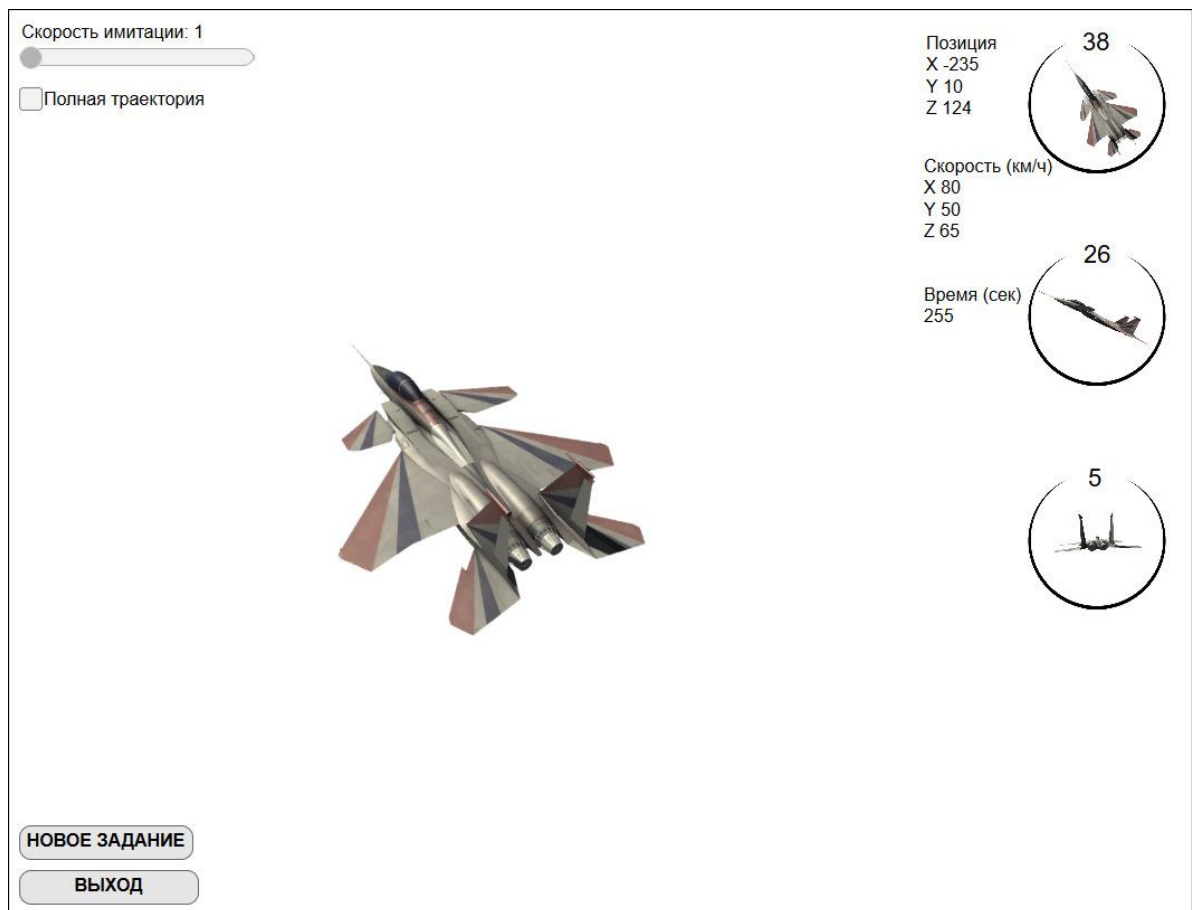


Рисунок 3.5.4 – Moskurs макет графического пользовательского интерфейса сцены имитации пролета

4 РЕАЛИЗАЦИЯ, ТЕСТИРОВАНИЕ И ОТЛАДКА ПОДСИСТЕМЫ ИМИТАЦИИ ДВИЖЕНИЯ ЛЕТАТЕЛЬНОГО АППАРАТА

4.1 Реализация компонентов подсистемы

Рассмотренные ранее диаграммы отражали концептуальные и логические аспекты построения системы имитации движения летательного аппарата для программного имитатора закабинного пространства. Особенность логического представления заключается в том, что оно оперирует понятиями не имеющими материального воплощения, иными словами, элементы логического представления (классы, ассоциации, состояния и т.д.) не существуют материально или физически, они лишь демонстрируют статическую структуру или динамические аспекты поведения системы.

При создании конкретной физической системы необходимо реализовать все элементы логического представления в конкретные материальные сущности, описание таких сущностей осуществляется с помощью физического представления модели.

На рисунке 4.1.1 демонстрируется диаграмма компонентов для разрабатываемой системы имитации движения летательного аппарата.

Каждый отдельный класс помещается в отдельный файл с именем класса, следовательно, диаграмма компонентов напоминает физическую диаграмму классов.

Все классы из пространства имен MathCore формируют библиотеку математического ядра, с которой взаимодействует графический пользовательский интерфейс, т.е. формируется последовательность целевых точек, она передается математическому ядру, в свою очередь ядро возвращает список, содержащий полетную информацию.

Таким образом, ядро не зависит от интерфейса и может быть повторно использовано в другом проекте.

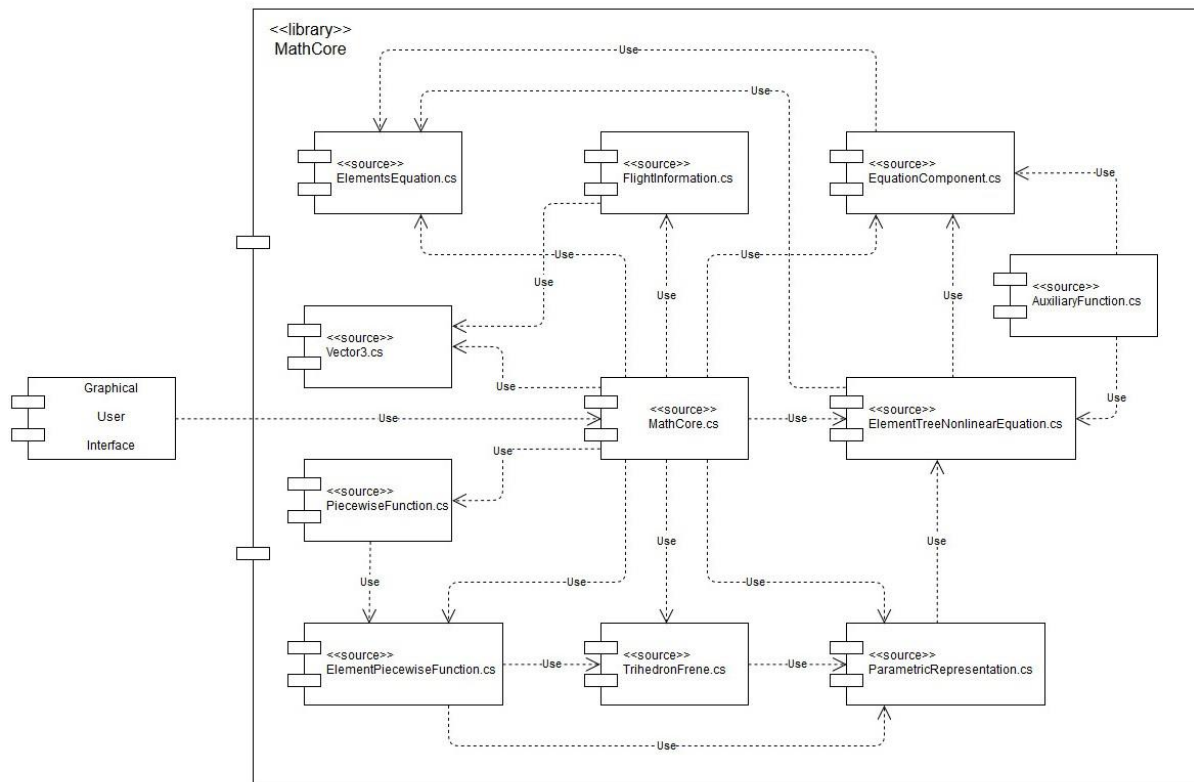


Рисунок 4.1.1 – Диаграмма компонентов

4.2 Реализация пользовательского интерфейса

Основываясь на функциональной модели, на вход подсистемы имитации движения летательного аппарата поступает полетное задание, на выходе имеем полетную информацию. Для генерации полетного задания необходим некий пользовательский интерфейс, с помощью которого будет сформирована последовательность опорных точек в пространстве. С другой стороны, для демонстрации полученной полетной информации требуется визуализировать пролет летательного аппарата, т.к. результатом расчета траектории движения является последовательность характеристик летательного аппарата в пространстве, иначе говоря, список чисел, основываясь на которых достаточно сложно оценить корректность полученной траектории. Для решения вышеописанных задач необходим инструмент с возможностями разработки двумерных и трехмерных приложений.

Unity – кроссплатформенная среда разработки двух- и трехмерных приложений, поддерживает два сценарных языка: C# и JavaScript (модифицированный). Проект в Unity подразделяется на сцены (отдельные файлы), содержащие отдельные наборы сценариев, объектов, и настроек.

С помощью редактора Unity и основываясь на разработанном макете, реализуем графический пользовательский интерфейс для формирования полетного задания. На рисунке 4.2.1 представлен стартовый вид приложения после запуска, соответственно нажав кнопку «Старт» осуществляется переход на сцену формирования полетного задания, а нажав «Выход» приложение закрывается и освобождает занимаемые ресурсы.

После перехода на сцену формирования полетного задания необходимо расставить опорные точки (рисунок 4.2.2).

На рисунке даны следующие обозначения:

1) Предварительный маршрут движения. Демонстрирует последовательность опорных точек, подсвечивает красным цветом точку активную для изменений в панели свойств.

2) Полетное задание. Демонстрирует опорные точки из полетного задания, точки находятся в той последовательности, в которой летательному аппарату необходимо пройти маршрут. Цветом выделяется точка активная для изменения в панели свойств.

3) Панель свойств. Применяется для изменения свойств конкретной опорной точки.

4) Кнопка FLIGHT. После создания необходимого маршрута кликнув по данной кнопке программа рассчитает маршрут движения летательного аппарата и загрузит сцену демонстрации полученного маршрута.

5) Кнопка удаления опорной точки. Удаляет конкретную опорную точку из полетного задания и исправляет предварительный маршрут движения.

Для установки опорной позиции потребуется кликнуть левой кнопкой мыши по желаемой позиции на сцене, после этого на сцене отобразится новая

точка и панель установки высоты (рисунок 4.2.3), высота устанавливается с помощью ScrollBar в диапазоне от 0 до 100 метров.

Кликнув по одной из точки в панели полетного задания отобразится панель свойств для выбранной точки, с помощью которой можно изменить:

- координату точки по X;
- координату точки по Z;
- высоту, на которой располагается точка;
- режим (модификацию) точки.

Для опорной точки доступно две модификации:

1) Точка маршрута. При использовании этой модификации летательный аппарат пролетит через данную точку.

2) Центр круга. При использовании этой модификации пользователю необходимо указать радиус круга, центр которого находится в конкретной опорной точке, и направление движения (по часовой или против часовой стрелки), в таком случае летательный аппарат пролетит по контуру круга в заданном направлении с заданным радиусом (рисунок 4.2.4).

После создания желаемого маршрута движения необходимо кликнуть по кнопке FLIGHT, результатом нажатия станет загрузка сцены демонстрации пролета летательного аппарата по заданному маршруту (рисунок 4.2.5).

На рисунке даны следующие обозначения:

- 1) Летательный аппарат.
- 2) Время, прошедшее с начала имитации пролета.
- 3) Проекция скоростей на генеральные оси координат.
- 4) Позиция летательного аппарата.
- 5) Летательный аппарат вид с тыла (демонстрирует угол крена).
- 6) Угол крена в градусах.
- 7) Летательный аппарат вид сбоку (демонстрирует угол тангажа).
- 8) Угол тангажа в градусах.
- 9) Летательный аппарат вид сверху (демонстрирует угол курса).
- 10) Угол курса в градусах.

- 11) Текущая скорость имитации.
- 12) ScrollBar для изменения скорости имитации в диапазоне от 1 до 100.
- 13) CheckBox для включения/выключения демонстрации полного маршрута летательного аппарата (рисунок 4.2.6).
- 14) Кнопка, клик по которой приведет к загрузке сцены для создания нового полетного задания.
- 15) Кнопка, клик по которой приведет к завершению работы приложения.

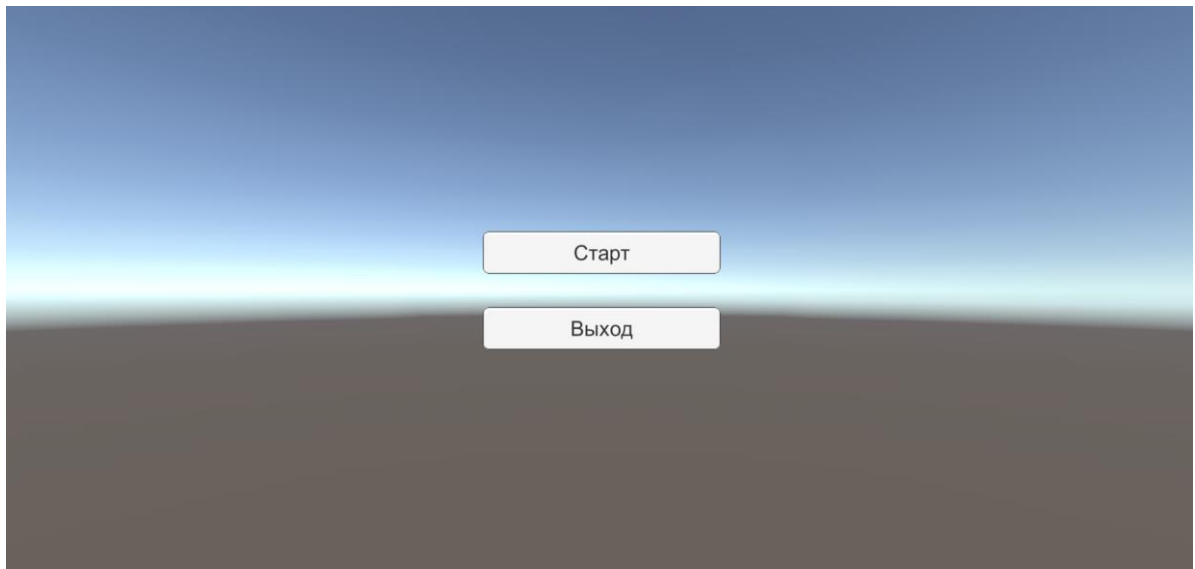


Рисунок 4.2.1 – Стартовый вид программы

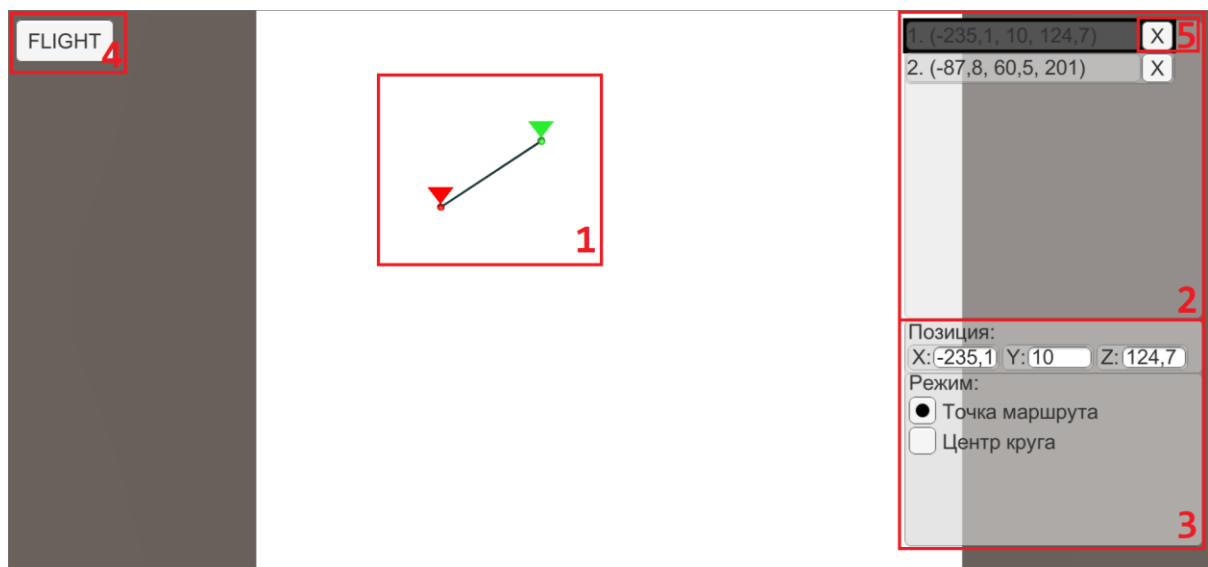


Рисунок 4.2.2 – Сцена формирования полетного задания

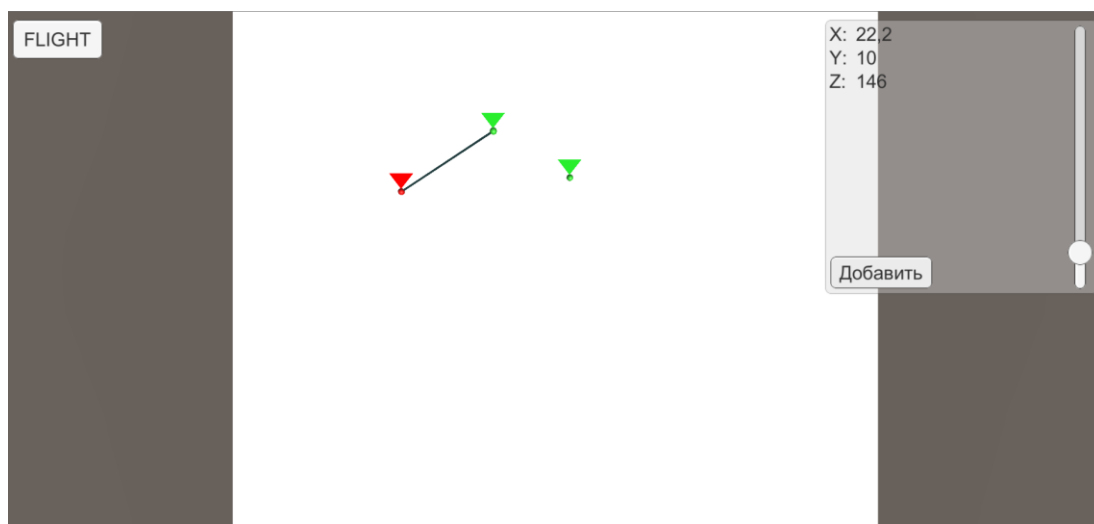


Рисунок 4.2.3 – Сцена формирования полетного задания, установка высоты опорной точки

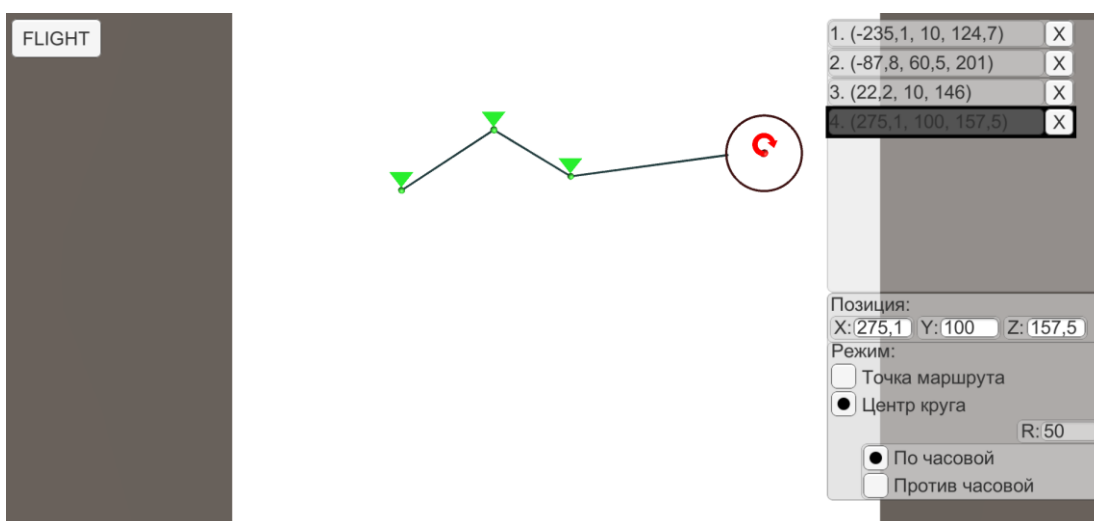


Рисунок 4.2.4 – Сцена формирования полетного задания, изменение режима опорной точки

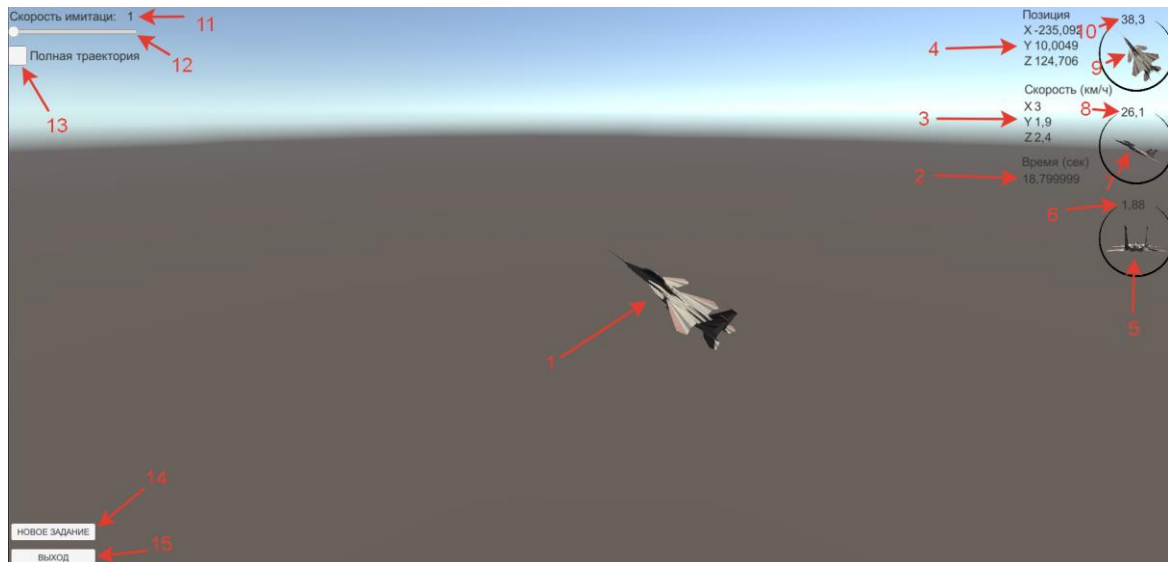


Рисунок 4.2.5 – Сцена демонстрации пролета летательного аппарата

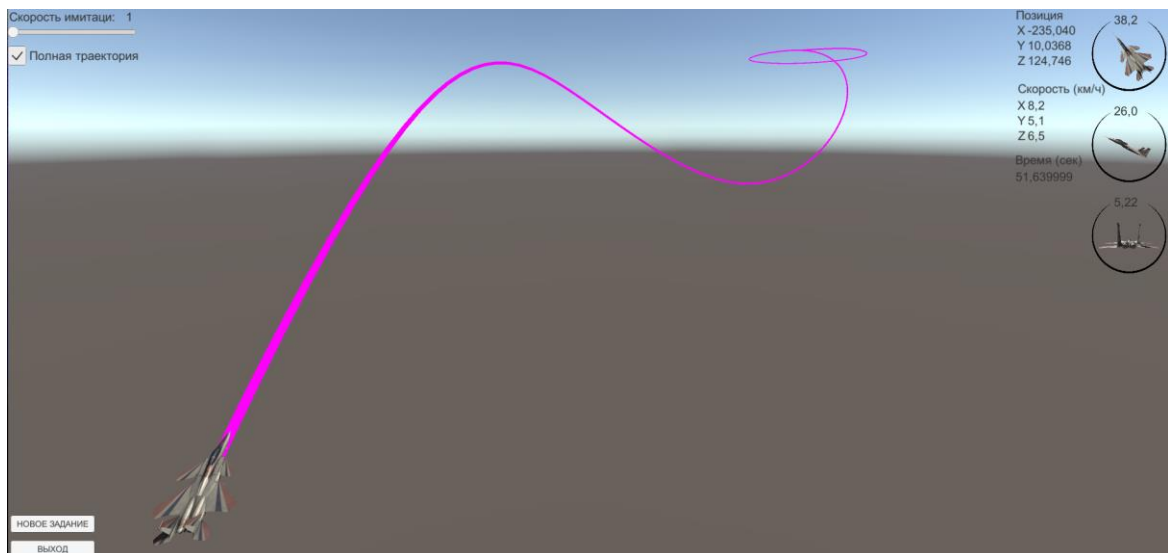


Рисунок 4.2.6 – Сцена демонстрации пролета летательного аппарата,
демонстрация полного маршрута

4.3 Тестирование и отладка

Для тестирования и отладки подсистемы имитации пролета летательного аппарата разработан отдельный класс `AuxiliaryFunction`, включающий методы визуализации хранимых математических выражений.

Методы по обработке и созданию деревьев операция являются достаточно сложными, для отлова и исправления ошибок применяется большое количество тестовых выражений, на основе которых можно судить о корректности работы

отдельных алгоритмов. Тестовые выражения поставляются в инфиксной нотации, они могут содержать любые операции из класса `ElementsEquation`, в том числе скобки, которые изменяют порядок выполнения операций, следовательно необходимо определить последовательность выполняемых операций в тестовых выражениях непосредственно перед применением того или иного метода. Для решения этой проблемы необходимо перевести исходное выражение из инфиксной нотации в постфиксную, т.к. главное преимущество постфиксной формы записи выражений заключается в однозначности порядка выполнения операций, благодаря этому отпадает необходимость введение приоритетов и ассоциативности операций. Существует алгоритм называемый «сортировочная станция» разработанный Эдсгером Дейкстра для перевода выражений в постфиксную нотацию.

Алгоритм использует стек операций, на вход подается выражение в инфиксной форме, на выходе получаем тоже выражение, но в постфиксной. В процессе перевода входная последовательность рассматривается слева направо, константы и переменные сразу отправляются в формируемую запись, в том же порядке что и встречаются в исходной последовательности. При появлении операции или скобки могут быть проведены следующие действия:

1) Если стек операций пуст или верхним элементом стека является открывающаяся скобка, то операция кладется в стек.

2) Если рассматриваемая операция имеет больший приоритет, чем верхняя операций в стеке, то рассматриваемая операция кладется в стек.

3) Если рассматриваемая операция имеет меньший или равный приоритет, чем верхняя операция в стеке, то операции, которые находятся в стеке, до ближайшей операции с приоритетом меньшим или все операции до ближайшей открывающейся скобки, переносятся в формируемую последовательность, а рассматриваемая операция заносится в стек.

Если текущая операция это открывающаяся скобка, то она кладется в стек, если текущая операция это закрывающаяся скобка, то в формируемую последовательность переносятся все операции из стека до ближайшей

открывающейся скобки. После завершения прохода по исходному выражению все операции, оставшиеся в стеке, переносятся в формируемую последовательность.

После получения правильной последовательности необходимо сформировать дерево операций, формирование происходит по стандартному алгоритму вычисления значения выражения в постфиксной форме, т.е. просматривается постфиксная запись слева направо, если встречается переменная или константа, то создается элемент дерева с соответствующей информационной частью, этот узел дерева отправляется в стек. Когда встречается операция, создается узел дерева с соответствующей операцией в информационной части, из стека берется необходимое количество операндов и назначаются наследниками вновь созданного узла, далее этот узел помещается в стек. Последовательность повторяется до тех пор, пока не закончится выражение, в результате вершиной стека станет искомое дерево операций.

После создания дерева операций и выполнением неких действий над ними необходимо проверить результат и сравнить его с эталонным. Просматривать значение ячеек памяти в отладчике не удобно и занимает много времени, особенно если дерево операций достаточно разрослось, для проверки корректности полученных результатов необходимо визуализировать полученное дерево, существует специальный язык описания графов, называемый DOT.

DOT – это язык описания графов, которые представляются с помощью текстового файла с расширением .gv или .dot в естественном для человека формате. Структура графа описывается в виде субграфов, элементы которого представляют конструкцию вида: `graph имя_графа {}`. Внутри фигурных скобок находятся инструкции, описывающие субграф, инструкции разделяются символом точки с запятой. Для визуализации графов представленных на языке DOT применяются множество программ, например Graphviz, OmniGraffle, ZGRViewer, VizierFX.

Формирование dot файла осуществляется прямым обходом дерева. На рисунке 4.3.1 демонстрируется одно из тестовых выражений, представленных в виде изображения, сформированного с помощью dot файла.

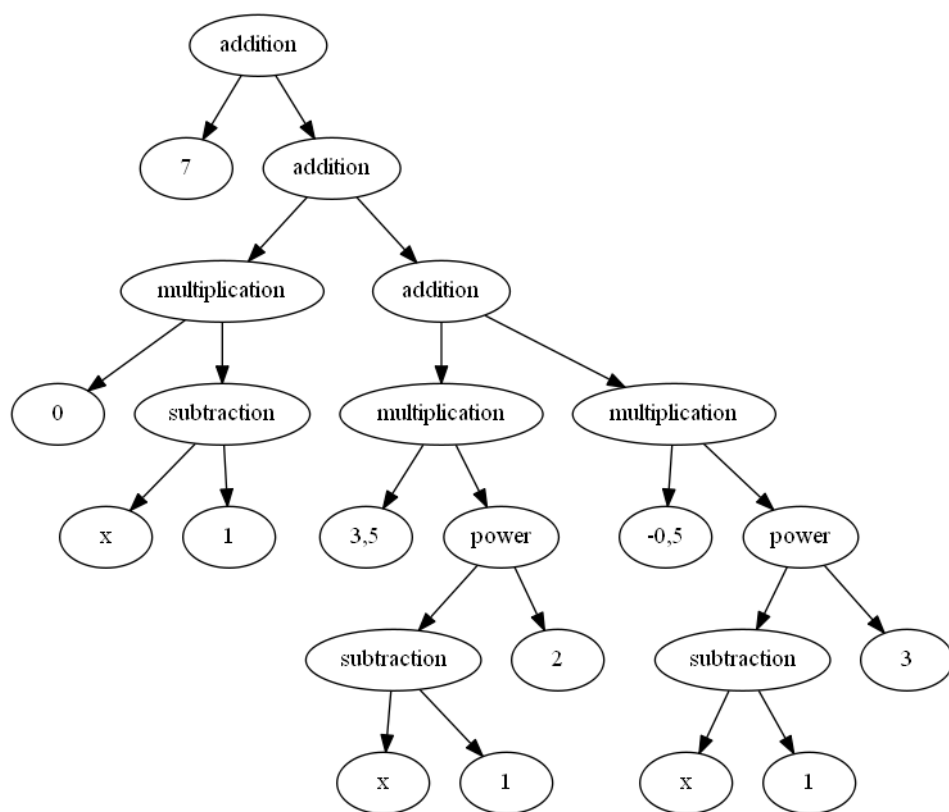


Рисунок 4.3.1 – Пример тестового выражения

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы были решены поставленные задачи:

1) Проанализирована задача и сформулированы требования к разработке подсистемы имитации движения летательного аппарата для программного имитатора закабинного пространства;

2) Определены спецификации к подсистеме имитации движения летательного аппарата;

3) Спроектирована подсистема имитации движения летательного аппарата;

4) Подсистема имитации движения летательного аппарата реализована, протестирована и отлажена.

Так как все задачи решены, то цель выпускной квалификационной работы, состоящей в разработке подсистемы имитации движения летательного аппарата для программного имитатора закабинного пространства, считается достигнутой. Проведенное моделирование показало, что подсистема соответствует сформулированным требованиям.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. ФГБОУ ВО «ОГУ имени И.С. Тургенева» Программный имитатор закабинного пространства для применения в составе технологического стенда комплексной настройки и проверки комплекса для обеспечения поисково-спасательных операций, проводимых с помощью летательных аппаратов в условиях Арктики [Текст] : пояснительная записка / Орёл, ФГБОУ ВО «ОГУ имени И.С. Тургенева», 2019. – 64 с.

2. Письменный, Д. Т. Конспект лекций по высшей математики: полный курс / Д. Т. Письменный. – 11-е изд. – М. : Айрис-пресс, 2013. – 608 с.

3. Бронштейн И. Н., Семендяев К. А. Справочник по математике для инженеров и учащихся втузов. – 13-е изд., исправленное. — М.: Наука, Гл. ред. физ.-мат. лит., 1986. –544 с.

4. Голованов Н.Н. Геометрическое моделирование. — М.: Издательство Физико-математической литературы, 2002, –472 с.

5. MATLAB.Exponenta [Электронный ресурс]. – Режим доступа: <http://matlab.exponenta.ru/> . – Дата обращения 24.05.19

6. Методы решения физико-математических задач [Электронный ресурс]. – Режим доступа: <https://1cov-edu.ru/> . – Дата обращения 25.05.19

7. MSDN [Электронный ресурс]. – Режим доступа: <https://msdn.microsoft.com/ru-ru/> . – Дата обращения 29.05.19

8. Unity Documentation [Электронный ресурс]. – Режим доступа: <https://docs.unity3d.com/ScriptReference/index.html> . – Дата обращения 31.05.19

ПРИЛОЖЕНИЕ А

(обязательное)

ЛИСТИНГ КОДА

Файл **AuxiliaryFunction.cs**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace MathCore
{
    /// <summary>
    /// Вспомогательный функционал
    /// </summary>
    public static class AuxiliaryFunction
    {
        /// <summary>
        /// Перевод из инфиксной нотации в постфиксную
        /// </summary>
        /// <param name="equation">Уравнение в инфиксной форме</param>
        public static List<EquationComponent> InfixNotationToPostfixNotation(EquationComponent[] equation)
        {
            // Приоритет операций
            Dictionary<ElementsEquation, int> precedence = new Dictionary<ElementsEquation, int>
            {
                [ElementsEquation.openBracket] = 0,
                [ElementsEquation.closedBracket] = 1,
                [ElementsEquation.addition] = 2,
                [ElementsEquation.subtraction] = 2,
                [ElementsEquation.multiplication] = 3,
                [ElementsEquation.division] = 3,
                [ElementsEquation.sinus] = 4,
                [ElementsEquation.cosine] = 4,
                [ElementsEquation.tangent] = 4,
                [ElementsEquation.cotangent] = 4,
                [ElementsEquation.arccosine] = 4,
                [ElementsEquation.arcsine] = 4,
                [ElementsEquation.arctangent] = 4,
                [ElementsEquation.arccotangent] = 4,
                [ElementsEquation.signum] = 4,
                [ElementsEquation.power] = 5
            };

            // Стек операций
            Stack<ElementsEquation> stackOperation = new Stack<ElementsEquation>();

            // Результирующая последовательность
            List<EquationComponent> resultSequence = new List<EquationComponent>();

            for (int i = 0; i < equation.Length; i++)
            {
                // Если на вход поступила переменная или число, то записываем это значение в результирующую
                // последовательность
            }
        }
    }
}

```

```

        if ((equation[i].typeComponent == ElementsEquation.number) || (equation[i].typeComponent ==
ElementsEquation.variable))
        {
            resultSequence.Add(equation[i]);
            continue;
        }

        // Если на вход поступила математическая операция и стек при этом пуст, то проталкиваем это значение в
стек операций
        if (stackOperation.Count == 0)
        {
            stackOperation.Push(equation[i].typeComponent);
            continue;
        }

        // Если на вход поступила открывающаяся скобка, то проталкиваем ее в стек
        if (equation[i].typeComponent == ElementsEquation.openBracket)
        {
            stackOperation.Push(equation[i].typeComponent);
            continue;
        }

        // Если на вход поступила закрывающаяся скобка, то извлекаем из стека все операции до первой
открывающейся скобки
        if (equation[i].typeComponent == ElementsEquation.closedBracket)
        {
            ElementsEquation tmpTypeElement = stackOperation.Pop();

            while (tmpTypeElement != ElementsEquation.openBracket)
            {
                resultSequence.Add(new EquationComponent(tmpTypeElement));
                tmpTypeElement = stackOperation.Pop();
            }

            continue;
        }

        // Общий случай
        // Извлекаем из стека операций все операции с большим или равным приоритетом в результирующую
последовательность, а потом помещаем в стек пришедшую операцию
        // Если операций в стеке с меньшим приоритетом то просто добавляем вновь пришедшую операцию в
стек
        ElementsEquation tmpType = stackOperation.Pop();
        if (precedence[tmpType] < precedence[equation[i].typeComponent])
        {
            stackOperation.Push(tmpType);
            stackOperation.Push(equation[i].typeComponent);
        }
        else
        {
            while (true)
            {
                if (precedence[tmpType] < precedence[equation[i].typeComponent])
                {
                    stackOperation.Push(tmpType);
                    stackOperation.Push(equation[i].typeComponent);
                    break;
                }
                else
                {
                    resultSequence.Add(new EquationComponent(tmpType));

```



```

        if (stackOperation.Count == 0)
        {
            stackOperation.Push(equation[i].typeComponent);
            break;
        }
        else
        {
            tmpType = stackOperation.Pop();
        }
    }
}
}

// Дописываем в результирующую последовательность операции оставшиеся в стеке
while (stackOperation.Count != 0)
{
    resultSequence.Add(new EquationComponent(stackOperation.Pop()));
}

return resultSequence;
}

/// <summary>
/// Перевод постфиксной нотации уравнения в дерево
/// </summary>
/// <param name="equation">
/// Уравнение в постфиксной форме
/// </param>
/// <returns>
/// Указатель на вершину дерева
/// </returns>
public static ElementTreeNonlinearEquation PostfixNotationToTree(List<EquationComponent> equation)
{
    // Стек операндов
    Stack<ElementTreeNonlinearEquation> stackOperand = new Stack<ElementTreeNonlinearEquation>();

    for (int i = 0; i < equation.Count; i++)
    {
        // Если на вход поступила переменная или число, то заносим это значение в стек операндов
        if ((equation[i].typeComponent == ElementsEquation.number) || (equation[i].typeComponent ==
ElementsEquation.variable))
        {
            stackOperand.Push(new ElementTreeNonlinearEquation(equation[i]));
            continue;
        }

        switch (equation[i].typeComponent)
        {
            case ElementsEquation.addition:
            {
                ElementTreeNonlinearEquation operandOne = stackOperand.Pop();
                ElementTreeNonlinearEquation operandTwo = stackOperand.Pop();
                ElementTreeNonlinearEquation result = new ElementTreeNonlinearEquation(equation[i]);

                operandOne.parent = result;
                operandTwo.parent = result;

                result.CreateDescendant();
                result.descendant.Add(operandTwo);
                result.descendant.Add(operandOne);
            }
        }
    }
}

```

```

        stackOperand.Push(result);
        break;
    }

case ElementsEquation.subtraction:
    {
        goto case ElementsEquation.addition;
        break;
    }

case ElementsEquation.multiplication:
    {
        goto case ElementsEquation.addition;
        break;
    }

case ElementsEquation.division:
    {
        goto case ElementsEquation.addition;
        break;
    }

case ElementsEquation.power:
    {
        goto case ElementsEquation.addition;
        break;
    }

case ElementsEquation.sinus:
    {
        ElementTreeNonlinearEquation operand = stackOperand.Pop();
        ElementTreeNonlinearEquation result = new ElementTreeNonlinearEquation(equation[i]);

        operand.parent = result;

        result.CreateDescendant();
        result.descendant.Add(operand);

        stackOperand.Push(result);
        break;
    }

case ElementsEquation.cosine:
    {
        goto case ElementsEquation.sinus;
        break;
    }

case ElementsEquation.tangent:
    {
        goto case ElementsEquation.sinus;
        break;
    }

case ElementsEquation.cotangent:
    {
        goto case ElementsEquation.sinus;
        break;
    }

case ElementsEquation.arccosine:
    {

```

```

        goto case ElementsEquation.sinus;
        break;
    }

    case ElementsEquation.arcsine:
    {
        goto case ElementsEquation.sinus;
        break;
    }

    case ElementsEquation.arctangent:
    {
        goto case ElementsEquation.sinus;
        break;
    }

    case ElementsEquation.arccotangent:
    {
        goto case ElementsEquation.sinus;
        break;
    }

    case ElementsEquation.signum:
    {
        goto case ElementsEquation.sinus;
        break;
    }
    }
}

return stackOperand.Pop();
}

public static void CreateDotFile(String path, ElementTreeNonlinearEquation root)
{
    StreamWriter sw = new StreamWriter(path);

    sw.WriteLine("digraph DG1 {");

    CreateDotFileTreeTraversal(sw, root, root.content.typeComponent.ToString(), 0);

    sw.WriteLine("}");

    sw.Close();
}

private static void CreateDotFileTreeTraversal(StreamWriter sw, ElementTreeNonlinearEquation root, String key, int
numberDescendant)
{
    String newKey;

    if (root.content.typeComponent == ElementsEquation.number)
    {
        newKey = key + Math.Round(Math.Abs(root.content.valueNumber)).ToString() +
root.content.typeComponent.ToString() + numberDescendant.ToString();
        sw.WriteLine(newKey + "[label=\"" + root.content.valueNumber.ToString() + "\"]");
    }
    else if (root.content.typeComponent == ElementsEquation.variable)
    {
        newKey = key + root.content.nameVariable + root.content.typeComponent.ToString() +
numberDescendant.ToString();
        sw.WriteLine(newKey + "[label=\"" + root.content.nameVariable + "\"]");
    }
}

```

```

    }
    else
    {
        newKey = key + root.content.typeComponent.ToString() + numberDescendant.ToString();
        sw.WriteLine(newKey + "[label=\"" + root.content.typeComponent.ToString() + "\"];");
    }

    if (root.parent != null)
    {
        sw.WriteLine(key + "->" + newKey + ";");
    }

    if (root.descendant == null)
    {
        return;
    }

    for (int i = 0; i < root.descendant.Count; i++)
    {
        CreateDotFileTreeTraversal(sw, root.descendant[i], newKey, i);
    }
}

public static void CreateWay(String path, List<FlightInformation> fi)
{
    StreamWriter sw = new StreamWriter(path);

    for (int i = 0; i < fi.Count; i++)
    {
        sw.WriteLine(
            i + "\t" +
            fi[i].time + "\t" +
            fi[i].position.x + "\t" +
            fi[i].position.z + "\t" +
            fi[i].position.y + "\t" +
            fi[i].vectorSpeed.x + "\t" +
            fi[i].vectorSpeed.y + "\t" +
            fi[i].vectorSpeed.z + "\t" +
            fi[i].kurs + "\t" +
            fi[i].kren + "\t" +
            fi[i].tangaj + "\t"
        );
    }

    sw.Close();
}
}
}

```

Файл ElementPiecewiseFunction.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MathCore
{
    /// <summary>
    /// Элемент кусочной функции
    /// </summary>
    public class ElementPiecewiseFunction

```

```

{
    /// <summary>
    /// Левая граница диапазона
    /// </summary>
    public double leftBoundaryRange;

    /// <summary>
    /// Правая граница диапазона
    /// </summary>
    public double rightBoundaryRange;

    /// <summary>
    /// Элемент кубического сплайна
    /// </summary>
    public ParametricRepresentation elementCubicSpline;

    /// <summary>
    /// Сопровождающий трехгранник Френе
    /// </summary>
    public TrihedronFrene trihedronFrene;

    /// <summary>
    /// Конструктор
    /// </summary>
    /// <param name="leftBoundaryRange">
    /// Левая граница диапазона
    /// </param>
    /// <param name="rightBoundaryRange">
    /// Правая граница диапазона
    /// </param>
    /// <param name="elementCubicSpline">
    /// Элемент кубического сплайна
    /// </param>
    /// <param name="trihedronFrene">
    /// Сопровождающий трехгранник Френе
    /// </param>
    public ElementPiecewiseFunction(double leftBoundaryRange, double rightBoundaryRange, ParametricRepresentation
elementCubicSpline, TrihedronFrene trihedronFrene)
    {
        this.leftBoundaryRange = leftBoundaryRange;
        this.rightBoundaryRange = rightBoundaryRange;
        this.elementCubicSpline = elementCubicSpline;
        this.trihedronFrene = trihedronFrene;
    }

    /// <summary>
    /// Конструктор
    /// </summary>
    public ElementPiecewiseFunction()
    {

    }
}
}

```

Файл ElementsEquation.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

```

```

namespace MathCore
{
    /// <summary>
    /// Элементы уравнения
    /// </summary>
    public enum ElementsEquation
    {
        /// <summary>
        /// Сложение
        /// </summary>
        addition,
        /// <summary>
        /// Вычитание
        /// </summary>
        subtraction,
        /// <summary>
        /// Умножение
        /// </summary>
        multiplication,
        /// <summary>
        /// Деление
        /// </summary>
        division,
        /// <summary>
        /// Степень
        /// </summary>
        power,
        /// <summary>
        /// Синус
        /// </summary>
        sinus,
        /// <summary>
        /// Арксинус
        /// </summary>
        arcsine,
        /// <summary>
        /// Косинус
        /// </summary>
        cosine,
        /// <summary>
        /// Арккосинус
        /// </summary>
        arccosine,
        /// <summary>
        /// Тангенс
        /// </summary>
        tangent,
        /// <summary>
        /// Арктангенс
        /// </summary>
        arctangent,
        /// <summary>
        /// Котангенс
        /// </summary>
        cotangent,
        /// <summary>
        /// Арккотангенс
        /// </summary>
        arccotangent,
        /// <summary>
        /// Открывающаяся скобка

```

```

    /// </summary>
    openBracket,
    /// <summary>
    /// Закрывающаяся скобка
    /// </summary>
    closedBracket,
    /// <summary>
    /// Переменная
    /// </summary>
    variable,
    /// <summary>
    /// Число
    /// </summary>
    number,
    /// <summary>
    /// Signum
    /// </summary>
    signum
}
}

```

Файл ElementTreeNonlinearEquation.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MathCore
{
    /// <summary>
    /// Элемент дерева для представления нелинейного уравнения
    /// </summary>
    public class ElementTreeNonlinearEquation
    {
        /// <summary>
        /// Указатель на родителя
        /// </summary>
        public ElementTreeNonlinearEquation parent;

        /// <summary>
        /// Список указателей на наследников
        /// </summary>
        public List<ElementTreeNonlinearEquation> descendant;

        /// <summary>
        /// Хранимая информация
        /// </summary>
        public EquationComponent content;

        public ElementTreeNonlinearEquation()
        {
            content = null;
        }

        public ElementTreeNonlinearEquation(EquationComponent content)
        {
            this.content = content;
        }

        public void CreateDescendant()

```

```

    {
        descendant = new List<ElementTreeNonlinearEquation>();
    }

    public void AddDescendant(ElementTreeNonlinearEquation newDescendant)
    {
        descendant.Add(newDescendant);
        descendant[descendant.Count - 1].parent = this;
    }

    public ElementTreeNonlinearEquation Clone()
    {
        return (ElementTreeNonlinearEquation)this.MemberwiseClone();
    }
}

```

Файл EquationComponent.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MathCore
{
    /// <summary>
    /// Компонент уравнения
    /// </summary>
    public class EquationComponent
    {
        /// <summary>
        /// Тип компонента
        /// </summary>
        public ElementsEquation typeComponent;
        /// <summary>
        /// Имя переменной (опционально)
        /// </summary>
        public String nameVariable;
        /// <summary>
        /// Число (опционально)
        /// </summary>
        public Double valueNumber;

        public EquationComponent(ElementsEquation type)
        {
            typeComponent = type;
        }

        public EquationComponent(double valueNumber)
        {
            typeComponent = ElementsEquation.number;
            this.valueNumber = valueNumber;
        }

        public EquationComponent(String nameVariable)
        {
            typeComponent = ElementsEquation.variable;
            this.nameVariable = nameVariable;
        }

        public EquationComponent(EquationComponent clone)

```



```

    {
        this.nameVariable = clone.nameVariable;
        this.typeComponent = clone.typeComponent;
        this.valueNumber = clone.valueNumber;
    }
}

```

Файл **FlightInformation.cs**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MathCore
{
    /// <summary>
    /// Полетная информация
    /// </summary>
    public class FlightInformation
    {
        /// <summary>
        /// Позиция
        /// </summary>
        public Vector3D position;

        /// <summary>
        /// Время
        /// </summary>
        public double time;

        /// <summary>
        /// Векторные скорости
        /// </summary>
        public Vector3D vectorSpeed;

        /// <summary>
        /// Курс
        /// </summary>
        public double kurs;

        /// <summary>
        /// Крен
        /// </summary>
        public double kren;

        /// <summary>
        /// Тангаж
        /// </summary>
        public double tangaj;

        /// <summary>
        /// Конструктор
        /// </summary>
        /// <param name="time">
        /// Время
        /// </param>
        /// <param name="position">
        /// Позиция
        /// </param>
        /// <param name="vectorSpeed">

```

```

/// Векторные скорости
/// </param>
/// <param name="kurs">
/// Курс
/// </param>
/// <param name="kren">
/// Крен
/// </param>
/// <param name="tangaj">
/// Тангаж
/// </param>
public FlightInformation(double time, Vector3D position, Vector3D vectorSpeed, double kurs, double kren, double
tangaj)
{
    this.time = time;
    this.position = position;
    this.vectorSpeed = vectorSpeed;
    this.kurs = kurs;
    this.kren = kren;
    this.tangaj = tangaj;
}

/// <summary>
/// Конструктор
/// </summary>
public FlightInformation()
{

}

}
}

```

Файл MathCore.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;

using MathCore;
using System.IO;

namespace MathCore
{
    public class MathCore
    {
        /// <summary>
        /// Преобразует матрицу заданную с помощью Array в матрицу хранимую с помощью List
        /// </summary>
        /// <param name="mtx"></param>
        /// <returns></returns>
        public static List<List<double>> MatrixArrayToLists(double[,] mtx)
        {
            List<List<double>> result = new List<List<double>>();

            for (int i = 0; i < mtx.GetLength(0); i++)
            {
                result.Add(new List<double>());
            }
        }
    }
}

```

```

        for (int j = 0; j < mtrx.GetLength(1); j++)
        {
            result[i].Add(mtrx[i, j]);
        }
    }

    return result;
}

/// <summary>
/// Округление с n знаками после запятой
/// </summary>
/// <param name="value">
/// Округляемое число
/// </param>
/// <param name="digits">
/// Количество знаков в дробной части, которые нужно оставить
/// </param>
/// <returns></returns>
public static double Rounding(double value, int digits)
{
    return Math.Round(value, digits);
}

/// <summary>
/// Округление с n знаками после запятой
/// </summary>
/// <param name="value">
/// Округляемое число
/// </param>
/// <param name="digits">
/// Количество знаков в дробной части, которые нужно оставить
/// </param>
/// <returns></returns>
public static double Rounding(double value, double digits)
{
    return Math.Round(value, digits.ToString().Length - 2);
}

/// <summary>
/// Округление по недостатку
/// </summary>
/// <param name="value">
/// Округляемое число
/// </param>
/// <param name="digits">
/// Погрешность округления
/// </param>
/// <returns></returns>
public static double RoundingShortage(double value, double digits)
{
    digits *= -1;

    if (value < 0)
    {
        return value - (Math.Pow(10, digits)) + (value % Math.Pow(10, digits));
    }

    return value - value % Math.Pow(10, digits);
}

```

```

/// <summary>
/// Округление по избытку
/// </summary>
/// <param name="value">
/// Округляемое число
/// </param>
/// <param name="digits">
/// Погрешность округления
/// </param>
/// <returns></returns>
public static double RoundingExcess(double value, double digits)
{
    digits *= -1;

    if (value < 0)
    {
        return value - value % Math.Pow(10, digits);
    }

    return value - (value % Math.Pow(10, digits)) + Math.Pow(10, digits);
}

/// <summary>
/// Дифференцирование уравнения
/// </summary>
/// <param name="root">
/// Указатель на корень дерева операций
/// </param>
/// <param name="variable">
/// Переменная по которой осуществляется дифференцирование
/// </param>
/// <returns>
/// Указатель на корень дерева продифференцированного уравнения
/// </returns>
public static ElementTreeNonlinearEquation Differentiation(ElementTreeNonlinearEquation root, String variable)
{
    switch (root.content.typeComponent)
    {
        case ElementsEquation.number:
        {
            return new ElementTreeNonlinearEquation(new EquationComponent((double)0));
            break;
        }

        case ElementsEquation.variable:
        {
            if (variable == root.content.nameVariable)
            {
                return new ElementTreeNonlinearEquation(new EquationComponent((double)1));
            }
            else
            {
                goto case ElementsEquation.number;
            }
            break;
        }

        case ElementsEquation.addition:
        {
            ElementTreeNonlinearEquation tmpRoot = new ElementTreeNonlinearEquation(new
EquationComponent(root.content.typeComponent));
            tmpRoot.CreateDescendant();

```

```

        tmpRoot.AddDescendant(Differentiation(root.descendant[0], variable));
        tmpRoot.AddDescendant(Differentiation(root.descendant[1], variable));

        return tmpRoot;
        break;
    }

    case ElementsEquation.subtraction:
    {
        goto case ElementsEquation.addition;
        break;
    }

    case ElementsEquation.multiplication:
    {
        ElementTreeNonlinearEquation tmpRoot = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.addition));
        tmpRoot.CreateDescendant();

        tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));
        tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));

        tmpRoot.descendant[0].CreateDescendant();
        tmpRoot.descendant[1].CreateDescendant();

        tmpRoot.descendant[0].AddDescendant(Differentiation(root.descendant[0], variable));
        tmpRoot.descendant[0].AddDescendant(root.descendant[1]);

        tmpRoot.descendant[1].AddDescendant(root.descendant[0]);
        tmpRoot.descendant[1].AddDescendant(Differentiation(root.descendant[1], variable));

        return tmpRoot;
        break;
    }

    case ElementsEquation.division:
    {
        ElementTreeNonlinearEquation tmpRoot = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.division));
        tmpRoot.CreateDescendant();

        tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.subtraction)));
        tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.power)));

        tmpRoot.descendant[0].CreateDescendant();

        tmpRoot.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));
        tmpRoot.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));

        tmpRoot.descendant[0].descendant[0].CreateDescendant();

        tmpRoot.descendant[0].descendant[0].AddDescendant(Differentiation(root.descendant[0], variable));
        tmpRoot.descendant[0].descendant[0].AddDescendant(root.descendant[1]);

        tmpRoot.descendant[0].descendant[1].CreateDescendant();

```

```

        tmpRoot.descendant[0].descendant[1].AddDescendant(root.descendant[0]);
        tmpRoot.descendant[0].descendant[1].AddDescendant(Differentiation(root.descendant[1], variable));

        tmpRoot.descendant[1].CreateDescendant();

        tmpRoot.descendant[1].AddDescendant(root.descendant[1]);
        tmpRoot.descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent((double)2)));

        return tmpRoot;

        break;
    }

    case ElementsEquation.sinus:
    {
        ElementTreeNonlinearEquation tmpRoot = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication));
        tmpRoot.CreateDescendant();

        tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.cosine)));
        tmpRoot.AddDescendant(Differentiation(root.descendant[0], variable));

        tmpRoot.descendant[0].CreateDescendant();

        tmpRoot.descendant[0].AddDescendant(root.descendant[0]);

        return tmpRoot;

        break;
    }

    case ElementsEquation.cosine:
    {
        ElementTreeNonlinearEquation tmpRoot = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication));
        tmpRoot.CreateDescendant();

        tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new EquationComponent((double)-1)));
        tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));

        tmpRoot.descendant[1].CreateDescendant();

        tmpRoot.descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.sinus)));
        tmpRoot.descendant[1].AddDescendant(Differentiation(root.descendant[0], variable));

        tmpRoot.descendant[1].descendant[0].CreateDescendant();

        tmpRoot.descendant[1].descendant[0].AddDescendant(root.descendant[0]);

        return tmpRoot;

        break;
    }

    case ElementsEquation.power:
    {

```

```

        ElementTreeNonlinearEquation tmpRoot = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication));
        tmpRoot.CreateDescendant();

        tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(root.descendant[1].content.valueNumber)));
        tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));

        tmpRoot.descendant[1].CreateDescendant();

        tmpRoot.descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.power)));
        tmpRoot.descendant[1].AddDescendant(Differentiation(root.descendant[0], variable));

        tmpRoot.descendant[1].descendant[0].CreateDescendant();

        tmpRoot.descendant[1].descendant[0].AddDescendant(root.descendant[0]);
        tmpRoot.descendant[1].descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(root.descendant[1].content.valueNumber - 1)));

        return tmpRoot;

        break;
    }

    case ElementsEquation.tangent:
    {
        ElementTreeNonlinearEquation tmpRoot = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication));
        tmpRoot.CreateDescendant();

        tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.division)));
        tmpRoot.AddDescendant(Differentiation(root.descendant[0], variable));

        tmpRoot.descendant[0].CreateDescendant();

        tmpRoot.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent((double)1)));
        tmpRoot.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.power)));

        tmpRoot.descendant[0].descendant[1].CreateDescendant();

        tmpRoot.descendant[0].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.cosine)));
        tmpRoot.descendant[0].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent((double)2)));

        tmpRoot.descendant[0].descendant[1].descendant[0].CreateDescendant();

        tmpRoot.descendant[0].descendant[1].descendant[0].AddDescendant(root.descendant[0]);

        return tmpRoot;

        break;
    }

    case ElementsEquation.cotangent:
    {

```

```

        ElementTreeNonlinearEquation tmpRoot = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication));
        tmpRoot.CreateDescendant();

        tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.division)));
        tmpRoot.AddDescendant(Differentiation(root.descendant[0], variable));

        tmpRoot.descendant[0].CreateDescendant();

        tmpRoot.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent((double)-1)));
        tmpRoot.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.power)));

        tmpRoot.descendant[0].descendant[1].CreateDescendant();

        tmpRoot.descendant[0].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.sinus)));
        tmpRoot.descendant[0].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent((double)2)));

        tmpRoot.descendant[0].descendant[1].descendant[0].CreateDescendant();

        tmpRoot.descendant[0].descendant[1].descendant[0].AddDescendant(root.descendant[0]);

        return tmpRoot;

    break;
}

case ElementsEquation.arcsine:
{
    ElementTreeNonlinearEquation tmpRoot = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication));
    tmpRoot.CreateDescendant();

    tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.division)));
    tmpRoot.AddDescendant(Differentiation(root.descendant[0], variable));

    tmpRoot.descendant[0].CreateDescendant();

    tmpRoot.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent((double)1)));
    tmpRoot.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.power)));

    tmpRoot.descendant[0].descendant[1].CreateDescendant();

    tmpRoot.descendant[0].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.subtraction)));
    tmpRoot.descendant[0].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent((double)0.5)));

    tmpRoot.descendant[0].descendant[1].descendant[0].CreateDescendant();

    tmpRoot.descendant[0].descendant[1].descendant[0].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent((double)1)));
    tmpRoot.descendant[0].descendant[1].descendant[0].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent(ElementsEquation.power)));

```



```

        tmpRoot.descendant[0].descendant[1].descendant[0].descendant[1].CreateDescendant();

        tmpRoot.descendant[0].descendant[1].descendant[0].descendant[1].AddDescendant(root.descendant[0]);
        tmpRoot.descendant[0].descendant[1].descendant[0].descendant[1].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent((double)2)));

        return tmpRoot;

        break;
    }

    case ElementsEquation.arccosine:
    {
        ElementTreeNonlinearEquation tmpRoot = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication));
        tmpRoot.CreateDescendant();

        tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.division)));
        tmpRoot.AddDescendant(Differentiation(root.descendant[0], variable));

        tmpRoot.descendant[0].CreateDescendant();

        tmpRoot.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent((double)-1)));
        tmpRoot.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.power)));

        tmpRoot.descendant[0].descendant[1].CreateDescendant();

        tmpRoot.descendant[0].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.subtraction)));
        tmpRoot.descendant[0].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent((double)0.5)));

        tmpRoot.descendant[0].descendant[1].descendant[0].CreateDescendant();

        tmpRoot.descendant[0].descendant[1].descendant[0].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent((double)1)));
        tmpRoot.descendant[0].descendant[1].descendant[0].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent(ElementsEquation.power)));

        tmpRoot.descendant[0].descendant[1].descendant[0].descendant[1].CreateDescendant();

        tmpRoot.descendant[0].descendant[1].descendant[0].descendant[1].AddDescendant(root.descendant[0]);
        tmpRoot.descendant[0].descendant[1].descendant[0].descendant[1].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent((double)2)));

        return tmpRoot;

        break;
    }

    case ElementsEquation.arctangent:
    {
        ElementTreeNonlinearEquation tmpRoot = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication));
        tmpRoot.CreateDescendant();

        tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.division)));
        tmpRoot.AddDescendant(Differentiation(root.descendant[0], variable));

```

```

        tmpRoot.descendant[0].CreateDescendant();

        tmpRoot.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent((double)1)));
        tmpRoot.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.addition)));

        tmpRoot.descendant[0].descendant[1].CreateDescendant();

        tmpRoot.descendant[0].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent((double)1)));
        tmpRoot.descendant[0].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.power)));

        tmpRoot.descendant[0].descendant[1].descendant[1].CreateDescendant();

        tmpRoot.descendant[0].descendant[1].descendant[1].AddDescendant(root.descendant[0]);
        tmpRoot.descendant[0].descendant[1].descendant[1].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent((double)2)));

        return tmpRoot;

        break;
    }

    case ElementsEquation.arccotangent:
    {
        ElementTreeNonlinearEquation tmpRoot = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication));
        tmpRoot.CreateDescendant();

        tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.division)));
        tmpRoot.AddDescendant(Differentiation(root.descendant[0], variable));

        tmpRoot.descendant[0].CreateDescendant();

        tmpRoot.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent((double)-1)));
        tmpRoot.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.addition)));

        tmpRoot.descendant[0].descendant[1].CreateDescendant();

        tmpRoot.descendant[0].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent((double)1)));
        tmpRoot.descendant[0].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.power)));

        tmpRoot.descendant[0].descendant[1].descendant[1].CreateDescendant();

        tmpRoot.descendant[0].descendant[1].descendant[1].AddDescendant(root.descendant[0]);
        tmpRoot.descendant[0].descendant[1].descendant[1].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent((double)2)));

        return tmpRoot;

        break;
    }
}

```

```

    return null;
}

/// <summary>
/// Оптимизация уравнения внутри дерева (умножение на ноль, сложение с нулем и т.д.)
/// </summary>
/// <param name="root">
/// Указатель на вершину дерева операций
/// </param>
/// <returns>
/// Оптимизированное уравнение
/// </returns>
public static void OptimizeEquation(ElementTreeNonlinearEquation root)
{
    /// <summary>
    /// Инверсия индекса в бинарном дереве операций;
    /// 0 => 1
    /// 1 => 0
    /// </summary>
    /// <param name="index">
    /// Индекс, который необходимо инвертировать
    /// </param>
    /// <returns>
    /// Инвертированный индекс
    /// </returns>
    int InvertingIndex(int index)
    {
        return (index == 0) ? 1 : 0;
    }

    if (root.descendant == null)
    {
        return;
    }

    for (int i = 0; i < root.descendant.Count; i++)
    {
        OptimizeEquation(root.descendant[i]);
    }

    switch (root.content.typeComponent)
    {
        case ElementsEquation.addition:
        {
            for (int i = 0; i < root.descendant.Count; i++)
            {
                if ((root.descendant[i].content.typeComponent == ElementsEquation.number) &&
                    (root.descendant[i].content.valueNumber == 0))
                {
                    root.content = root.descendant[InvertingIndex(i)].content;
                    root.descendant = root.descendant[InvertingIndex(i)].descendant;

                    break;
                }
            }

            break;
        }

        case ElementsEquation.subtraction:
        {

```

```

        if ((root.descendant[1].content.typeComponent == ElementsEquation.number) &&
(root.descendant[1].content.valueNumber == 0))
        {
            root.content = root.descendant[0].content;
            root.descendant = root.descendant[0].descendant;
        }

        break;
    }

    case ElementsEquation.multiplication:
    {
        for (int i = 0; i < root.descendant.Count; i++)
        {
            if ((root.descendant[i].content.typeComponent == ElementsEquation.number) &&
(root.descendant[i].content.valueNumber == 0))
            {
                root.content = root.descendant[i].content;
                root.descendant = root.descendant[i].descendant;

                break;
            }

            if ((root.descendant[i].content.typeComponent == ElementsEquation.number) &&
(root.descendant[i].content.valueNumber == 1))
            {
                root.content = root.descendant[InvertingIndex(i)].content;
                root.descendant = root.descendant[InvertingIndex(i)].descendant;

                break;
            }
        }

        break;
    }

    case ElementsEquation.division:
    {
        if ((root.descendant[1].content.typeComponent == ElementsEquation.number) &&
(root.descendant[1].content.valueNumber == 1))
        {
            root.content = root.descendant[0].content;
            root.descendant = root.descendant[0].descendant;
        }

        break;
    }

    case ElementsEquation.power:
    {
        if ((root.descendant[1].content.typeComponent == ElementsEquation.number) &&
(root.descendant[1].content.valueNumber == 1))
        {
            root.content = root.descendant[0].content;
            root.descendant = root.descendant[0].descendant;

            break;
        }

        if ((root.descendant[1].content.typeComponent == ElementsEquation.number) &&
(root.descendant[1].content.valueNumber == 0))
        {

```

```

        root.content = new EquationComponent((double)1);
        root.descendant = null;

        break;
    }

    break;
}
}

/// <summary>
/// Дифференцирование n-го порядка
/// </summary>
/// <param name="root">
/// Корень дерева операций, представляющий исходное уравнение
/// </param>
/// <param name="variable">
/// Переменная по которой осуществляется дифференцирование
/// </param>
/// <param name="order">
/// Порядок дифференцирования
/// </param>
/// <returns>
/// Корень дерева операций, представляющий уравнение после дифференцирования
/// </returns>
public static ElementTreeNonlinearEquation DifferentiationNOrder(ElementTreeNonlinearEquation root, String
variable, int order)
{
    ElementTreeNonlinearEquation result = root;
    for (int i = 0; i < order; i++)
    {
        result = Differentiation(result, variable);
        OptimizeEquation(result);
    }

    return result;
}

/// <summary>
/// Решение СЛАУ методом Гауса
/// </summary>
/// <param name="coefficientMatrix">
/// Расширенная матрица
/// </param>
/// <returns>
/// Массив корней
/// </returns>
public static double[] DecisionSLAEMethodGauss(double[,] coefficientMatrix)
{
    List<List<double>> mtrxCoefficient = MatrixArrayToLists(coefficientMatrix);
    double[] result = new double[coefficientMatrix.GetLength(1) - 1];

    for (int i = 0; i < coefficientMatrix.GetLength(0); i++)
    {
        for (int j = i; j < mtrxCoefficient.Count; j++)
        {
            if ((mtrxCoefficient[j][i] == 0))
            {
                continue;
            }

```

```

        double maxCff = mtrxCoefficient[j][i];

        for (int k = i; k < mtrxCoefficient[i].Count; k++)
        {
            mtrxCoefficient[j][k] /= maxCff;
        }

    }

    for (int j = i + 1; j < mtrxCoefficient.Count; j++)
    {
        if (mtrxCoefficient[j][i] == 0)
        {
            continue;
        }

        for (int k = i; k < mtrxCoefficient[i].Count; k++)
        {
            mtrxCoefficient[j][k] -= mtrxCoefficient[i][k];
        }

    }
}

for (int i = result.Length - 1; i >= 0; i--)
{
    double tmp = mtrxCoefficient[i][mtrxCoefficient[i].Count - 1];

    for (int j = i + 1; j < mtrxCoefficient[i].Count - 1; j++)
    {
        tmp -= mtrxCoefficient[i][j] * result[j];
    }

    result[i] = tmp;
}

return result;
}

/// <summary>
/// Вычисление значения уравнения при заданных значениях переменных
/// </summary>
/// <param name="valueVariable">
/// Словарь, где ключ имя переменной
/// </param>
/// <param name="root">
/// Корень дерева
/// </param>
/// <returns>
/// Значение уравнения
/// </returns>
public static double CalculateEquationValue(Dictionary<String, double> valueVariable,
ElementTreeNonlinearEquation root)
{
    switch (root.content.typeComponent)
    {
        {
            case ElementsEquation.addition:
            {
                return CalculateEquationValue(valueVariable, root.descendant[0]) +
CalculateEquationValue(valueVariable, root.descendant[1]);
            }
        }
    }
}

```

```

        break;
    }

    case ElementsEquation.subtraction:
    {
        return CalculateEquationValue(valueVariable, root.descendant[0]) - CalculateEquationValue(valueVariable,
root.descendant[1]);
        break;
    }

    case ElementsEquation.multiplication:
    {
        return CalculateEquationValue(valueVariable, root.descendant[0]) *
CalculateEquationValue(valueVariable, root.descendant[1]);
        break;
    }

    case ElementsEquation.division:
    {
        return CalculateEquationValue(valueVariable, root.descendant[0]) / CalculateEquationValue(valueVariable,
root.descendant[1]);
        break;
    }

    case ElementsEquation.power:
    {
        return Math.Pow(CalculateEquationValue(valueVariable, root.descendant[0]),
CalculateEquationValue(valueVariable, root.descendant[1]));
        break;
    }

    case ElementsEquation.sinus:
    {
        return Math.Sin(CalculateEquationValue(valueVariable, root.descendant[0]));
        break;
    }

    case ElementsEquation.signum:
    {
        return Math.Sign(CalculateEquationValue(valueVariable, root.descendant[0]));
        break;
    }

    case ElementsEquation.arcsine:
    {
        return Math.Asin(CalculateEquationValue(valueVariable, root.descendant[0]));
        break;
    }

    case ElementsEquation.cosine:
    {
        return Math.Cos(CalculateEquationValue(valueVariable, root.descendant[0]));
        break;
    }

    case ElementsEquation.arccosine:
    {
        return Math.Acos(CalculateEquationValue(valueVariable, root.descendant[0]));
        break;
    }

    case ElementsEquation.tangent:

```

```

        {
            return Math.Tan(CalculateEquationValue(valueVariable, root.descendant[0]));
            break;
        }

    case ElementsEquation.arctangent:
    {
        return Math.Atan(CalculateEquationValue(valueVariable, root.descendant[0]));
        break;
    }

    case ElementsEquation.cotangent:
    {
        return 1 / Math.Tan(CalculateEquationValue(valueVariable, root.descendant[0]));
        break;
    }

    case ElementsEquation.arccotangent:
    {
        return Math.PI / 2 - Math.Atan(CalculateEquationValue(valueVariable, root.descendant[0]));
        break;
    }

    case ElementsEquation.variable:
    {
        return valueVariable[root.content.nameVariable];
        break;
    }

    case ElementsEquation.number:
    {
        return root.content.valueNumber;
        break;
    }

    default:
    {
        return 0;
        break;
    }
}

}

/// <summary>
/// Возведение выражения в заданную степень
/// </summary>
/// <param name="root">
/// Корень дерева операций
/// </param>
/// <param name="degree">
/// Степень, в которую необходимо возвести выражение
/// </param>
/// <returns>
/// Выражение в заданной степени
/// </returns>
public static ElementTreeNonlinearEquation ExponentiationExpression(ElementTreeNonlinearEquation root, double
degree)
{
    ElementTreeNonlinearEquation newRoot = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.power));
    newRoot.CreateDescendant();
}

```



```

newRoot.AddDescendant(root.Clone());
newRoot.AddDescendant(new ElementTreeNonlinearEquation(new EquationComponent(degree)));

return newRoot;
}

/// <summary>
/// Возведение вектора в степень
/// </summary>
/// <param name="vector">
/// Параметрическое представление вектора
/// </param>
/// <param name="degree">
/// Степень, в которую необходимо возвести выражение
/// </param>
/// <returns>
/// Вектор в заданной степени
/// </returns>
public static ParametricRepresentation ConstructionVectorPower(ParametricRepresentation vector, double degree)
{
    return new ParametricRepresentation(ExponentiationExpression(vector.x.Clone(), 2),
    ExponentiationExpression(vector.y.Clone(), 2), ExponentiationExpression(vector.z.Clone(), 2));
}

/// <summary>
/// Модуль вектора разложенного по ортам
/// </summary>
/// <param name="_vector">
/// Параметрическое представление векторай
/// </param>
/// <returns>
/// Модуль вектора
/// </returns>
public static ElementTreeNonlinearEquation ModuleVector(ParametricRepresentation _vector)
{
    ParametricRepresentation vector = new ParametricRepresentation(_vector.x.Clone(), _vector.y.Clone(),
    _vector.z.Clone());
    ParametricRepresentation squareVector = ConstructionVectorPower(vector, 2);

    ElementTreeNonlinearEquation result = new ElementTreeNonlinearEquation(new
    EquationComponent(ElementsEquation.addition));
    result.CreateDescendant();
    result.AddDescendant(new ElementTreeNonlinearEquation(new
    EquationComponent(ElementsEquation.addition)));
    result.AddDescendant(squareVector.z);

    result.descendant[0].CreateDescendant();
    result.descendant[0].AddDescendant(squareVector.x);
    result.descendant[0].AddDescendant(squareVector.y);

    return ExponentiationExpression(result, 0.5);
}

/// <summary>
/// Скалярное произведение векторов
/// </summary>
/// <param name="_a">
/// Первый вектор
/// </param>
/// <param name="_b">
/// Второй вектор

```

```

/// </param>
/// <returns>
/// Результат скалярного произведения векторов
/// </returns>
public static ElementTreeNonlinearEquation ScalarProductVector(ParametricRepresentation _a,
ParametricRepresentation _b)
{
    ParametricRepresentation a = new ParametricRepresentation(_a.x.Clone(), _a.y.Clone(), _a.z.Clone());
    ParametricRepresentation b = new ParametricRepresentation(_b.x.Clone(), _b.y.Clone(), _b.z.Clone());
    ElementTreeNonlinearEquation result = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.addition));

    result.CreateDescendant();
    result.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.addition)));
    result.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));

    result.descendant[1].CreateDescendant();
    result.descendant[1].AddDescendant(a.z);
    result.descendant[1].AddDescendant(b.z);

    result.descendant[0].CreateDescendant();
    result.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));
    result.descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));

    result.descendant[0].descendant[0].CreateDescendant();
    result.descendant[0].descendant[0].AddDescendant(a.x);
    result.descendant[0].descendant[0].AddDescendant(b.x);

    result.descendant[0].descendant[1].CreateDescendant();
    result.descendant[0].descendant[1].AddDescendant(a.y);
    result.descendant[0].descendant[1].AddDescendant(b.y);

    return result;
}

/// <summary>
/// Разность векторов
/// </summary>
/// <param name="_a">
/// Первый вектор
/// </param>
/// <param name="_b">
/// Второй вектор
/// </param>
/// <returns>
/// Результат вычитания
/// </returns>
public static ParametricRepresentation DifferenceVector(ParametricRepresentation _a, ParametricRepresentation _b)
{
    ParametricRepresentation a = new ParametricRepresentation(_a.x.Clone(), _a.y.Clone(), _a.z.Clone());
    ParametricRepresentation b = new ParametricRepresentation(_b.x.Clone(), _b.y.Clone(), _b.z.Clone());
    ParametricRepresentation result = new ParametricRepresentation();

    ElementTreeNonlinearEquation tmpRoot = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.subtraction));
    tmpRoot.CreateDescendant();
    tmpRoot.AddDescendant(a.x);
    tmpRoot.AddDescendant(b.x);

```

```

result.x = tmpRoot;

tmpRoot = new ElementTreeNonlinearEquation(new EquationComponent(ElementsEquation.subtraction));
tmpRoot.CreateDescendant();
tmpRoot.AddDescendant(a.y);
tmpRoot.AddDescendant(b.y);
result.y = tmpRoot;

tmpRoot = new ElementTreeNonlinearEquation(new EquationComponent(ElementsEquation.subtraction));
tmpRoot.CreateDescendant();
tmpRoot.AddDescendant(a.z);
tmpRoot.AddDescendant(b.z);
result.z = tmpRoot;

return result;
}

/// <summary>
/// Смешанное произведение векторов
/// </summary>
/// <param name="_a">
/// Первый вектор
/// </param>
/// <param name="_b">
/// Второй вектор
/// </param>
/// <returns>
/// Результат смешанного произведения
/// </returns>
public static ParametricRepresentation MixedProductVector(ParametricRepresentation _a, ParametricRepresentation
_b)
{
    ParametricRepresentation a = new ParametricRepresentation(_a.x.Clone(), _a.y.Clone(), _a.z.Clone());
    ParametricRepresentation b = new ParametricRepresentation(_b.x.Clone(), _b.y.Clone(), _b.z.Clone());
    ParametricRepresentation result = new ParametricRepresentation();

    ElementTreeNonlinearEquation tmpRoot = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.subtraction));
    tmpRoot.CreateDescendant();
    tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));
    tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));

    tmpRoot.descendant[0].CreateDescendant();
    tmpRoot.descendant[0].AddDescendant(a.y);
    tmpRoot.descendant[0].AddDescendant(b.z);

    tmpRoot.descendant[1].CreateDescendant();
    tmpRoot.descendant[1].AddDescendant(a.z);
    tmpRoot.descendant[1].AddDescendant(b.y);

    result.x = tmpRoot;

    tmpRoot = new ElementTreeNonlinearEquation(new EquationComponent(ElementsEquation.subtraction));
    tmpRoot.CreateDescendant();
    tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));
    tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));

    tmpRoot.descendant[0].CreateDescendant();

```

```

tmpRoot.descendant[0].AddDescendant(a.z);
tmpRoot.descendant[0].AddDescendant(b.x);

tmpRoot.descendant[1].CreateDescendant();
tmpRoot.descendant[1].AddDescendant(a.x);
tmpRoot.descendant[1].AddDescendant(b.z);

result.y = tmpRoot;

tmpRoot = new ElementTreeNonlinearEquation(new EquationComponent(ElementsEquation.subtraction));
tmpRoot.CreateDescendant();
tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));
tmpRoot.AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));

tmpRoot.descendant[0].CreateDescendant();
tmpRoot.descendant[0].AddDescendant(a.x);
tmpRoot.descendant[0].AddDescendant(b.y);

tmpRoot.descendant[1].CreateDescendant();
tmpRoot.descendant[1].AddDescendant(a.y);
tmpRoot.descendant[1].AddDescendant(b.x);

result.z = tmpRoot;

return result;
}

/// <summary>
/// Создать трехгранник Ферне
/// </summary>
/// <param name="equation">
/// Параметрическое представления кривой
/// </param>
/// <returns>
/// Трехгранник Френе
/// </returns>
public static TrihedronFrene CreateTrihedronFrene(ParametricRepresentation equation)
{
    //https://1cov-edu.ru/termeh/kinematika/tochki/estestvennyy-trekhgrannik/

    TrihedronFrene frene = new TrihedronFrene();

    // касательная
    ParametricRepresentation vectorSpeed = new ParametricRepresentation(DifferentiationNOrder(equation.x, "t", 1),
DifferentiationNOrder(equation.y, "t", 1), DifferentiationNOrder(equation.z, "t", 1));
    ElementTreeNonlinearEquation moduleSpeed = ModuleVector(vectorSpeed);

    frene.tangentVector = new ParametricRepresentation();

    frene.tangentVector.x = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.division));
    frene.tangentVector.x.CreateDescendant();
    frene.tangentVector.x.AddDescendant(vectorSpeed.x);
    frene.tangentVector.x.AddDescendant(moduleSpeed);

    frene.tangentVector.y = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.division));
    frene.tangentVector.y.CreateDescendant();
    frene.tangentVector.y.AddDescendant(vectorSpeed.y);
    frene.tangentVector.y.AddDescendant(moduleSpeed);

```

```

        frene.tangentVector.z = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.division));
        frene.tangentVector.z.CreateDescendant();
        frene.tangentVector.z.AddDescendant(vectorSpeed.z);
        frene.tangentVector.z.AddDescendant(moduleSpeed);

        // нормаль
        ParametricRepresentation vectorAcceleration = new
ParametricRepresentation(DifferentiationNOrder(vectorSpeed.x, "t", 1), DifferentiationNOrder(vectorSpeed.y, "t", 1),
DifferentiationNOrder(vectorSpeed.z, "t", 1));
        // касательное ускорение
        ElementTreeNonlinearEquation tangentialAcceleration = ScalarProductVector(frene.tangentVector,
vectorAcceleration);
        // нормальное ускорение
        ParametricRepresentation normalAcceleration = DifferenceVector(vectorAcceleration, new
ParametricRepresentation(tangentialAcceleration, tangentialAcceleration, tangentialAcceleration));
        // Модуль вектора нормального ускорения
        ElementTreeNonlinearEquation moduleVectorNormalAcceleration = ModuleVector(normalAcceleration);

        frene.normalVector = new ParametricRepresentation();

        frene.normalVector.x = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.division));
        frene.normalVector.x.CreateDescendant();
        frene.normalVector.x.AddDescendant(normalAcceleration.x);
        frene.normalVector.x.AddDescendant(moduleVectorNormalAcceleration);

        frene.normalVector.y = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.division));
        frene.normalVector.y.CreateDescendant();
        frene.normalVector.y.AddDescendant(normalAcceleration.y);
        frene.normalVector.y.AddDescendant(moduleVectorNormalAcceleration);

        frene.normalVector.z = new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.division));
        frene.normalVector.z.CreateDescendant();
        frene.normalVector.z.AddDescendant(normalAcceleration.z);
        frene.normalVector.z.AddDescendant(moduleVectorNormalAcceleration);

        // бинормаль
        frene.binormalVector = MixedProductVector(frene.tangentVector, frene.normalVector);

        return frene;
    }

    /// <summary>
    /// Создание кубического сплайна
    /// </summary>
    /// <param name="point">
    /// Массив точек, где [i, 0] = x, [i, 1] = f(x)
    /// </param>
    /// <returns>
    /// Массив указателей на деревья функций
    /// </returns>
    public static ElementTreeNonlinearEquation[] CreateCubicSpline(double[,] point)
    {
        //http://matlab.exponenta.ru/spline/book1/12.php

        //Для сплайнов проходящих через 2 или 3 точки особый случай
        //Для 2 точек докидываем доаполнительную точку
        //Для 3 точек расширенная матрица обращается в линейное уравнение

```

```

double[,] point_ = point;

double GetH(int i)
{
    return point_[i + 1, 0] - point_[i, 0];
}

double GetAlpha(int i)
{
    return 1 / GetH(i);
}

double GetBeta(int i)
{
    return 2 * (1 / GetH(i) + 1 / GetH(i + 1));
}

double GetLambda(int i)
{
    return 1 / GetH(i + 1);
}

double GetFunctionDifference(int i)
{
    return (point_[i + 1, 1] - point_[i, 1]) / (point_[i + 1, 0] - point_[i, 0]);
}

double GetDelta(int i)
{
    return 3 * ((GetFunctionDifference(i + 1) / GetH(i + 1)) + (GetFunctionDifference(i) / GetH(i)));
}

double GetC(int i, List<double> _B)
{
    return (3 * GetFunctionDifference(i) - _B[i + 1] - 2 * _B[i]) / (GetH(i));
}

double GetD(int i, List<double> _B)
{
    return (_B[i] + _B[i + 1] - 2 * GetFunctionDifference(i)) / (Math.Pow(GetH(i), 2));
}

if (point.GetLength(0) == 2)
{
    point_ = new double[3, 2];

    point_[0, 0] = point[0, 0];
    point_[0, 1] = point[0, 1];

    point_[1, 0] = point[1, 0];
    point_[1, 1] = point[1, 1];

    point_[2, 0] = point[1, 0] * 2;
    point_[2, 1] = point[1, 1] * 2;
}

double[,] mixedMatrix = new double[point_.GetLength(0) - 2, point_.GetLength(0) - 1];

mixedMatrix[0, 0] = GetBeta(0);
mixedMatrix[0, 1] = GetLambda(0);
mixedMatrix[0, mixedMatrix.GetLength(1) - 1] = GetDelta(0);

```

```

for (int i = 1; i < mixedMatrix.GetLength(0) - 1; i++)
{
    mixedMatrix[i, i - 1] = GetAlpha(i);
    mixedMatrix[i, i] = GetBeta(i);
    mixedMatrix[i, i + 1] = GetLambda(i);
    mixedMatrix[i, mixedMatrix.GetLength(1) - 1] = GetDelta(i);
}

if (point_.GetLength(0) > 3)
{
    mixedMatrix[mixedMatrix.GetLength(0) - 1, mixedMatrix.GetLength(0) - 2] =
GetAlpha(mixedMatrix.GetLength(0) - 1);
    mixedMatrix[mixedMatrix.GetLength(0) - 1, mixedMatrix.GetLength(0) - 1] =
GetBeta(mixedMatrix.GetLength(0) - 1);
    mixedMatrix[mixedMatrix.GetLength(0) - 1, mixedMatrix.GetLength(1) - 1] =
GetDelta(mixedMatrix.GetLength(0) - 1);
}

List<double> B = new List<double>();
B.Add(0);
B.AddRange(DecisionSLAEMethodGauss(mixedMatrix));
B.Add(0);

List<double> C = new List<double>();
List<double> D = new List<double>();

for (int i = 0; i < B.Count - 1; i++)
{
    C.Add(GetC(i, B));
    D.Add(GetD(i, B));
}

int count = D.Count;

if (point.GetLength(0) == 2)
{
    count = D.Count - 1;
}

ElementTreeNonlinearEquation[] result = new ElementTreeNonlinearEquation[count];

for (int i = 0; i < count; i++)
{
    result[i] = new ElementTreeNonlinearEquation(new EquationComponent(ElementsEquation.addition));
    result[i].CreateDescendant();

    result[i].AddDescendant(new ElementTreeNonlinearEquation(new EquationComponent(point_[i, 1])));
    result[i].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.addition)));

    result[i].descendant[1].CreateDescendant();
    result[i].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));
    result[i].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.addition)));

    result[i].descendant[1].descendant[0].CreateDescendant();
    result[i].descendant[1].descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(B[i])));
    result[i].descendant[1].descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.subtraction)));

```

```

        result[i].descendant[1].descendant[0].descendant[1].CreateDescendant();
        result[i].descendant[1].descendant[0].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent("t")));
        result[i].descendant[1].descendant[0].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(point_[i, 0]));

        result[i].descendant[1].descendant[1].CreateDescendant();
        result[i].descendant[1].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));
        result[i].descendant[1].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.multiplication)));

        result[i].descendant[1].descendant[1].descendant[0].CreateDescendant();
        result[i].descendant[1].descendant[1].descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(C[i]));
        result[i].descendant[1].descendant[1].descendant[0].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.power)));

        result[i].descendant[1].descendant[1].descendant[0].descendant[1].CreateDescendant();
        result[i].descendant[1].descendant[1].descendant[0].descendant[1].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent(ElementsEquation.subtraction)));
        result[i].descendant[1].descendant[1].descendant[0].descendant[1].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent(2)));

        result[i].descendant[1].descendant[1].descendant[0].descendant[1].descendant[0].CreateDescendant();
        result[i].descendant[1].descendant[1].descendant[0].descendant[1].descendant[0].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent("t")));
        result[i].descendant[1].descendant[1].descendant[0].descendant[1].descendant[0].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent(point_[i, 0]));

        result[i].descendant[1].descendant[1].descendant[1].CreateDescendant();
        result[i].descendant[1].descendant[1].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(D[i]));
        result[i].descendant[1].descendant[1].descendant[1].AddDescendant(new ElementTreeNonlinearEquation(new
EquationComponent(ElementsEquation.power)));

        result[i].descendant[1].descendant[1].descendant[1].descendant[1].CreateDescendant();
        result[i].descendant[1].descendant[1].descendant[1].descendant[1].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent(ElementsEquation.subtraction)));
        result[i].descendant[1].descendant[1].descendant[1].descendant[1].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent(3)));

        result[i].descendant[1].descendant[1].descendant[1].descendant[1].descendant[0].CreateDescendant();
        result[i].descendant[1].descendant[1].descendant[1].descendant[1].descendant[0].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent("t")));
        result[i].descendant[1].descendant[1].descendant[1].descendant[1].descendant[0].AddDescendant(new
ElementTreeNonlinearEquation(new EquationComponent(point_[i, 0]));
    }

    return result;
}

/// <summary>
/// Возвращает угол в градусах, тангенс которого равен отношению двух указанных чисел.
/// </summary>
/// <param name="x">
/// Координата x точки
/// </param>
/// <param name="y">
/// Координата y точки
/// </param>

```



```

/// <returns>
/// Угол в градусах
/// </returns>
public static double ArcTangent2D(double x, double y)
{
    return Math.Atan2(y, x) * 180 / Math.PI;
}

/// <summary>
/// Возвращает угол в радианах, тангенс которого равен отношению двух указанных чисел.
/// </summary>
/// <param name="x">
/// Координата x точки
/// </param>
/// <param name="y">
/// Координата y точки
/// </param>
/// <returns>
/// Угол в радианах
/// </returns>
public static double ArcTangent2R(double x, double y)
{
    return Math.Atan2(y, x);
}

/// <summary>
/// Перевод из градусов в радианы
/// </summary>
/// <param name="degree">
/// Значение угла в градусах
/// </param>
/// <returns>
/// Значение угла в радианах
/// </returns>
public static double DegreeToRadian(double degree)
{
    return Convert.ToSingle(degree * Math.PI / 180);
}

/// <summary>
/// Перевод из радиан в градусы
/// </summary>
/// <param name="degree">
/// Значение угла в радианах
/// </param>
/// <returns>
/// Значение угла в градусах
/// </returns>
public static double RadianToDegree(double degree)
{
    return Convert.ToSingle(degree * 180 / Math.PI);
}

/// <summary>
/// Создание маршрута
/// </summary>
/// <param name="points">
/// Список опорных точек
/// </param>
/// <param name="stepTime">
/// Шаг по времени
/// </param>

```

```

/// <returns>
/// Список состояний ЛА в моменты времени
/// </returns>
public static List<FlightInformation> CreateWay(List<Vector3D> points, double stepTime, double maxSpeed)
{
    return CreateWay(stepTime, CreatePiecewiseFunctionPosition(points, maxSpeed));
}

/// <summary>
/// Создание маршрута
/// </summary>
/// <param name="stepTime">
/// Временной шаг
/// </param>
/// <param name="piecewiseFunction">
/// Кусочная функция, представляющая маршрут
/// </param>
/// <returns>
/// Список состояний ЛА в моменты времени
/// </returns>
public static List<FlightInformation> CreateWay(double stepTime, PiecewiseFunction piecewiseFunction)
{
    List<FlightInformation> result = new List<FlightInformation>();

    int currentFunction = 0;
    double currentTime = piecewiseFunction.function[0].leftBoundaryRange;
    Dictionary<String, double> valueVariable = new Dictionary<string, double>();
    valueVariable.Add("t", currentTime);

    ElementTreeNonlinearEquation derivativeX =
    DifferentiationNOrder(piecewiseFunction.function[currentFunction].elementCubicSpline.x, "t", 1);
    ElementTreeNonlinearEquation derivativeY =
    DifferentiationNOrder(piecewiseFunction.function[currentFunction].elementCubicSpline.y, "t", 1);
    ElementTreeNonlinearEquation derivativeZ =
    DifferentiationNOrder(piecewiseFunction.function[currentFunction].elementCubicSpline.z, "t", 1);

    ElementTreeNonlinearEquation derivative2X = DifferentiationNOrder(derivativeX, "t", 1);
    ElementTreeNonlinearEquation derivative2Y = DifferentiationNOrder(derivativeY, "t", 1);
    ElementTreeNonlinearEquation derivative2Z = DifferentiationNOrder(derivativeZ, "t", 1);

    double kren;

    Vector3D vectorSpeed = new Vector3D();
    Vector3D vectorAcceleration = new Vector3D();

    double tangentVectorY;

    while (currentTime < piecewiseFunction.function[piecewiseFunction.function.Count - 1].rightBoundaryRange)
    {
        valueVariable["t"] = currentTime;

        vectorSpeed.x = MathCore.CalculateEquationValue(valueVariable, derivativeX);
        vectorSpeed.y = MathCore.CalculateEquationValue(valueVariable, derivativeY);
        vectorSpeed.z = MathCore.CalculateEquationValue(valueVariable, derivativeZ);

        vectorAcceleration.x = MathCore.CalculateEquationValue(valueVariable, derivative2X);
        vectorAcceleration.y = MathCore.CalculateEquationValue(valueVariable, derivative2Y);
        vectorAcceleration.z = MathCore.CalculateEquationValue(valueVariable, derivative2Z);

        tangentVectorY = MathCore.CalculateEquationValue(valueVariable,
        piecewiseFunction.function[currentFunction].trihedronFrene.tangentVector.y);
    }
}

```

```

        kren = -1 * Math.Atan((((vectorSpeed.x * vectorAcceleration.y * 1) + (vectorSpeed.y * vectorAcceleration.z * 0)
+ (vectorAcceleration.x * 0 * vectorSpeed.z) - (vectorSpeed.z * vectorAcceleration.y * 0) - (vectorSpeed.y *
vectorAcceleration.x * 1) - (vectorAcceleration.z * 0 * vectorSpeed.x)) / (9.8 *
vectorSpeed.LengthVector(Vector3D.zeroVector))) * (1 - Math.Pow(tangentVectorY, 2)));

        result.Add(new FlightInformation(
            currentTime,
            new Vector3D(
                MathCore.CalculateEquationValue(valueVariable,
                piecewiseFunction.function[currentFunction].elementCubicSpline.x),
                MathCore.CalculateEquationValue(valueVariable,
                piecewiseFunction.function[currentFunction].elementCubicSpline.y),
                MathCore.CalculateEquationValue(valueVariable,
                piecewiseFunction.function[currentFunction].elementCubicSpline.z)
            ),
            vectorSpeed,
            MathCore.ArcTangent2R(
                MathCore.CalculateEquationValue(valueVariable,
                piecewiseFunction.function[currentFunction].trihedronFrene.tangentVector.x),
                MathCore.CalculateEquationValue(valueVariable,
                piecewiseFunction.function[currentFunction].trihedronFrene.tangentVector.z)
            ),
            kren,
            Math.Asin(tangentVectorY)
        )
    );

    currentTime += stepTime;

    if (currentTime > piecewiseFunction.function[currentFunction].rightBoundaryRange)
    {
        currentFunction += 1;

        if(currentFunction < piecewiseFunction.function.Count)
        {
            derivativeX = DifferentiationNOrder(piecewiseFunction.function[currentFunction].elementCubicSpline.x,
            "t", 1);
            derivativeY = DifferentiationNOrder(piecewiseFunction.function[currentFunction].elementCubicSpline.y,
            "t", 1);
            derivativeZ = DifferentiationNOrder(piecewiseFunction.function[currentFunction].elementCubicSpline.z,
            "t", 1);

            derivative2X = DifferentiationNOrder(derivativeX, "t", 1);
            derivative2Y = DifferentiationNOrder(derivativeY, "t", 1);
            derivative2Z = DifferentiationNOrder(derivativeZ, "t", 1);
        }
    }
}

return result;
}

/// <summary>
/// Создание кусочной функции
/// </summary>
/// <param name="points">
/// Список опорных точек
/// </param>
/// <param name="maxSpeed">
/// Максимальная скорость км/ч
/// </param>

```

```

/// <returns>
/// Кусочная функция
/// </returns>
public static PiecewiseFunction CreatePiecewiseFunctionPosition(List<Vector3D> points, double maxSpeed)
{
    PiecewiseFunction result = new PiecewiseFunction();

    ElementTreeNonlinearEquation[] arrayCubicSpline;

    double[,] arrayPoint = new double[points.Count, 2];

    double[] timeArrival = new double[points.Count];

    timeArrival[0] = 0;

    for (int i = 1; i < points.Count; i++)
    {
        timeArrival[i] = timeArrival[i - 1] + points[i - 1].LengthVector(points[i]);
    }

    //x

    for (int i = 0; i < points.Count; i++)
    {
        arrayPoint[i, 0] = timeArrival[i];
        arrayPoint[i, 1] = points[i].x;
    }

    arrayCubicSpline = CreateCubicSpline(arrayPoint);

    for (int i = 0; i < points.Count - 1; i++)
    {
        result.function.Add(new ElementPiecewiseFunction(timeArrival[i], timeArrival[i + 1], new
ParametricRepresentation(arrayCubicSpline[i], null, null), null));
    }

    //y

    for (int i = 0; i < points.Count; i++)
    {
        arrayPoint[i, 0] = timeArrival[i];
        arrayPoint[i, 1] = points[i].y;
    }

    arrayCubicSpline = CreateCubicSpline(arrayPoint);

    for (int i = 0; i < points.Count - 1; i++)
    {
        result.function[i].elementCubicSpline.y = arrayCubicSpline[i];
    }

    //z

    for (int i = 0; i < points.Count; i++)
    {
        arrayPoint[i, 0] = timeArrival[i];
        arrayPoint[i, 1] = points[i].z;
    }

    arrayCubicSpline = CreateCubicSpline(arrayPoint);

    for (int i = 0; i < points.Count - 1; i++)

```

```

{
    result.function[i].elementCubicSpline.z = arrayCubicSpline[i];

    result.function[i].trihedronFrene = CreateTrihedronFrene(result.function[i].elementCubicSpline);
}

// корректировка по скорости

int currentFunction = 0;
double currentTime = result.function[0].leftBoundaryRange;
Dictionary<String, double> valueVariable = new Dictionary<string, double>();
valueVariable.Add("t", currentTime);
double stepTime = 0.04;
double MAX_SPEED = 0.000277777777778 * maxSpeed;
double absoluteMaximumSpeed = Double.MinValue;

ElementTreeNonlinearEquation derivativeX =
DifferentiationNOrder(result.function[currentFunction].elementCubicSpline.x, "t", 1);
ElementTreeNonlinearEquation derivativeY =
DifferentiationNOrder(result.function[currentFunction].elementCubicSpline.y, "t", 1);
ElementTreeNonlinearEquation derivativeZ =
DifferentiationNOrder(result.function[currentFunction].elementCubicSpline.z, "t", 1);

double speedX;
double speedY;
double speedZ;

while (currentTime < result.function[result.function.Count - 1].rightBoundaryRange)
{
    valueVariable["t"] = currentTime;

    speedX = MathCore.CalculateEquationValue(valueVariable, derivativeX);
    speedY = MathCore.CalculateEquationValue(valueVariable, derivativeY);
    speedZ = MathCore.CalculateEquationValue(valueVariable, derivativeZ);

    absoluteMaximumSpeed = Math.Max(absoluteMaximumSpeed, Math.Max(Math.Abs(speedX),
Math.Max(Math.Abs(speedY), Math.Abs(speedZ))));

    currentTime += stepTime;

    if (currentTime > result.function[currentFunction].rightBoundaryRange)
    {
        currentFunction += 1;

        if (currentFunction < result.function.Count)
        {
            derivativeX = DifferentiationNOrder(result.function[currentFunction].elementCubicSpline.x, "t", 1);
            derivativeY = DifferentiationNOrder(result.function[currentFunction].elementCubicSpline.y, "t", 1);
            derivativeZ = DifferentiationNOrder(result.function[currentFunction].elementCubicSpline.z, "t", 1);
        }
    }
}

if (absoluteMaximumSpeed < MAX_SPEED)
{
    for (int i = 0; i < points.Count - 1; i++)
    {
        result.function[i].trihedronFrene = CreateTrihedronFrene(result.function[i].elementCubicSpline);
    }

    return result;
}

```

```

    }

    result.function.Clear();

    timeArrival[0] = 0;

    for (int i = 1; i < points.Count; i++)
    {
        timeArrival[i] = timeArrival[i - 1] + points[i - 1].LengthVector(points[i]) *
RoundingExcess(absoluteMaximumSpeed / MAX_SPEED, 0);
    }

    //x

    for (int i = 0; i < points.Count; i++)
    {
        arrayPoint[i, 0] = timeArrival[i];
        arrayPoint[i, 1] = points[i].x;
    }

    arrayCubicSpline = CreateCubicSpline(arrayPoint);

    for (int i = 0; i < points.Count - 1; i++)
    {
        result.function.Add(new ElementPiecewiseFunction(timeArrival[i], timeArrival[i + 1], new
ParametricRepresentation(arrayCubicSpline[i], null, null), null));
    }

    //y

    for (int i = 0; i < points.Count; i++)
    {
        arrayPoint[i, 0] = timeArrival[i];
        arrayPoint[i, 1] = points[i].y;
    }

    arrayCubicSpline = CreateCubicSpline(arrayPoint);

    for (int i = 0; i < points.Count - 1; i++)
    {
        result.function[i].elementCubicSpline.y = arrayCubicSpline[i];
    }

    //z

    for (int i = 0; i < points.Count; i++)
    {
        arrayPoint[i, 0] = timeArrival[i];
        arrayPoint[i, 1] = points[i].z;
    }

    arrayCubicSpline = CreateCubicSpline(arrayPoint);

    for (int i = 0; i < points.Count - 1; i++)
    {
        result.function[i].elementCubicSpline.z = arrayCubicSpline[i];

        result.function[i].trihedronFrene = CreateTrihedronFrene(result.function[i].elementCubicSpline);
    }

    return result;
}

```

```

    }
}

```

Файл ParametricRepresentation.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MathCore
{
    /// <summary>
    /// Параметрическое представление
    /// </summary>
    public class ParametricRepresentation
    {
        public ElementTreeNonlinearEquation x;
        public ElementTreeNonlinearEquation y;
        public ElementTreeNonlinearEquation z;

        public ParametricRepresentation()
        {

        }

        public ParametricRepresentation(ElementTreeNonlinearEquation x, ElementTreeNonlinearEquation y,
        ElementTreeNonlinearEquation z)
        {
            this.x = x;
            this.y = y;
            this.z = z;
        }
    }
}

```

Файл PiecewiseFunction.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MathCore
{
    /// <summary>
    /// Кусочная функций
    /// </summary>
    public class PiecewiseFunction
    {
        /// <summary>
        /// Список элементов
        /// </summary>
        public List<ElementPiecewiseFunction> function;

        /// <summary>
        /// Конструктор
        /// </summary>
        public PiecewiseFunction()
        {
            function = new List<ElementPiecewiseFunction>();
        }
    }
}

```

```

    }

}

Файл TrihedronFrene.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MathCore
{
    /// <summary>
    /// Трехгранник Френе
    /// </summary>
    public class TrihedronFrene
    {
        /// <summary>
        /// Касательный вектор
        /// </summary>
        public ParametricRepresentation tangentVector;

        /// <summary>
        /// Вектор нормали
        /// </summary>
        public ParametricRepresentation normalVector;

        /// <summary>
        /// Вектор бинормали
        /// </summary>
        public ParametricRepresentation binormalVector;

        public TrihedronFrene()
        {

        }

        /// <summary>
        /// Конструктор
        /// </summary>
        /// <param name="tangentVector">
        /// Касательный вектор
        /// </param>
        /// <param name="normalVector">
        /// Вектор нормали
        /// </param>
        /// <param name="binormalVector">
        /// Вектор бинормали
        /// </param>
        public TrihedronFrene(ParametricRepresentation tangentVector, ParametricRepresentation normalVector,
ParametricRepresentation binormalVector)
        {
            this.normalVector = normalVector;
            this.tangentVector = tangentVector;
            this.binormalVector = binormalVector;
        }
    }
}

```



```

Файл Vector3D.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;

namespace MathCore
{
    public class Vector3D
    {
        public static Vector3D zeroVector = new Vector3D(0, 0, 0);
        public double x;
        public double y;
        public double z;

        public Vector3D(double x, double y, double z)
        {
            this.x = x;
            this.y = y;
            this.z = z;
        }

        public Vector3D()
        {
        }

        public Vector3 ToVector3Unity()
        {
            return new Vector3((float)x, (float)y, (float)z);
        }
        public double LengthVector(Vector3D bVector)
        {
            return Math.Sqrt(Math.Pow(bVector.x - x, 2) + Math.Pow(bVector.y - y, 2) + Math.Pow(bVector.z - z, 2));
        }
    }
}

```

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

УДОСТОВЕРЯЮЩИЙ ЛИСТ № 150341/п
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

на демонстрационный материал, представленный в электронном виде

Студента Назарова Алексея Ивановича шифр 150341/п
Институт приборостроения, автоматизации и информационных технологий
Кафедра программной инженерии
Направление 09.03.04 Программная инженерия
Направленность (профиль) Промышленная разработка программного обеспечения

Наименование документа: Демонстрационные плакаты к выпускной
квалификационной работе

Документ разработал:

Студент

Назаров А. И.

14.06.19 

Документ согласован:


Руководитель

Фролов А. И.

15.06.19 

Нормоконтроль

Ужаринский А.Ю.

17.06.19 

Документ утвержден:

Зав. кафедрой

Фролов А.И.

20.06.19 

Орел 2019

ИНФОРМАЦИОННО-ПОИСКОВАЯ ХАРАКТЕРИСТИКА
ДОКУМЕНТА НА ЭЛЕКТРОННОМ НОСИТЕЛЕ

Наименование		Характеристики документа на электронном носителе
группы атрибутов	атрибута	
1. Описание документа	Обозначение документа (идентификатор(ы) файла(ов))	\\Презентация Назаров.ppt
	Наименование документа	Демонстрационные плакаты к выпускной квалификационной работе
	Класс документа	ЕСКД
	Вид документа	Оригинал документа на электронном носителе
	Аннотация	Демонстрационный материал, отображающий основные этапы выполнения выпускной квалификационной работы
	Использование документа	Операционная система Windows 10, Microsoft PowerPoint 2010
2. Даты и время	Дата и время копирования документа	20.06.19
	Дата создания документа	10.06.19
	Дата утверждения документа	19.06.19
3. Создатели	Автор	Назаров А. И.
	Изготовитель	Назаров А. И.
4. Внешние ссылки	Ссылки на другие документы	Удостоверяющий лист № 150341/п
5. Защита	Санкционирование	ОГУ имени И.С. Тургенева
	Классификация защиты	По законодательству РФ
6. Характеристики содержания	Объем информации документа	2 572 331 Б

7. Структура документа(ов)	Наименование плаката (слайда) №1	Титульный лист
	Наименование плаката (слайда) №2	Назначение и архитектура Имитатора закабинного пространства
	Наименование плаката (слайда) №3	Функции и входные/выходные данные Подсистемы имитации движения летательного аппарата
	Наименование плаката (слайда) №4	Математическая модель пролета по узловым точкам
	Наименование плаката (слайда) №5	Методика расчета полетных данных
	Наименование плаката (слайда) №6	Структура подсистемы имитации движения Летательного аппарата
	Наименование плаката (слайда) №7	Алгоритм вычисления коэффициентов Кубического сплайна
	Наименование плаката (слайда) №8	Алгоритм определения орт естественного трехгранника
	Наименование плаката (слайда) №9	Структура данных дерева операций
	Наименование плаката (слайда) №10	Диаграмма компонентов подсистемы имитации Движения летательного аппарата
	Наименование плаката (слайда) №11	Пример работы имитатора Пролета летательного аппарата



СПРАВКА

о результатах проверки текстового документа на наличие заимствований

Проверка выполнена в системе
Антиплагиат.ВУЗ

Автор работы	Назаров Алексей Иванович
Подразделение	ИПАИТ, кафедра программной инженерии
Тип работы	Выпускная квалификационная работа
Название работы	Сборка 3.0.docx
Название файла	Сборка 3.0.docx
Процент заимствования	5,82%
Процент цитирования	1,24%
Процент оригинальности	92,95%
Дата проверки	12:00:28 14 июня 2019г.
Модули поиска	Модуль поиска ИПС "Адилет"; Модуль выделения библиографических записей; Сводная коллекция ЭБС; Коллекция РГБ; Цитирование; Модуль поиска переводных заимствований; Коллекция eLIBRARY.RU; Коллекция ГАРАНТ; Модуль поиска Интернет; Коллекция Медицина; Модуль поиска перефразирований eLIBRARY.RU; Модуль поиска перефразирований Интернет; Коллекция Патенты; Модуль поиска общеупотребительных выражений; Модуль поиска "ФГБОУ ВО ОГУ им. И.С.Тургенева"; Кольцо вузов
Работу проверил	Ужаринский Антон Юрьевич ФИО проверяющего

Дата подписи

14.06.19


Подпись проверяющего

Чтобы убедиться
в подлинности справки,
используйте QR-код, который
содержит ссылку на отчет.



Ответ на вопрос, является ли обнаруженное заимствование
корректным, система оставляет на усмотрение проверяющего.
Предоставленная информация не подлежит использованию
в коммерческих целях.