# Частное учреждение образования

# «Колледж бизнеса и права»

# Допущен к защите

Заместитель директора

по учебной работе

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И.В.Малафей

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019

## Программа эмуляции полета ракеты

## Пояснительная записка

# ДП Т.594048.401

Председатель цикловой комиссии ( Т.Г. Багласова )

Руководитель проекта ( О.В. Гальцова )

Консультант по экономической части ( Д.И. Кошепарова )

Консультант по охране труда ( Н.К. Фоменко )

Учащийся ( А.С. Ковалёв )

Рецензент ( )

2019

Содержание

[Введение 4](#_Toc9761317)

[1 Объектно-ориентированный анализ и проектирование системы 5](#_Toc9761318)

[1.1 Сущность задачи 5](#_Toc9761319)

[1.2 Проектирование модели 6](#_Toc9761320)

[2 Вычислительная система 10](#_Toc9761321)

[2.1 Требования к аппаратным и операционным ресурсам 10](#_Toc9761322)

[2.2 Инструменты разработки 10](#_Toc9761323)

[3 Проектирование задачи 11](#_Toc9761324)

[3.1 Требования к приложению 11](#_Toc9761325)

[3.2 Концептуальный прототип 11](#_Toc9761326)

[3.3 Организация данных 13](#_Toc9761327)

[3.4 Функции и элементы управления 14](#_Toc9761328)

[3.5 Проектирование справочной системы приложения 19](#_Toc9761329)

[4 Описание программного средства 20](#_Toc9761330)

[4.1 Общие сведения 20](#_Toc9761331)

[4.2 Функциональное назначение 20](#_Toc9761332)

[4.3 Входные и выходные данные 20](#_Toc9761333)

[5 Методика испытаний 21](#_Toc9761334)

[5.1 Технические требования 21](#_Toc9761335)

[5.2 Функциональное тестирование 21](#_Toc9761336)

[6 Применение 30](#_Toc9761337)

[6.1 Назначение программы 30](#_Toc9761338)

[6.2 Условия применения 30](#_Toc9761339)

[6.3 Справочная система 30](#_Toc9761340)

[7 Охрана труда и окружающей среды 35](#_Toc9761341)

[7.1 Правовые, нормативные, социально-экономические и организационные вопросы охраны труда 35](#_Toc9761342)

[7.2 Обеспечение защиты от рентгеновского излучения при эксплуатации ПЭВМ 37](#_Toc9761343)

[7.3 Пожарная безопасность 38](#_Toc9761344)

[7.4 Охрана окружающей среды 39](#_Toc9761345)

[8 Экономический раздел 41](#_Toc9761346)

[8.1 Технико-экономическое обоснование разработки программного средства 41](#_Toc9761347)

[8.2 Составление плана по разработке программного средства 41](#_Toc9761348)

[8.3 Определение цены программного средства 41](#_Toc9761349)

[8.4 Экономическая эффективность разработки 45](#_Toc9761350)

[Заключение 47](#_Toc9761351)

[Список информационных источников 48](#_Toc9761352)

[Приложение А. Текст программы 50](#_Toc9761353)

у

Изм.

Лист

№ докум.

Подпись

Дата

Лист

3

ДП Т.594048.401 ПЗ

Разраб.

Ковалёв А.С.

Провер.

Гальцова О.В.

Т. контр..

Багласова Т.Г.

Н. контр.

Ржеутская Н.В.

Утверд.

Багласова Т.Г.

Программа эмуляции полета ракеты

Лит.

Листов

91

КБП

У

Введение

В современном мире существует большое множество разнообразных программ эмуляции, без которых век информационных технологий не мог бы существовать и прогрессивно развиваться. Современный мир не может обойтись без возможности протестировать и воспроизвести различные процессы, будь то столкновение атомов, течение воды, или ветряные потоки внутри какой-либо системы, и программы эмуляции позволяют реализовать это. Возможность эмуляции необходима для многих областей деятельности человека, будь то разработка мобильных приложений, снос зданий или проектирование ракеты.

Целью разработки программы для эмуляции полета ракеты в пространстве является разработка программы «Maneuver.exe», которая позволит реализовать на программном уровне эмуляцию полета ракеты в соответствии с маневрами в пространстве.

В результате работы над дипломным проектом будет разработано программное средство и составлена пояснительная записка, которая включает в себя восемь разделов, содержащих необходимую и достаточную информацию по использованию данного программного средства [1].

В первом разделе «Объектно-ориентированный анализ и проектирование системы» раскрывается организационная сущность задачи, описывается предметная область и круг задач, которые должны быть автоматизированы. Описывается задача, перечисляются основные функции программы. Строится информационная модель, отражающая сущности задачи, их свойства и взаимосвязи.

Во втором разделе «Вычислительная система» перечисляются требования к аппаратному обеспечению и конфигурации компьютера, проводится характеристика операционной системы, обоснование выбранной среды для разработки приложения.

В третьем разделе «Проектирование задачи» проводится объектно-ориентированный анализ задачи, строится концептуальный прототип системы меню, диалоговых окон и элементов управления, приводится организация данных в контексте выбранных систем управления базами данных, отражаются основные функции программы, и логическая и физическая организация, приводится структура справочной системы.

В четвертом разделе «Описание программного средства» представлены общие сведения о программном средстве и его функциональном назначении, описываются входные и выходные данные.

В пятом разделе «Методика испытаний» описываются требования к техническим средствам для проведения испытаний, требования к характеристикам программы применительно к условиям эксплуатации, требования к информационной и программной совместимости. Представляются результаты функционального тестирования.

Шестой раздел «Применение» предназначен для описания сведений о назначении программного средства и области его применения. В этом разделе приводится структура справочной системы, а также методика ее использования.

Седьмой раздел «Охрана труда и охрана окружающей среды» предназначен для описания сведений о назначении охраны труда и окружающей среды на предприятии и ее необходимости. В этом разделе описываются основные правила и требования, которые необходимо соблюдать во избежание производственного травматизма и несчастных случаев.

В восьмом экономическом разделе приводится технико-экономическое обоснование, составляется план по разработке программного средства, производится расчет стоимости дипломного проекта.

В заключении будет проанализировано созданное программное средство, определена степень соответствия поставленной задачи и выполненной работы.

Приложение А будет содержать текст программы.

В графической части представлены диаграммы вариантов использования, классов, деятельности, последовательности и компонентов.

# 1 Объектно-ориентированный анализ и проектирование системы

## Сущность задачи

Предметной областью решаемой задачи является полет ракеты. Полет ракеты рассматривается как множество траекторий полета снаряда между ключевыми точками, к которым привязываются соответствующие манёвры. Манёвром называется точка, в которой происходит изменение скорости и направления движения (курса) летательного аппарата.

Расчет траектории полета ракеты сводится к построению графиков полета между точками манёвра с учетом параметров ракеты при манёврах.

К сожалению, человеческий фактор не позволяет автоматизировать расчет траектории полета ракеты (так как большими проблемами являются огромная вероятность ошибки из-за огромных объемов данных). Для данных целей на месте прохождения практики, предприятии ОАО «АЛЕВКУРП», до начала прохождения практики была разработана и удачно эксплуатировалась библиотека для вычисления траектории полёта ракеты. Данная библиотека была внедрена для создания данного программного средства. Часть формул, приведенных ниже в этом разделе, была использована в данной библиотеке.

Однако разработка программы, решающей подобный круг задач, только расширит спектр предлагаемых товаров на рынке программного обеспечения.

Исходя из анализа предметной области решаемой задачи, можно выделить следующие задачи, подлежащие автоматизации:

* автоматизация расчета траектории полета ракеты;
* автоматизация воспроизведения полета ракеты;

Существует общее описание расчета скорости полета тела формулой Циолковского [11]. Расчет скорости полета ракеты представлен формулой 1.1.

, 1.1

где V – конечная скорость летательного аппарата;

I – удельный импульс ракетного двигателя;

M1 – начальная масса летательного аппарата;

M2 – конечная масса летательного аппарата.

Удельный импульс ракетного двигателя представлен формулой 1.2.

, 1.2

где I – удельный импульс ракетного двигателя;

P – значение тяги двигателя;

Q – секундный расход массы тела.

Для большинства расчетов в сферической системе координат используются координаты вектора [12]. Расчет координат вектора представлен формулой 1.3.

, 1.3

где – координаты вектора;

– значение координаты «*x*» точки «*A*»;

– значение координаты «*x*» точки «*B*»;

– значение координаты «*y*» точки «*A*»;

– значение координаты «*y*» точки «*B*»;

– значение координаты «*z*» точки «*A*»;

– значение координаты «*z*» точки «*B*».

Так же для манипуляции данными в сферической системе координат используется длина вектора [13]. Расчет длины вектора представлен формулой 1.4.

, 1.4

где – длина вектора;

– координаты вектора;

– координата «*x*» вектора «»;

– координата «*y*» вектора «»;

– координата «*z*» вектора «».

В сферической системе координат используются такие понятия как зенитный и азимутальный углы (*θ* и *φ* соответственно) [14]. Расчет зенитного угла представлен формулой 1.5.

, 1.5

где *θ* – зенитный угол;

– координата «*z*» вектора точки «*B*» вектора *«AB»*;

– длина вектора.

Расчет азимутального угла представлен формулой 1.6.

, 1.6

где – азимутальный угол;

– координата «*y*» вектора точки «*B*» вектора *«AB»*;

– координата «*x*» вектора точки «*B*» вектора *«AB»*.

Аналогом программного средства является программа OpenRocket, но разрабатываемое программное средство предлагает более простой способ взаимодействия и простоту использования.

## 1.2 Проектирование модели

Цель моделирования данных состоит в обеспечении разработчика информационной системы концептуальной схемой базы данных в форме одной модели или нескольких локальных моделей, которые относительно легко могут быть отображены в любую систему баз данных.

Наиболее распространенным средством моделирования данных являются диаграммы «Cущность-связь» (ERD). С их помощью определяются важные для предметной области объекты (сущности), их свойства (атрибуты) и отношения друг с другом (связи). ERD непосредственно используются для проектирования реляционных баз данных. Нотация ERD была впервые введена П. Ченном и получила дальнейшее развитие в работах Баркера. Диаграмма «Cущность-связь» представлена на рисунке 1.

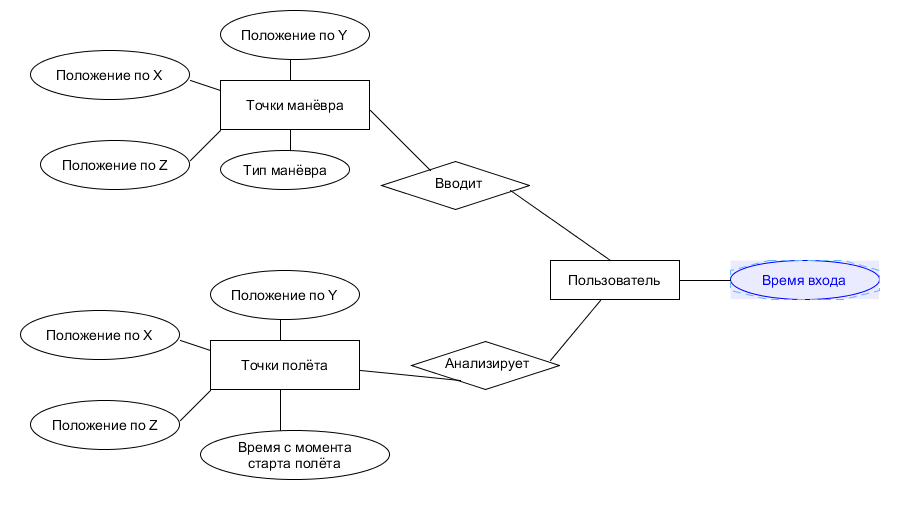


Рисунок 1 – Диаграмма «Сущность-связь»

Исходя из исследования предметной области, можно выделить следующие сущности, необходимые для разработки: «Точки полёта», «Точки манёвра», «Пользователь».

Для сущности «Точки манёвра» атрибутами будут являться:

* положение по X;
* положение по Y;
* положение по Z;
* тип маневра.

Для сущности «Точки полёта» атрибутами будут являться:

* положение по X;
* положение по Y;
* положение по Z;
* время со старта полета.

Для сущности «Пользователь» атрибутом будет являться время входа.

Перечень этих диаграмм представляет собой неотъемлемую часть графической нотации унифицированного языка моделирования (UML), сам процесс объектно-ориентированного программирования (ООП) неразрывно связан с процессом построения этих диаграмм. Совокупность построенных таким образом диаграмм содержит всю информацию, необходимую для реализации проекта сложной системы.

В рамках языка UML все представления о модели сложной системы фиксируются в виде специальных графических конструкций – диаграмм. В терминах языка UML определены следующие виды диаграмм: диаграмма вариантов использования, диаграмма классов, диаграмма деятельности, диаграмма последовательности, диаграмма компонентов.

Суть диаграммы вариантов использования состоит в том, что проектируемая система представляется в виде множества сущностей или актёров, взаимодействующих с системой с помощью, так называемых, вариантов использования.

Варианты использования описывают не только взаимодействия между пользователями и сущностью, но также реакции сущности на получение отдельных сообщений от пользователей и восприятие этих сообщений за пределами сущности. Варианты использования могут включать в себя описание особенностей способов реализации сервиса и различных исключительных ситуаций, таких как корректная обработка ошибок системы. Множество вариантов использования в целом должно определять все возможные стороны ожидаемого поведения системы.

Актёр представляет собой внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует её функциональные возможности для достижения определённых целей или решения частных задач. При этом актёры служат для обозначения согласованного множества ролей, которые могут играть пользователи в процессе взаимодействия с проектируемой системой. Каждый актёр может рассматриваться как некоторая отдельная роль относительно конкретного варианта использования.

Данная программа имеет следующие основные функции:

– осуществить возможность работы со сценой;

– осуществить отображение информации;

– создать отчет.

К вспомогательным функциям, расширяющим возможности системы, относятся следующие функции:

– предоставить информацию о летающем теле;

– предоставить справочную систему;

– предоставить информацию о манёврах.

Диаграмма вариантов использования представлена в графической части на листе 1 [15].

При моделировании поведения проектируемой или анализируемой системы возникает необходимость детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, переход в следующее состояние срабатывает только при завершении этой операции. Графически диаграмма деятельности представляется в форме графа, вершинами которого являются состояния действия, а дугами - переходы от одного состояния действия к другому. Основная цель использования диаграмм деятельности - визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. Диаграмма деятельности представлена в графической части на листе 2 [16].

Для моделирования взаимодействия объектов в UML используются соответствующие диаграммы взаимодействия. Если рассматривать взаимодействия объектов во времени, тогда для представления временных особенностей передачи и приема сообщений между объектами используется диаграмма последовательности.

Временной аспект поведения имеет существенное значение при моделировании синхронных процессов, описывающих взаимодействия объектов. Именно для этой цели и используются диаграммы последовательности, в которых ключевым моментом является динамика взаимодействия объектов во времени. При этом диаграмма последовательности имеет как бы два измерения: одно - слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии; второе - вертикальная временная ось, направленная сверху вниз, на которой начальному моменту времени соответствует самая верхняя часть диаграммы. Диаграмма последовательности для проектируемой системы представлена в графической части на листе 3[17].

Рассмотренные ранее диаграммы отражали концептуальные аспекты построения модели системы и относились к логическому уровню представления. Особенность логического представления заключается в том, что оно оперирует понятиями, которые не имеют самостоятельного материального воплощения. Другими словами, различные элементы логического представления, такие как классы, ассоциации, состояния, сообщения, не существуют материально или физически. Они лишь отражают наше понимание структуры физической системы или аспекты ее поведения.

Основное назначение логического представления состоит в анализе структурных и функциональных отношений между элементами модели системы. Однако для создания конкретной физической системы необходимо, некоторым образом, реализовать все элементы логического представления в конкретные материальные сущности. Для описания таких реальных сущностей предназначен другой аспект модельного представления, а именно физическое представление модели.

Диаграмма компонентов описывает объекты реального мира - компоненты программного обеспечения. Эта диаграмма позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами. Вид диаграммы компонентов для данной проектируемой системы представлен в графической части на листе 4 [18]. Она содержит следующие компоненты:

– программные компоненты: «.cs», «.xaml», «.resx»;

– файл проекта Visual Studio 2019: «Maneuver.sln»;

– файлы данных «\*.json» и «\*.xml»;

– файл математической библиотеки вычисления полёта ракеты «ManeuverMath.dll»;

– файл справочной системы «Help.chm».

Диаграмма классов демонстрирует общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей между ними. Данная диаграмма описывает общую иерархию объектов системы, их взаимоотношений и функций. Вид диаграммы компонентов для данной проектируемой системы представлен в графической части на листе 5 [19].

# 2 Вычислительная система

## 2.1 Требования к аппаратным и операционным ресурсам

Конфигурация компьютера, на котором будет разрабатываться программное приложение, состоит из следующих компонентов:

* операционная система Windows 10 1809 (x64);
* процессор Intel Core i7 7700HQ;
* оперативная память 8 Гбайт типа DDR4;
* твердотельный накопитель (SSD) 250 Гбайт;
* видеокарта Nvidia GTX 1050 ti.

Для удобной работы с программой необходимо наличие мыши и клавиатуры.

## 2.2 Инструменты разработки

Инструментами разработки будут являться:

* операционная система Windows 10;
* среда программирования Visual Studio 2019;
* язык программирования C#;
* приложение создания справочной системы HelpNDoc;
* приложение создания диаграмм Umlet;
* система создания инсталляционного пакета InstallShield.

Программное приложение будет разрабатываться и тестироваться под управлением операционной системы Windows 10. Данная версия операционной системы Windows сочетает в себе преимущества всех прошлых версий, расширяя средства эффективной разработки [20].

Выбор Integrated Development Environment (IDE) Visual Studio 2019 обоснован ее функциональностью, простотой проектирования приложений и удобством работы с программным кодом. Также данная IDE предлагает уже встроенные инструменты взаимодействия с базами данных различных компаний, будь то Oracle, Microsoft и так далее [21].

C#, находясь в среде Microsoft Windows, имеет простой интерфейс взаимодействия с данными любого формата [22].

Приложение создания справочной системы HelpNDoc имеет простой и понятный интерфейс, позволяет создавать справочные системы высокого уровня сложности и включать такую функциональность в справочную систему, как поиск и индексирование [23].

Umlet — утилита для разработки моделей различного рода процессов и проектирования приложений на основе языка UML. С ее помощью можно быстро строить UML диаграммы, экспортировать диаграммы в форматы eps, pdf, jpg, svg, сохранять их в буфере обмена и разрабатывать новые объекты UML [24].

InstallShield — система создания инсталляторов и пакетов программного обеспечения для Microsoft Windows и Linux с закрытыми исходными кодами. Программный инструмент используется главным образом для установки программного обеспечения для Microsoft Windows, как на десктопные, так и серверные платформы, а также может быть использован для управления приложениями на различных портативных и мобильных устройствах [25].

# 3 Проектирование задачи

## 3.1 Требования к приложению

Данное приложение не требует никаких специальных средств защиты либо ограничений прав доступа к данным по причине того, что оно будет использоваться как закрытое программное средство внутри организации ОАО «АЛЕВКУРП». Данное программное средство будет использоваться в ОАО «АЛЕВКУРП» как один из модулей проверки корректной работы ракет, или ракетных модулей.

Разрабатываемое приложение должно иметь понятный и удобный в использовании интерфейс, чтобы взаимодействие между программой и пользователям было максимально упрощено. Добавление, удаление и изменение будет происходить с помощью табличной формы. Взаимодействие с 3D-пространством будет осуществляться внутри окна в виде отдельного рабочего пространства с помощью компьютерной мыши.

Все входные данные должны проверяться на ошибки. При совершении пользователем ошибки, ему будет предоставлено диалоговое окно с разъяснением ошибки.

Кроме этого, формы в приложении должны быть выполнены в едином стиле, сдержанной цветовой гамме, иметь стандартные элементы управления данными, не должны быть перегружены информацией, и в свое время понятны простому пользователю. Так же управление и манипулирование данными внутри 3D-пространства должно быть максимально простым и логичным для пользователя.

При конструировании форм в необходимых случаях нужно предусмотреть возможность защиты данных от изменений, например, при просмотре информации, установить ограничения на корректировку записей, в 3D-пространстве ограничения на изменения состояний объекта.

В создаваемом программном средстве должна быть справочная система, в которой будут предусмотрены все возможные способы работы системы. Справочная система должна содержать ответы на вопросы, которые могут возникнуть у пользователя во время эксплуатации.

## 3.2 Концептуальный прототип

Концептуальный прототип состоит из описания внешнего пользовательского интерфейса, а именно, элементов управления.

При создании данного приложения важную роль играют формы, так как они являются основным диалоговым средством работы пользователя.

При проектировании концептуального прототипа предполагается, что при загрузке программы первой будет загружаться форма «MainWindow». На ней будут отображаться элементы управления «Button» для выполнения определённых действий, таких как редактирование, удаление, добавление, а также «DataGrid» для отображения данных в табличном формате.

Практически на всех формах используется компонент Label или TextBlock для осуществления соответствующих надписей на них и создания ссылок на соответствующие страницы.

Макет окна «MainWindow» представлен на рисунке 2.

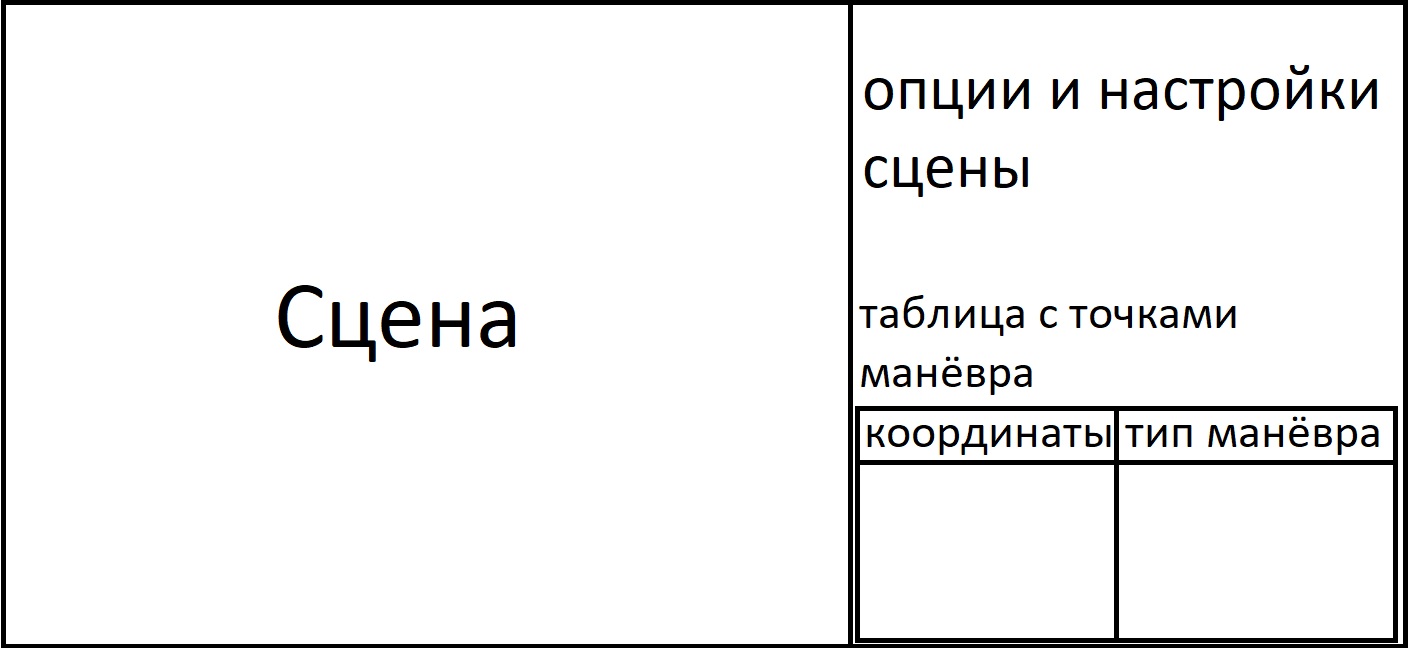


Рисунок 2 – Макет окна «MainWindow»

На главном окне располагаются сцены, меню по работе со сценой, опции и настройки сцены.

На окне «ManeuverView» располагается сцена, кнопки работы со сценой и экспорта сцены. Макет окна «ManeuverView» представлен на рисунке 3.

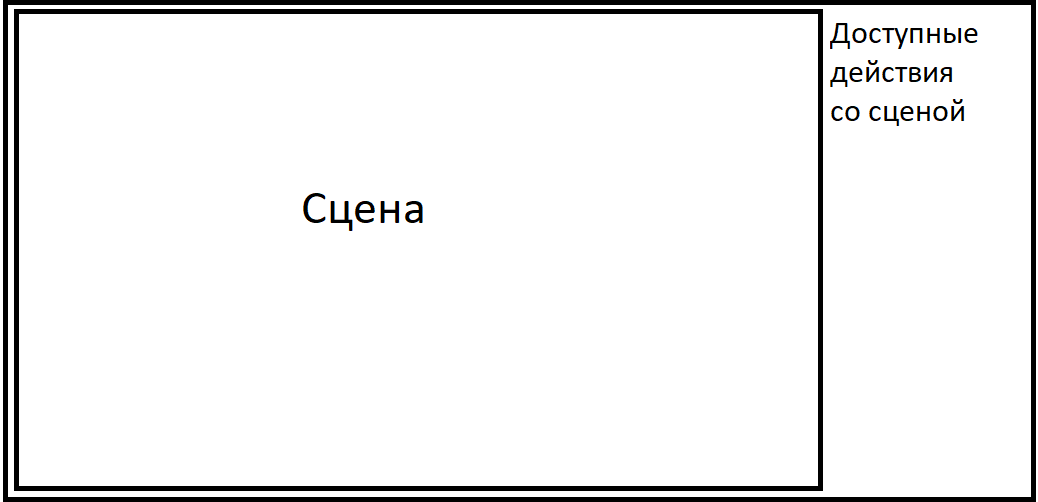


Рисунок 3 – Макет окна «ManeuverView»

На окне «ManeuverView» редактирования данных присутствуют кнопки для работы со сценой.

## 3.3 Организация данных

Организация данных подразумевает создание модели данных, главными элементами которой являются сущности.

Объектно-ориентированная модель основана на понятии объекта сущности, представлением которого является таблица. В объектно-ориентированной модели данных любая сущность реального мира представляется всего одним понятием — объектом. С объектом ассоциируется состояние и поведение. Состояние объекта определяется значениями его свойств — атрибутов. Значениями свойства могут являться примитивные значения (такие, как строки или целые числа) и непримитивные объекты. Непримитивный объект, в свою очередь, состоит из набора свойств. Следовательно, объекты можно рекурсивно определять в терминах других объектов. Поведение объекта определяется с помощью методов, которые оперируют над состоянием объекта.

Структура базы данных разрабатываемого программного средства включает две таблицы.

Структура данных таблиц, и их краткое описание приводится в таблицах 1-2.

Таблица «maneuver points» хранит информацию о точках манёвра. Структура приведена в таблице 1.

Таблица 1 – Структура таблицы «maneuver points».

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер поля, байт | Описание поля |
| x | float | 4 | Положение точки в пространстве по «x» |
| y | float | 4 | Положение точки в пространстве по «y» |
| z | float | 4 | Положение точки в пространстве по «z» |
| type | text | varchar(max) | Тип манёвра |

Таблица «flight points» хранит информацию о точках полёта. Структура приведена в   
таблице 2.

Таблица 2 – Структура таблицы «flight points».

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер поля, байт | Описание поля |
| x | float | 4 | Положение точки в пространстве по «x» |
| y | float | 4 | Положение точки в пространстве по «y» |
| z | float | 4 | Положение точки в пространстве по «z» |
| time | decimal | 8 | Время с момента полета в миллисекундах |

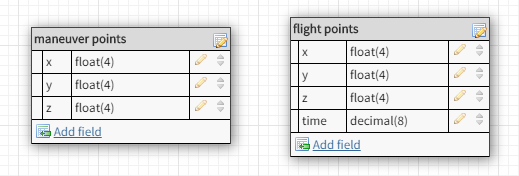


Рисунок 4 – Схема БД

## 3.4 Функции и элементы управления

Основными функциями данного проекта являются эмуляция полёта ракеты и редактирование сцены полёта.

Для работы со сценой используются триггеры нажатия, перетаскивания и отпускания мыши на 3D-пространстве. Код триггеров нажатия, перетаскивания и отпускания мыши представлен ниже.

private void OpenGLView\_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)

{

switch (e.Button)

{

case System.Windows.Forms.MouseButtons.Left:

case System.Windows.Forms.MouseButtons.Right:

case System.Windows.Forms.MouseButtons.Middle:

ProgramStateHelper.IsMouseKeyPressed = true;

ProgramStateHelper.LastPressedKey = e.Button;

ProgramStateHelper.LastMouseCursorPressedPosition = new Point(e.X, e.Y);

break;

}

}

private void OpenGLView\_MouseUp(object sender, System.Windows.Forms.MouseEventArgs e)

{

// не нужно реагировать на случай, когда какой-то дурак решил перетаскивание с другого UI-элемента сделать

if (ProgramStateHelper.LastPressedKey != e.Button) return;

switch (e.Button)

{

case System.Windows.Forms.MouseButtons.Left:

// если хотя бы одна из ранее зафиксированных координат мыши не совпадает с текущим, то и не надо производить выделение

if (ProgramStateHelper.LastMouseCursorPressedPosition.X != e.X || ProgramStateHelper.LastMouseCursorPressedPosition.Y != e.Y)

return;

// производим выделение объектов (hit test)

var selected = Scene.DoHitTest(e.X, e.Y, this.OpenGLView);

switch(selected)

{

case ManeuverPoint point:

CurrentSelectedObjectPropertyGrid.SelectedObject = point;

Scene.Camera.Target.Set(point.Location.X, point.Location.Y, point.Location.Z);

OpenGLView.Refresh();

break;

default:

CurrentSelectedObjectPropertyGrid.SelectedObject = null;

break;

}

ProgramStateHelper.IsMouseKeyPressed = false;

break;

case System.Windows.Forms.MouseButtons.Middle:

ProgramStateHelper.LastPressedKey = System.Windows.Forms.MouseButtons.None;

ProgramStateHelper.IsMouseKeyPressed = false;

break;

}

}

private void OpenGLView\_MouseMove(object sender, System.Windows.Forms.MouseEventArgs e)

{

// если клавиша мышы не задействована, то и нечего нам тут делать

if (!ProgramStateHelper.IsMouseKeyPressed) return;

// объяснение

// если нажаты alt, ctrl и колесико мышки, то выполняется вращение самого target

// также, если нажаты только alt и колесико мышки, то выполняется вокруг текущего target

if (Keyboard.IsKeyDown(Key.LeftAlt) && Keyboard.IsKeyDown(Key.LeftCtrl) && ProgramStateHelper.LastPressedKey == System.Windows.Forms.MouseButtons.Middle)

{

// повторение: этот блок выполняет вращение самого target

}

else if (Keyboard.IsKeyDown(Key.LeftAlt) && ProgramStateHelper.LastPressedKey == System.Windows.Forms.MouseButtons.Middle)

{

// повторение: этот блок выполняет вращение вокруг target

// получаем предыдущие координаты мыши

double last\_mouse\_x = ProgramStateHelper.LastMouseCursorPressedPosition.X;

double last\_mouse\_y = ProgramStateHelper.LastMouseCursorPressedPosition.Y;

// получаем текущее положение мыши

var current\_mouse\_x = e.X;

var current\_mouse\_y = e.Y;

/\* вычисления + изменение камеры \*/

// инициализация относительного вектора (camera.position -> cameta.target) и вычисление длинны относительного вектора

Point3DFloat vertex = new Point3DFloat()

{

X = Scene.Camera.Position.X - Scene.Camera.Target.X,

Y = Scene.Camera.Position.Y - Scene.Camera.Target.Y,

Z = Scene.Camera.Position.Z - Scene.Camera.Target.Z

};

vertex.X += Scene.Camera.Target.X;

vertex.Y += Scene.Camera.Target.Y;

vertex.Z += Scene.Camera.Target.Z;

double radius = Math.Sqrt(vertex.X \* vertex.X + vertex.Y \* vertex.Y + vertex.Z \* vertex.Z);

// получение текущих theta и phi (подробнее: https://en.wikipedia.org/wiki/Spherical\_coordinate\_system)

double theta = Math.Acos(Scene.Camera.Position.Z / radius); // угол наклона по оси z (упрощенное объяснение)

double phi = Math.Atan2(Scene.Camera.Position.Y, Scene.Camera.Position.X); // угол наклона по оси xy (упрощенное объяснение)

// изменение theta и phi

theta -= (float)(current\_mouse\_y - last\_mouse\_y) \* ProgramStateHelper.MouseKoef;

phi -= (float)(current\_mouse\_x - last\_mouse\_x) \* ProgramStateHelper.MouseKoef;

// вычисление новых позиций x, y, z

double newX = SphericalSystemHelper.Calculate3DXPointPosition(radius, theta, phi);

double newY = SphericalSystemHelper.Calculate3DYPointPosition(radius, theta, phi);

double newZ = SphericalSystemHelper.Calculate3DZPointPosition(radius, theta);

Scene.Camera.Position.Set((float)newX, (float)newY, (float)newZ);

// апдейт текущего положения мыши

ProgramStateHelper.LastMouseCursorPressedPosition = new Point(e.X, e.Y);

OpenGLView.Refresh();

}

else if (ProgramStateHelper.LastPressedKey == System.Windows.Forms.MouseButtons.Middle)

{

// этот блок выполняет перемещение камеры и цели камеры (target-а) перпендикулярно

}

}

Для осуществления отображения информации используется триггер, выполняющий отрисовку сцены. Код триггера представлен ниже.

private void OpenGLView\_DrawScene(object sender, SharpGL.RenderEventArgs args)

{

// draw scene

Scene.Render();

}

Создание отчета выполняется путем сохранения файла в указанную директорию с указанным именем файла пользователем. Код функции сохранения отчета представлен ниже.

private void SavePlanClick(object sender, RoutedEventArgs e)

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

var dialogResult = saveFileDialog.ShowDialog(this);

if (dialogResult != true)

return;

string filePath = saveFileDialog.FileName;

string ext = Path.GetExtension(filePath);

string fileContent = JsonConvert.SerializeObject(FlightPoints, Newtonsoft.Json.Formatting.Indented, new JsonSerializerSettings() { TypeNameHandling = TypeNameHandling.Auto });

switch (ext)

{

case ".json":

File.WriteAllText(filePath, fileContent);

break;

case ".xml":

var xmlDoc = JsonConvert.DeserializeXmlNode("{\"ManeuverPlanPoint\":" + fileContent + "}", "root");

xmlDoc.Save(filePath);

break;

default:

break;

}

}

При клике мышью в области сцены будет срабатывать триггер, выполняющий выделение объекта, на котором был произведен клик. Код триггера представлен ниже.

private void OpenGLView\_MouseUp(object sender, System.Windows.Forms.MouseEventArgs e)

{

// не нужно реагировать на случай, когда пользователь решил перетаскивание с другого UI-элемента сделать

if (ProgramStateHelper.LastPressedKey != e.Button) return;

switch (e.Button)

{

case System.Windows.Forms.MouseButtons.Left:

// если хотя бы одна из ранее зафиксированных коордитнат мыши не совпадает с текущим, то и не надо производить выделение

if (ProgramStateHelper.LastMouseCursorPressedPosition.X != e.X || ProgramStateHelper.LastMouseCursorPressedPosition.Y != e.Y)

return;

// производим выделение объектов (hit test)

var selected = Scene.DoHitTest(e.X, e.Y, this.OpenGLView);

switch(selected)

{

case ManeuverPoint point:

CurrentSelectedObjectPropertyGrid.SelectedObject = point;

Scene.Camera.Target. Set(point.Location.X,point.Location.Y,point.Location.Z);

OpenGLView.Refresh();

break;

default:

CurrentSelectedObjectPropertyGrid.SelectedObject = null;

break;

ProgramStateHelper.IsMouseKeyPressed = false;

break;

case System.Windows.Forms.MouseButtons.Middle:

ProgramStateHelper.LastPressedKey = System.Windows.Forms.MouseButtons.None;

ProgramStateHelper.IsMouseKeyPressed = false;

break;

}

}

При изменении состояния объекта в сцене или состояния сцены будет производится рендер сцены. Код процедуры представлен ниже.

public void Render()

{

var gl = OpenGLContext;

// производится очистка экрана (очистка буфферов глубины и цвета)

gl.Clear(OpenGL.GL\_COLOR\_BUFFER\_BIT | OpenGL.GL\_DEPTH\_BUFFER\_BIT);

gl.MatrixMode(SharpGL.Enumerations.MatrixMode.Projection);

gl.LoadIdentity();

// получение размера вьюпорта

int[] viewport\_size = new int[4];

gl.GetInteger(OpenGL.GL\_VIEWPORT, viewport\_size);

// настройка отображения матрицы (объектов на экране) с заданием угла обзора, дальностью отрисовки

// (значения буфера глубины)

float x = (float)viewport\_size[2] / viewport\_size[3];

gl.Perspective(45, x, 0.1, double.MaxValue);

// задание точки отображения (наблюдатель, цель)

Camera.LookAt(gl);

// отрисовка всех объектов в сцене

foreach (SceneElement element in SceneElements)

element.Render(gl);

gl.Flush();

}

При выделении объекта в сцене мышью будет производиться функция отрисовки объектов с заданным цветом. Код процедуры представлен ниже.

public void ColorPickRender(OpenGL context, byte[] rgba, Point3DFloat location)

{

if (ViewObject is ISelectedPrimitive selectModeView)

selectModeView.ColorPickDraw(context, rgba, Location);

}

Полный текст программы представлен в приложении А

## 3.5 Проектирование справочной системы приложения

В справочной системе будет содержаться подробное описание работы программы с описанием необходимых для корректной работы программы действий. Данная система позволит пользователю получить конкретную информацию по интересующему вопросу

Справочная система будет выполнена в виде скомпилированного chm-файла и разработана в пакете «HelpNDoc».[23]

Справка имеет следующие разделы:

* «О программе»;
* «Системные требования»;
* «Руководство пользователя»;
* «Возможные вопросы и их решение».

# 4 Описание программного средства

## 4.1 Общие сведения

Программные средство «Эмуляция полета ракеты» предназначено снизить затраты времени на расчет траектории полета и формирование отчетов по полётам.

Программное средство создано в среде разработки Visual Studio 2019 на языке программирования C#. В качестве системы разработки был выбран WPF (Windows Presentation Foundation).Программа может работать на операционных системах Windows 7, 8, 8.1, 10. Программа не требовательна к системным ресурсам. Для успешного использования программы на компьютере должно быть свободного места на диске не меньше 6 Мбайт, оперативная память должна быть от 1 Гбайт.

Для работы данного программного средства необходима предустановка следующих программных средств:

* Microsoft Microsoft Visual C++ Redistributable 2017;
* Microsoft NET Framework v4.7.2 или выше.

Microsoft Net Framework - программная платформа, выпущенная компанией Microsoft в 2002 году. Основой платформы является общеязыковая среда исполнения Common Language Runtime, которая подходит для разных языков программирования. Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду.

Требуется предварительная инсталляция программы, процесс инсталляции можно запустить с помощью исполняемого файла «Setup.exe», далее необходимо следовать инструкциям установщика.

## 4.2 Функциональное назначение

Назначение программного средство «Maneuver» является эмуляция полета ракеты. В данной программе классом решаемых задач являются:

– возможность добавлять, изменять и удалять точки манёвров;

– возможность запускать эмуляцию полёта;

– возможность создавать отчёты.

## 4.3 Входные и выходные данные

Входной является информация, которую пользователь вносит в файл сцены, заполняя необходимые поля таблицы «maneuver points».

Для разрабатываемого программного средства входной будет служить информация о точке манёвра (координаты, тип манёвра).

Входными данными будут являться данные, вводимые пользователем при добавлении новой информации о точках манёвра в программе.

Выходными данными являются файлы формата xml и json, содержащие в себе информацию как о полёте, так и о точках манёвра.

# 5 Методика испытаний

## 5.1 Технические требования

Для полноценного функционирования разработанного программного приложения, качественного выполнения всех поставленных задач, требуется ПК следующей конфигурации:

* + процессор AMD Athlon(tm) X4 760 Quad Core Processor 3.80 МГц;
  + оперативная память 1 Гбайт;
  + монитор;
  + OpenGL 4.0 (и выше) совместимый графический ускоритель;
  + 6 Мбайт свободного места на винчестере;
  + мышь, клавиатура.

## 5.2 Функциональное тестирование

В процессе написания программного средства необходимо производить тестирование на правильность работы приложения. Одной из основных задач тестирования является устранение ошибок, происходящих при вводе данных.

Функциональное тестирование – это тестирование функций приложения на соответствие требованиям. Оценка производится в соответствии с ожидаемыми и полученными результатами (на основании функциональной спецификации), при условии, что функции отрабатывали на различных значениях.

Тестирование программы будет производиться последовательно, переходя из одной части программы в другую. Во время теста будут проверяться все действия с программой, навигация пунктам меню, которые может произвести пользователь. После чего, все собранные и найденные ошибки будут исправлены.

Главное окно программы представлено на рисунке 5.

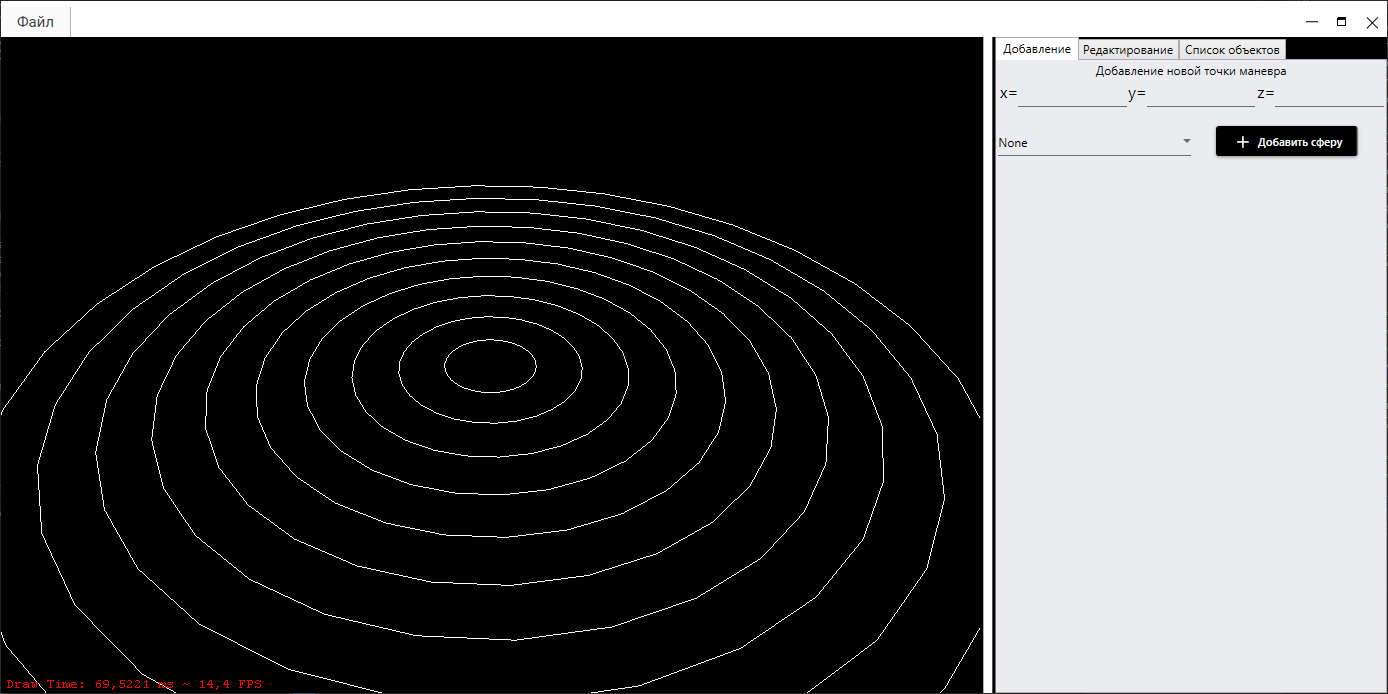


Рисунок 5 – Главное окно

Таблица 3 – Test-case для выбора объекта

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Тип теста | Описание | Шаги воспроизведения | Результат |
| 001 | Позитивный | Выбор объекта на сцене | 1. Ввести данные новой точки маневра:  * Координата «x»; * Координата «y»; * Координата «z»; * Тип маневра.  1. Нажать кнопку добавить сферу;   3. Нажать на появившуюся сферу. | Ожидаемый результат: в меню «Редактирование» появились параметры выделенной точки |
| Фактический результат: результат соответствует ожидаемому. Все условия соблюдены. Результат добавления данных представлен на рисунках 6, 7 и 8. |

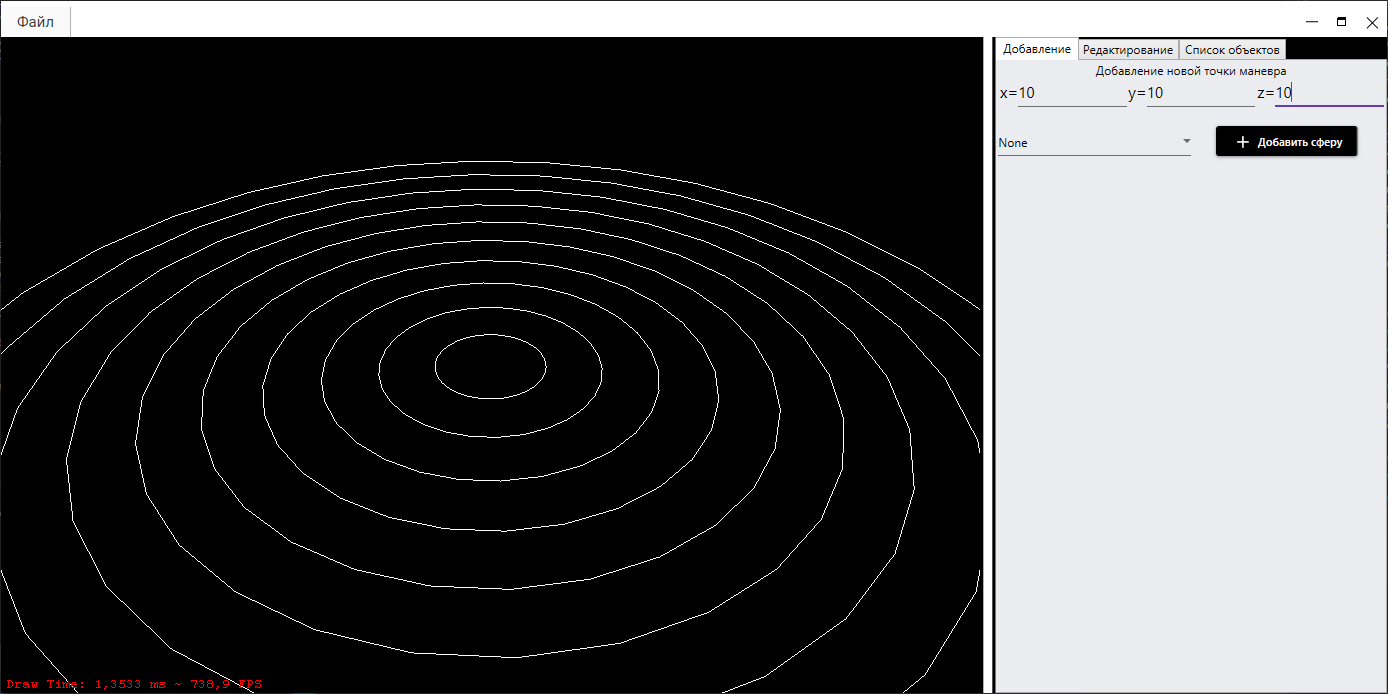


Рисунок 6 – Тест выбора объекта

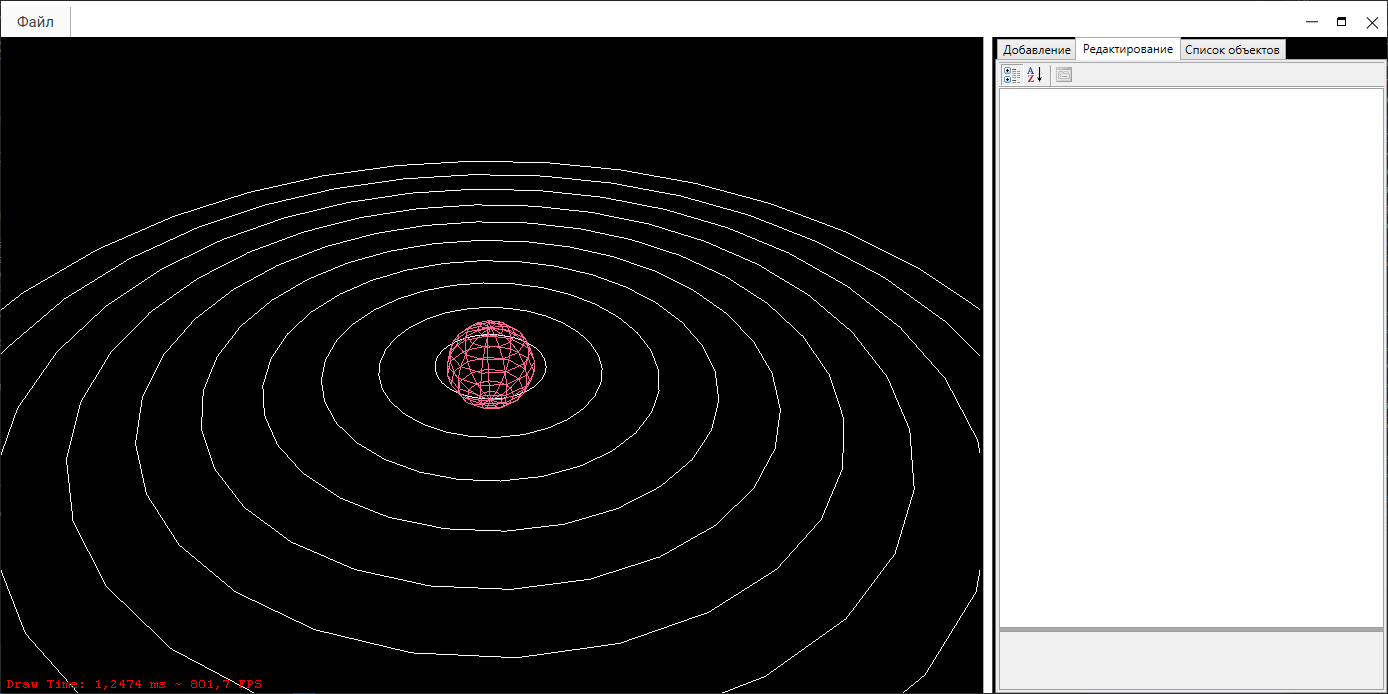


Рисунок 7 – Тест выбора объекта

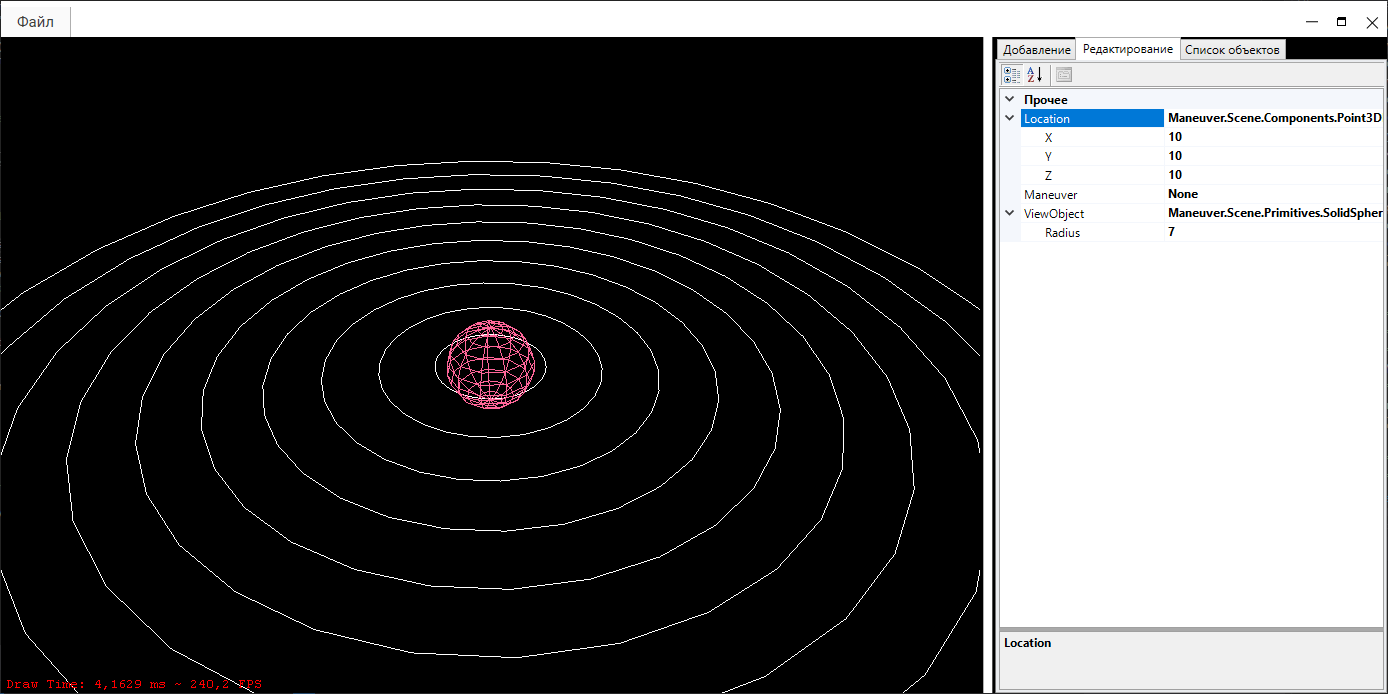


Рисунок 8 – Результат выбора объекта

Таблица 4 – Test-case для изменения радиуса сферы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Тип теста | Описание | Шаги воспроизведения | Результат |
| 002 | Позитивный | Изменение радиуса сферы | 1. Нажать на сферу; 2. Перейти в меню «Редактирование»; 3. Выбрать параметр «Radius» в свойстве «ViewObject»; 4. Изменить значение. | Ожидаемый результат: радиус выбранной сферы изменен |
| Фактический результат: результат соответствует ожидаемому. Все условия соблюдены. Результат изменения данных представлен на рисунке 9 и 10. |

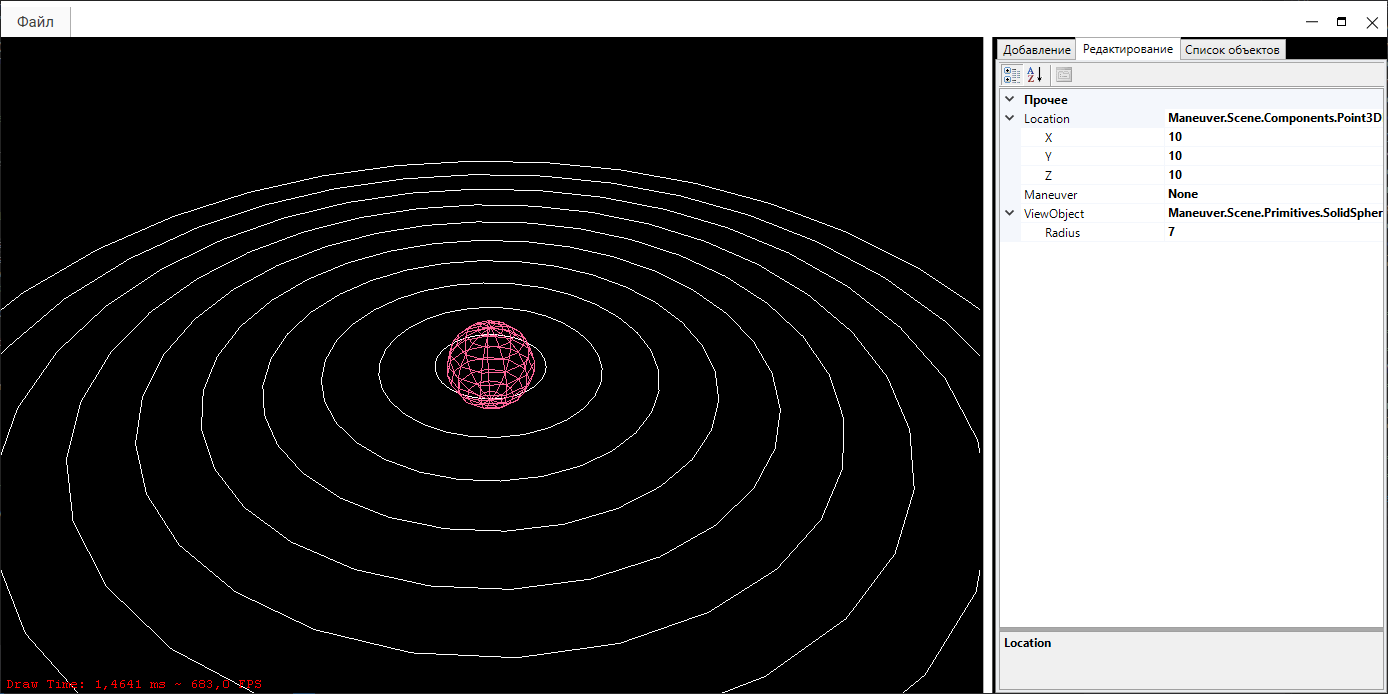


Рисунок 9 - Тест изменения радиуса сферы

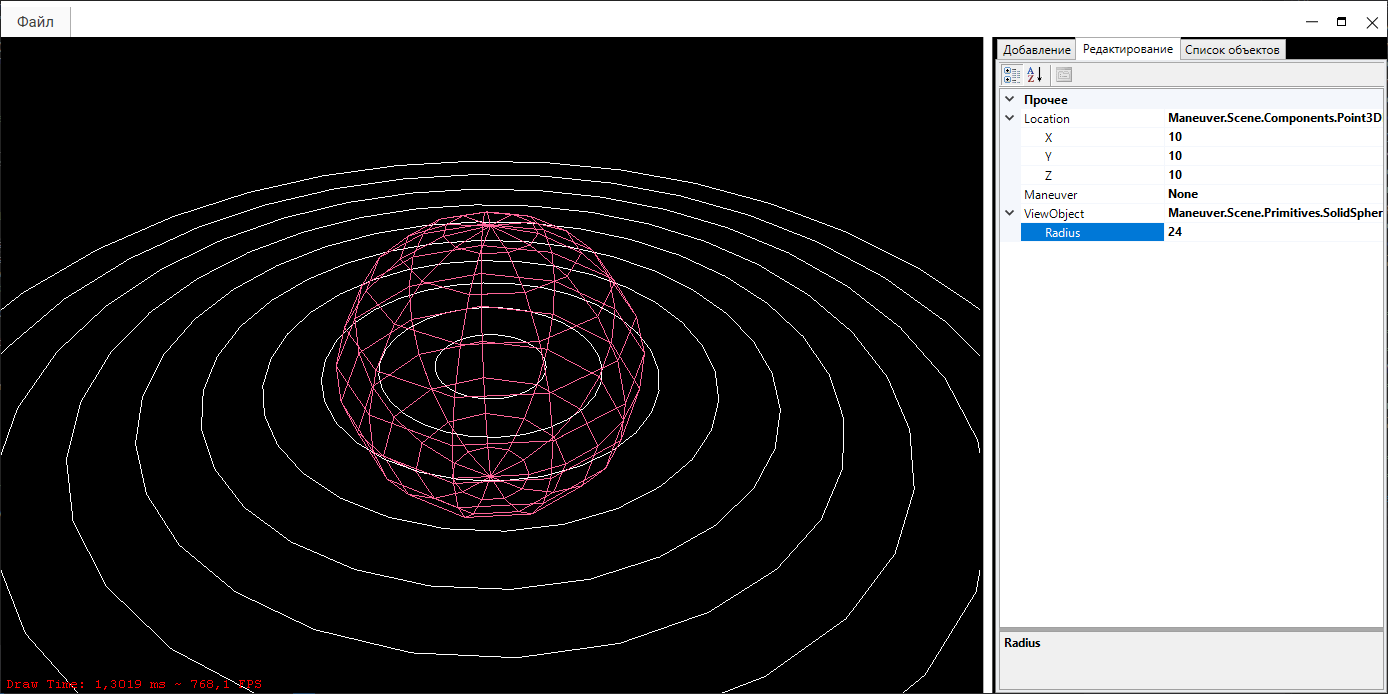


Рисунок 10 - Результат изменения радиуса сферы

Таблица 5 – Test-case для изменения расположения сферы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Тип теста | Описание | Шаги воспроизведения | Результат |
| 001 | Позитивный | Изменение расположения сферы | 1. Нажать на сферу; 2. Перейти в меню «Редактирование»; 3. Выбрать параметр «Location»; 4. Изменить значение «X». | Ожидаемый результат: расположение сферы изменено |
| Фактический результат: результат соответствует ожидаемому. Все условия соблюдены. Результат изменения данных представлен на рисунке 11 и 12. |

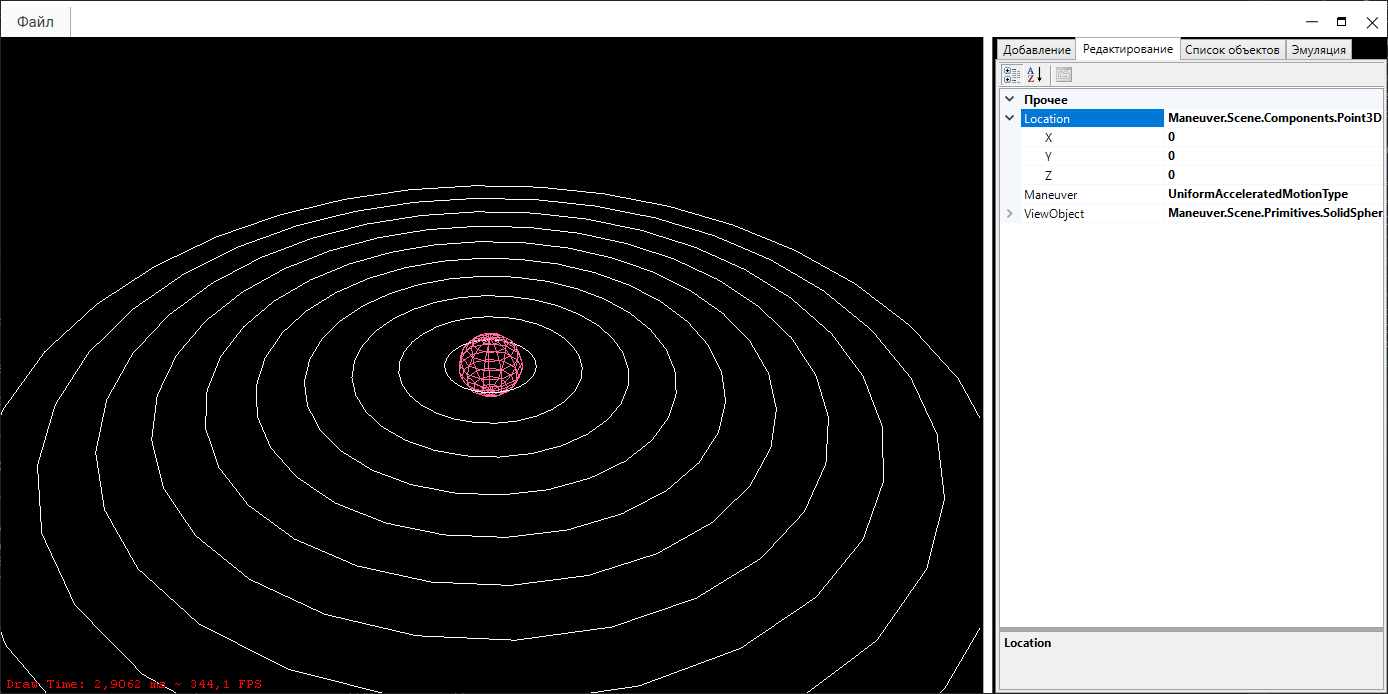


Рисунок 11 - Тест изменения расположения сферы

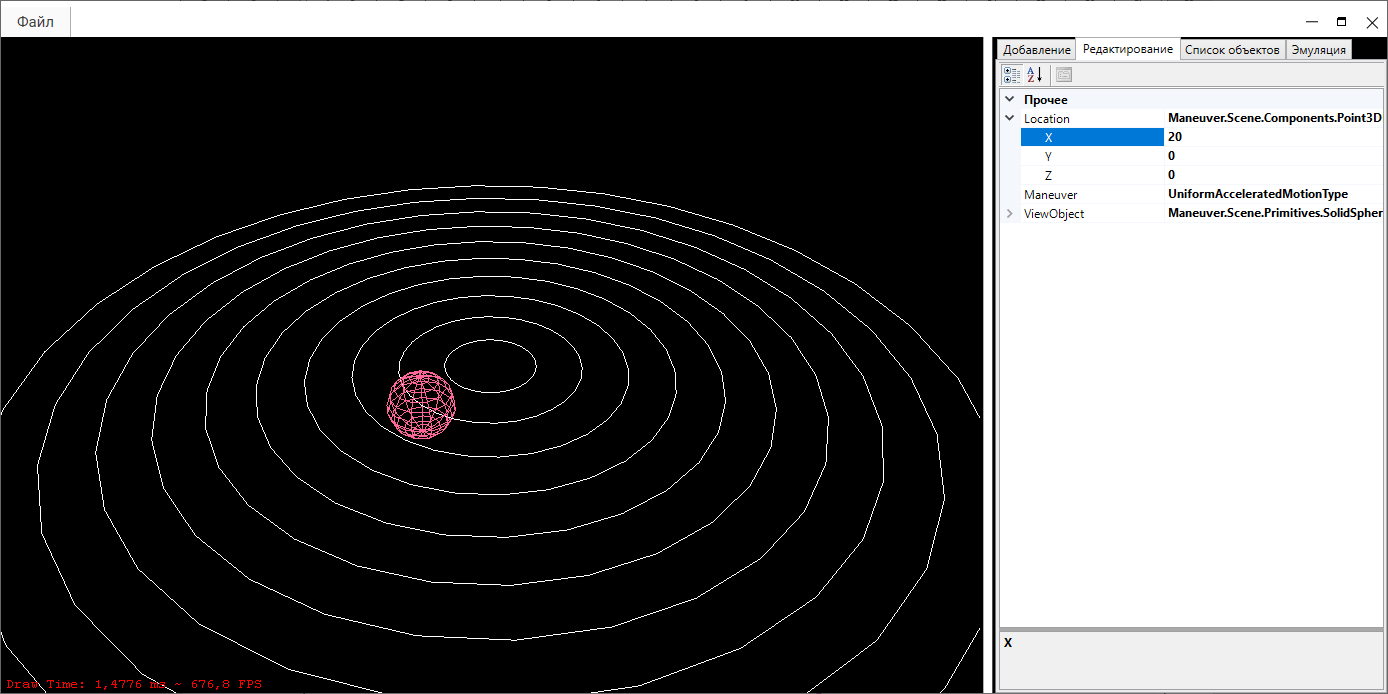


Рисунок 12 - Результат изменения расположения сферы

Таблица 6 – Test-case для сохранения плана полёта

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Тип теста | Описание | Шаги воспроизведения | Результат |
| 001 | Позитивный | Сохранение плана полёта | 1. Запустить эмуляцию полёта; 2. Нажать кнопку «Сохранить план полёта»; 3. Выбрать или ввести имя файла для сохранения с расширением .json или .xml; 4. Нажать «Выбрать». | Ожидаемый результат: файл с заданным именем сохранен |
| Фактический результат: результат соответствует ожидаемому. Все условия соблюдены. Результат изменения данных представлен на рисунке 13, 14 и 15. |

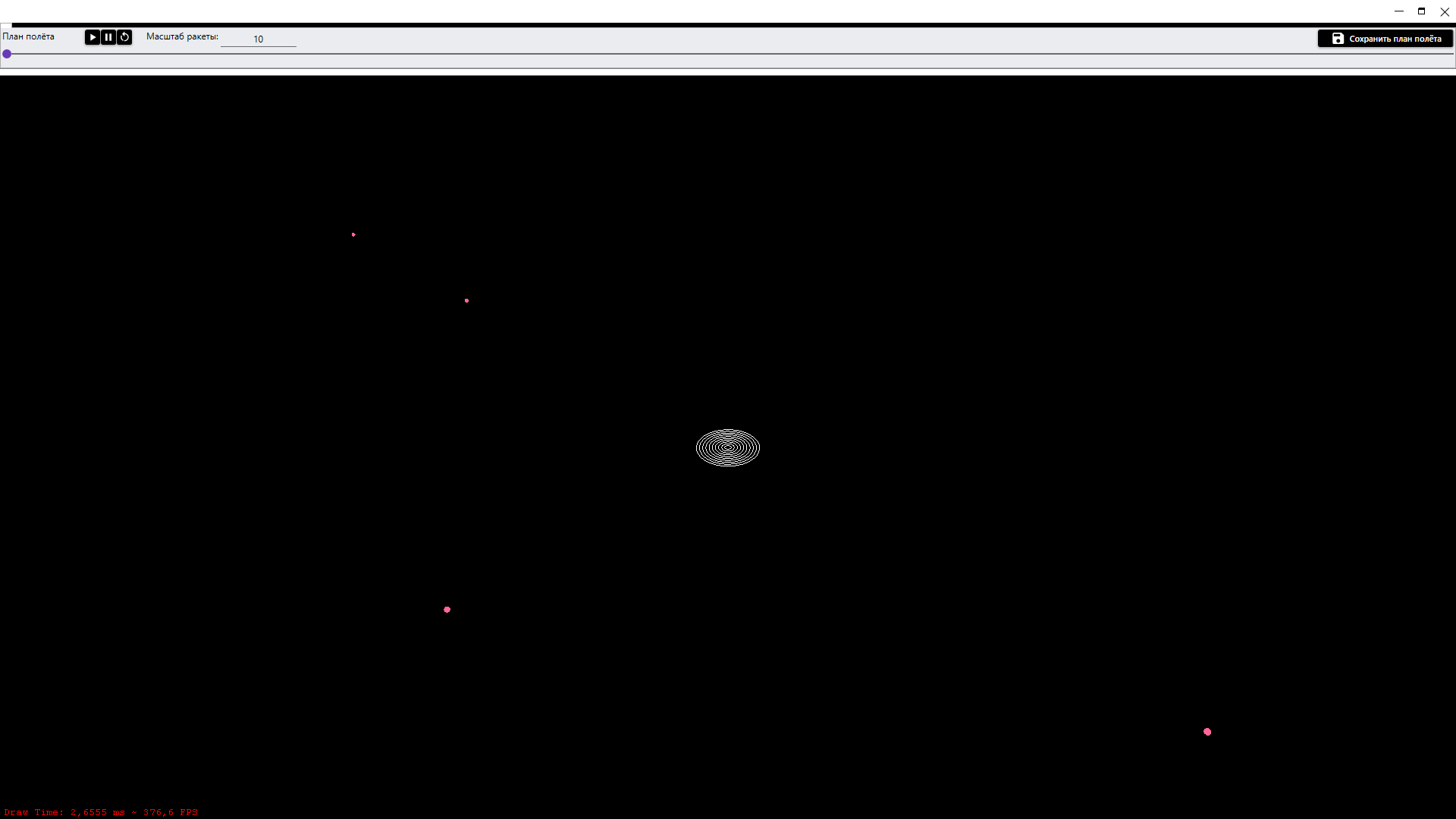


Рисунок 13 - Тест сохранения плана полёта

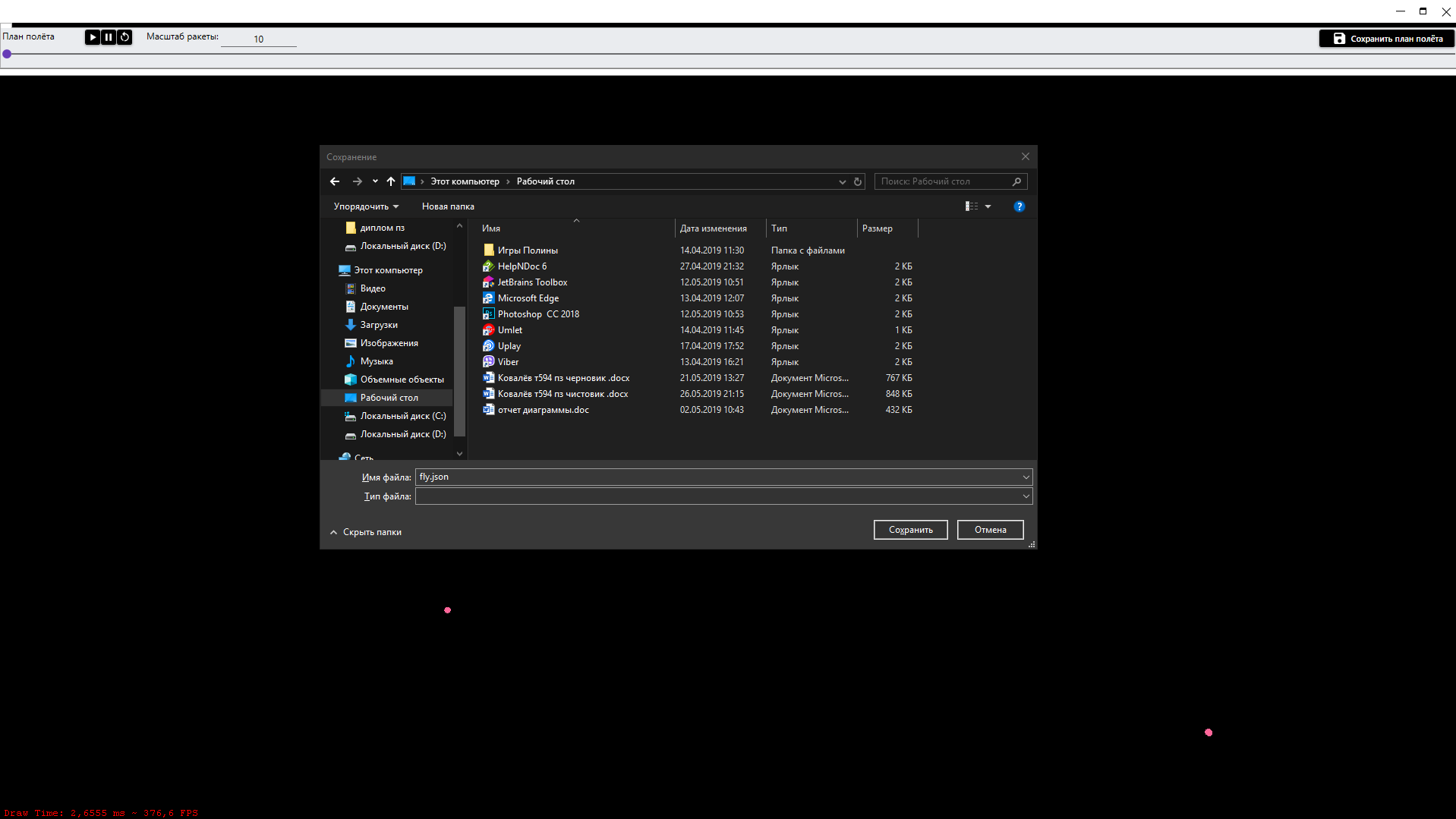


Рисунок 14 - Тест сохранения плана полёта

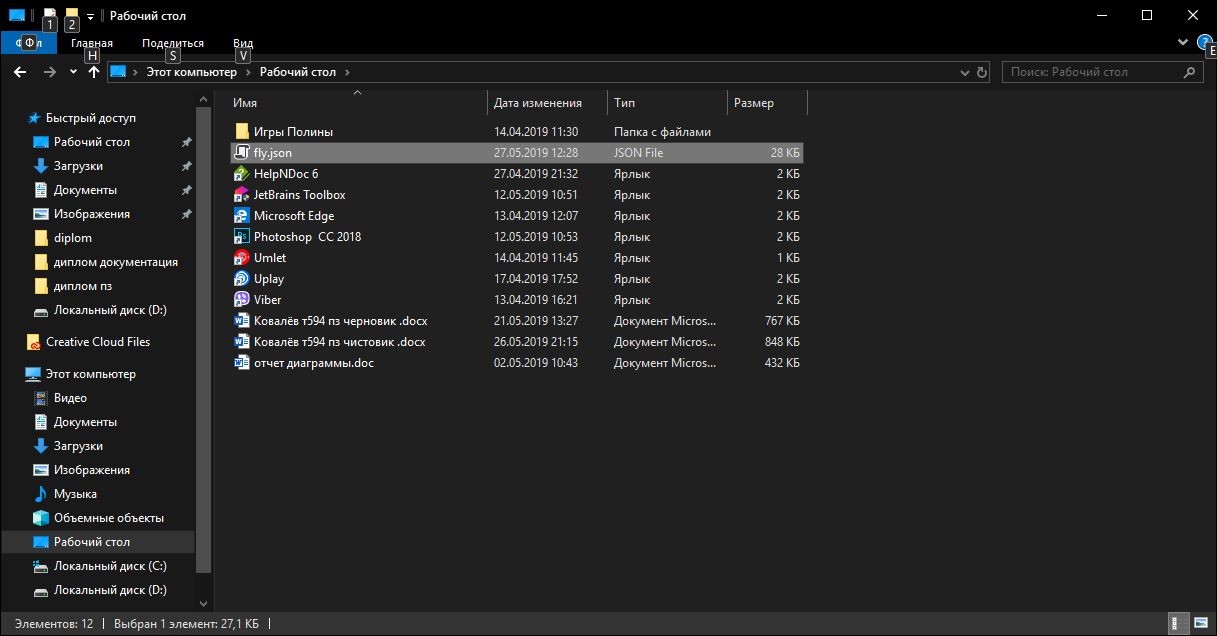


Рисунок 15 - Результат сохранения плана полёта

Таблица 7 – Test-case для отображения плана полёта

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Тип теста | Описание | Шаги воспроизведения | Результат |
| 001 | Позитивный | Отображение плана полёта | 1. Открыть существующий план полёта, нажав «План полёта – Открыть существующий»; 2. Выбрать файл с планом полёта; 3. Нажать кнопку «Play» | Ожидаемый результат: началось проигрывание полёта |
| Фактический результат: результат соответствует ожидаемому. Все условия соблюдены. Результат изменения данных представлен на рисунке 16, 17 и 18. |

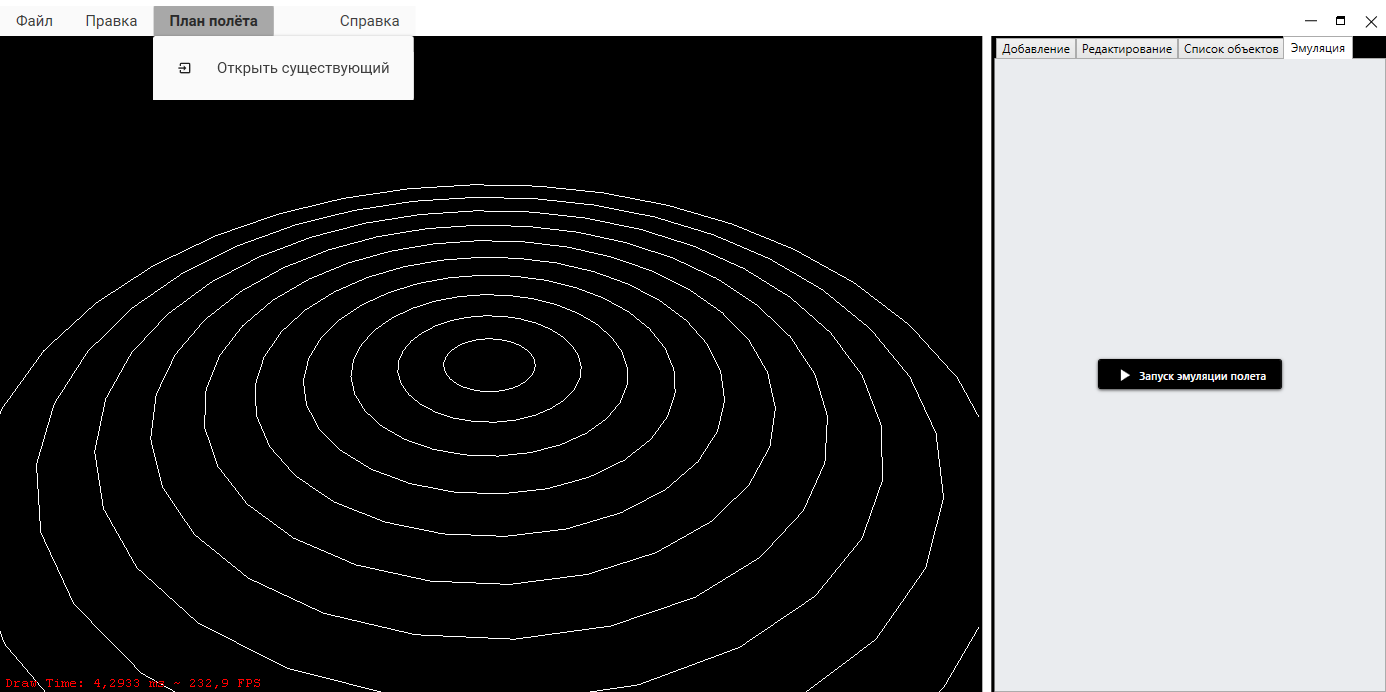


Рисунок 16 - Тест отображения плана полёта

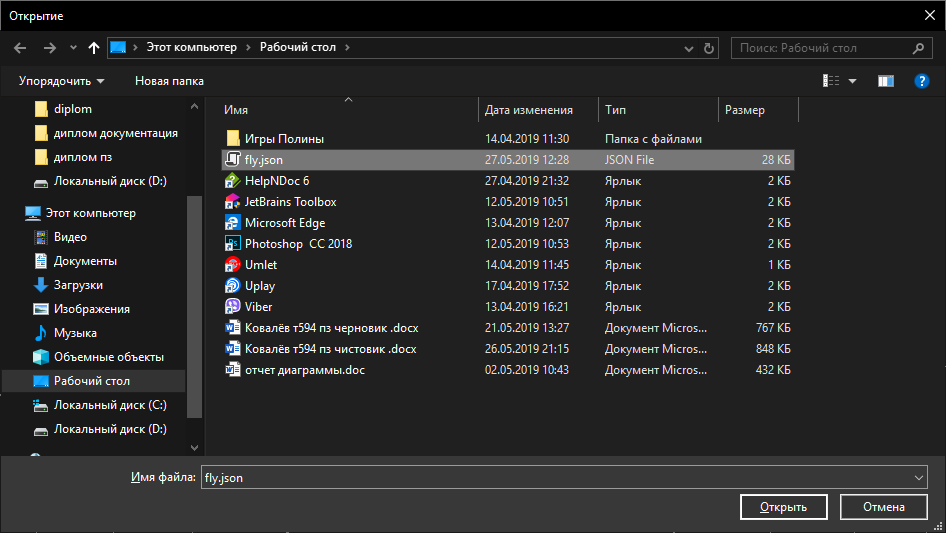


Рисунок 17 - Тест отображения плана полёта

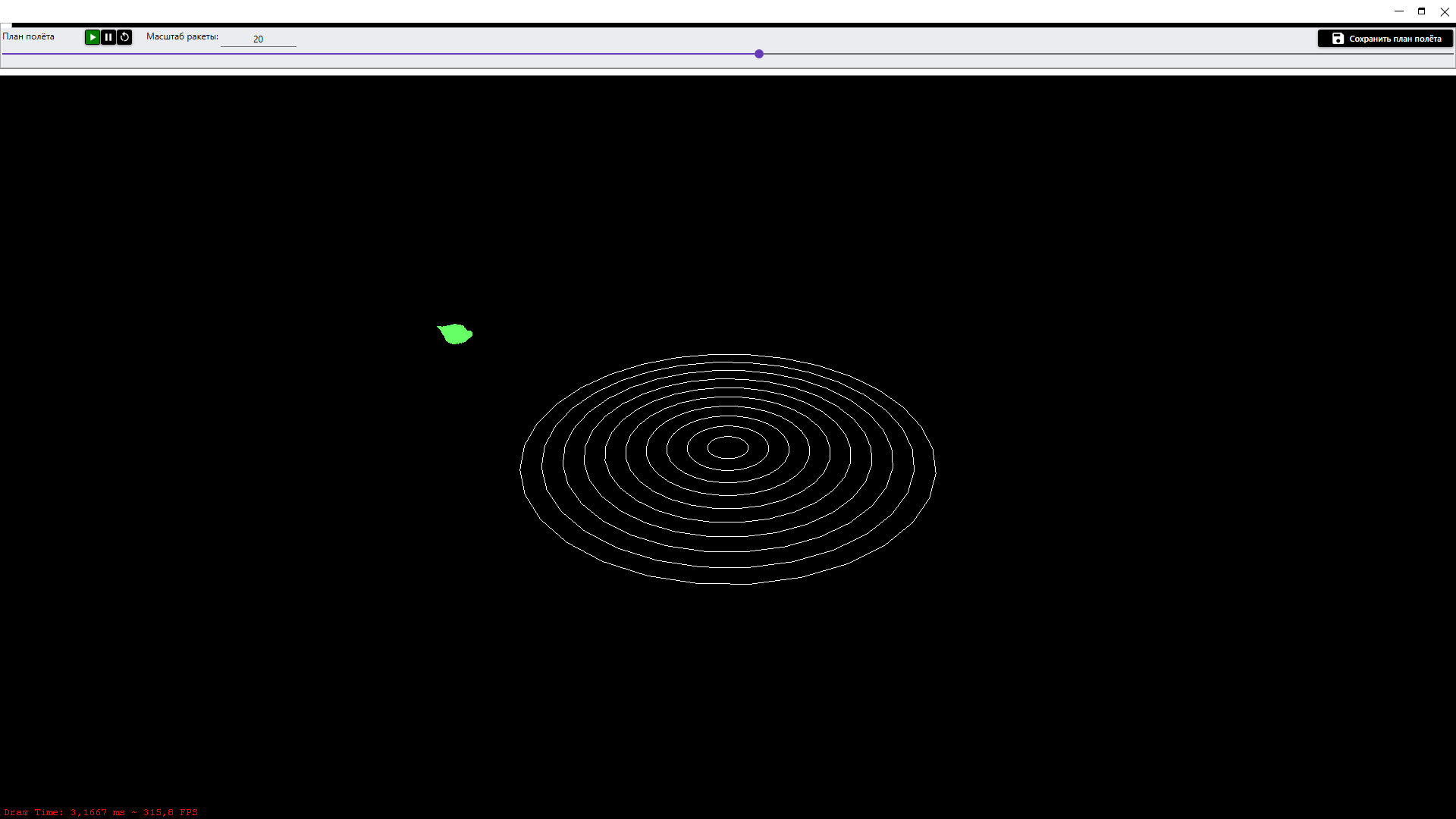


Рисунок 18 - Результат отображения плана полёта

# 6 Применение

## 6.1 Назначение программы

Программа «Maneuver.exe» предназначена для эмуляции полета ракеты. В этой программе пользователь создает сцену полёта, изменяет её, так же в этой программе можно просмотреть данные по полету ракеты и сохранения полученных данных в файлы типа xml или json.

## 6.2 Условия применения

Для применения данного программного средства необходимы следующие технически требования:

* процессор Intel Core 2 Duo или выше;
* OpenGL 4.0 (и выше) совместимый графический ускоритель;
* минимальный объем оперативной памяти — 1 Гбайт;
* подключение к интернету, база данных располагается на сервере;
* операционная система Windows 7 и выше;
* устройство для чтения дисков.

## 6.3 Справочная система

Справочная система программного средства представляет собой отдельный файл «Help.chm» открывается по нажатию кнопки «Справка» в новом окне программы. В справочной системе даны ответы на типичные вопросы, возникающие при работе с приложением, что, несомненно, должно помочь при освоении программного средства.

Справка имеет следующие разделы:

* «О программе»;
* «Системные требования»;
* «Руководство пользователя»;
* «Возможные вопросы и их решение».

Раздел «О программе» представлен на рисунке 19.

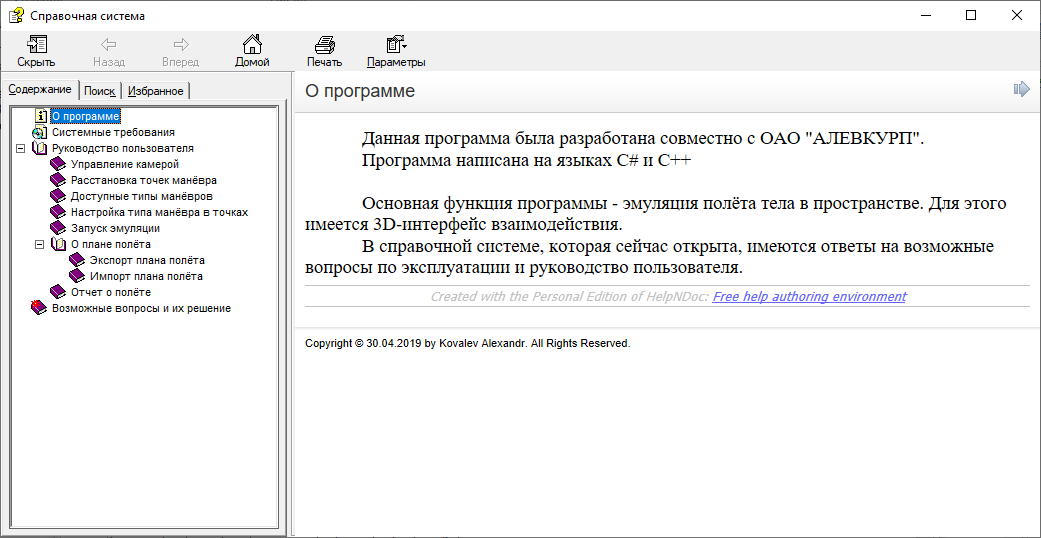


Рисунок 19 – Окно раздела «О программе» в справке

Раздел «Системные требования» представлен на рисунке 20.

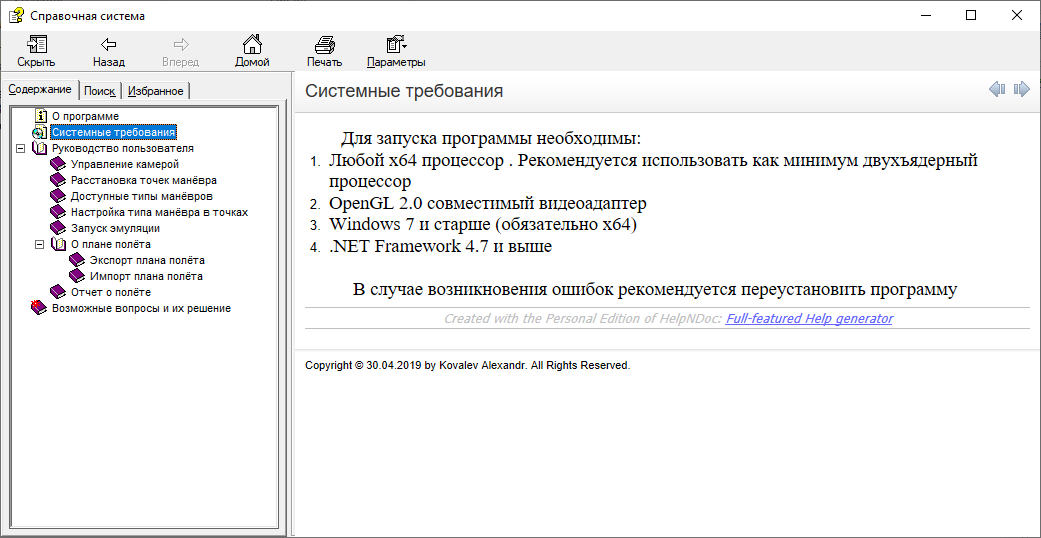


Рисунок 20 – Окно раздела «Системные требования» в справке

Раздел «Руководство пользователя» представлен на рисунках 21, 22, 23, 24, 25 и 26.

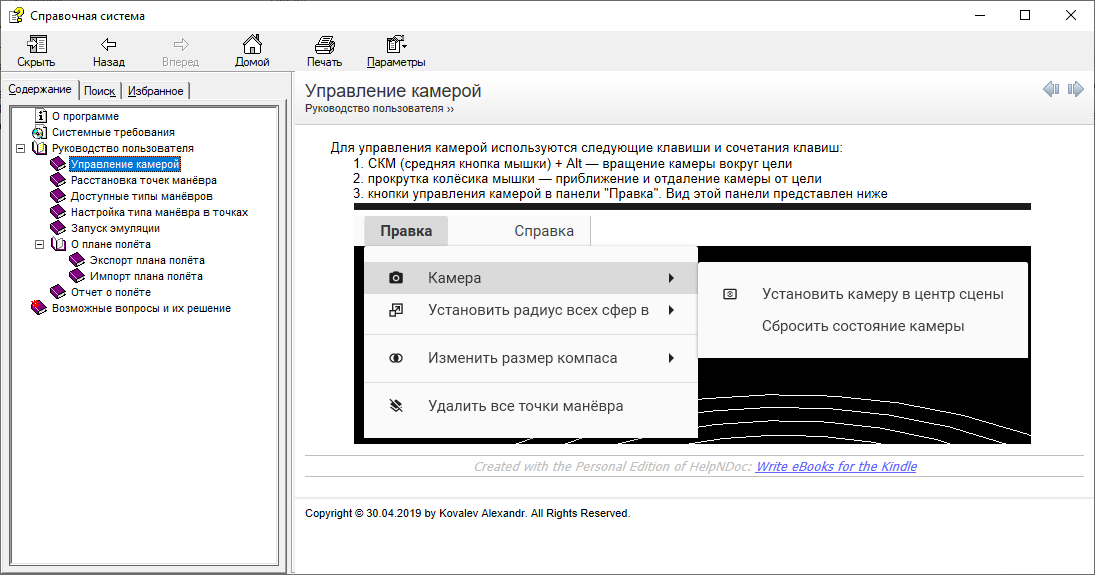


Рисунок 21 – Окно подраздела «Управление камерой» раздела «Руководство пользователя» в справке

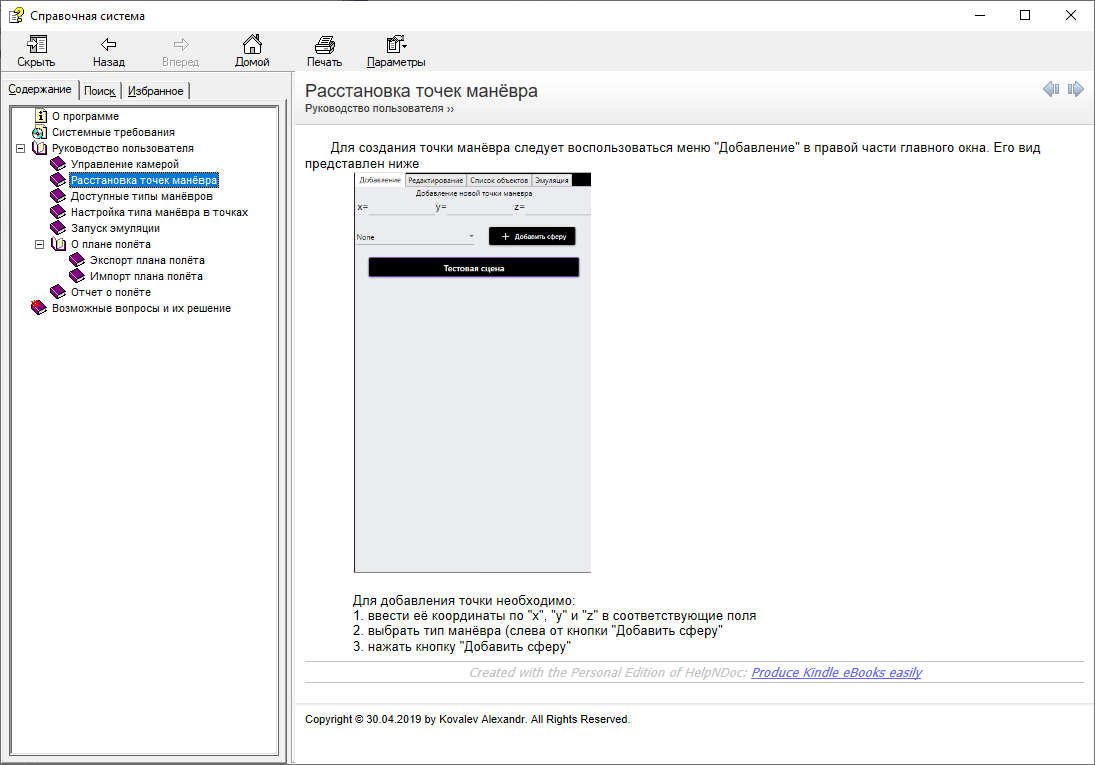


Рисунок 22 – Окно подраздела «Расстановка точек манёвра» раздела «Руководство пользователя» в справке

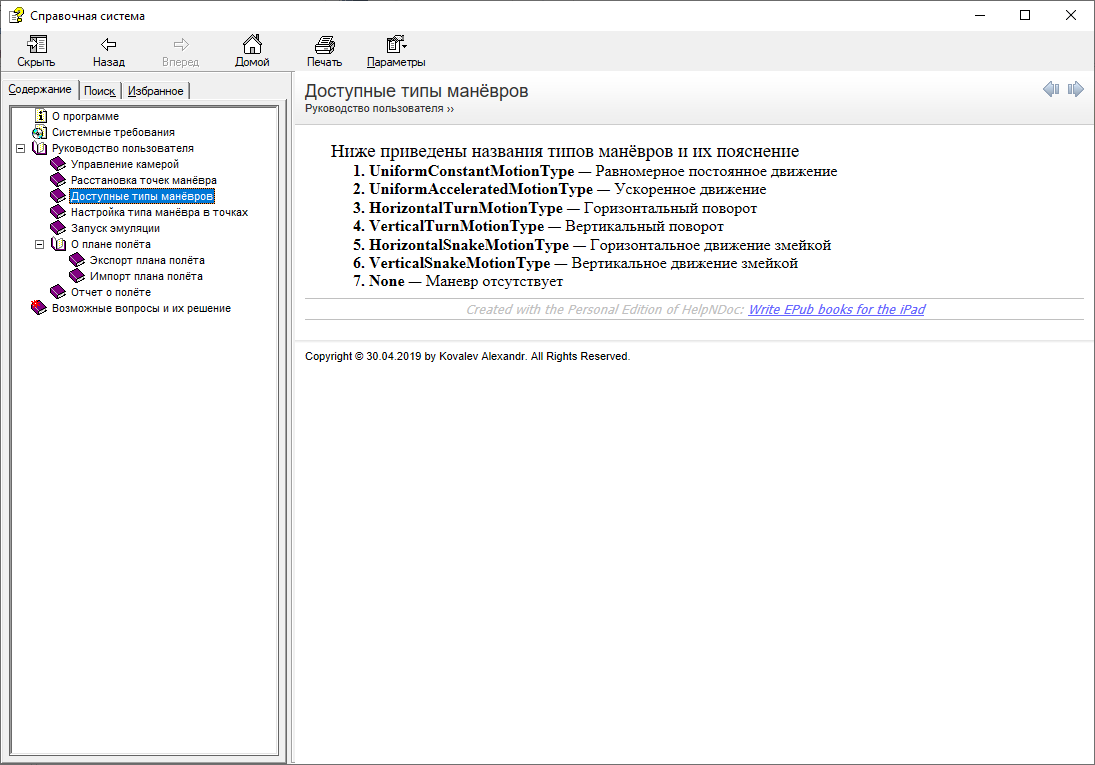


Рисунок 23 – Окно подраздела «Доступные типы манёвров» раздела «Руководство пользователя» в справке

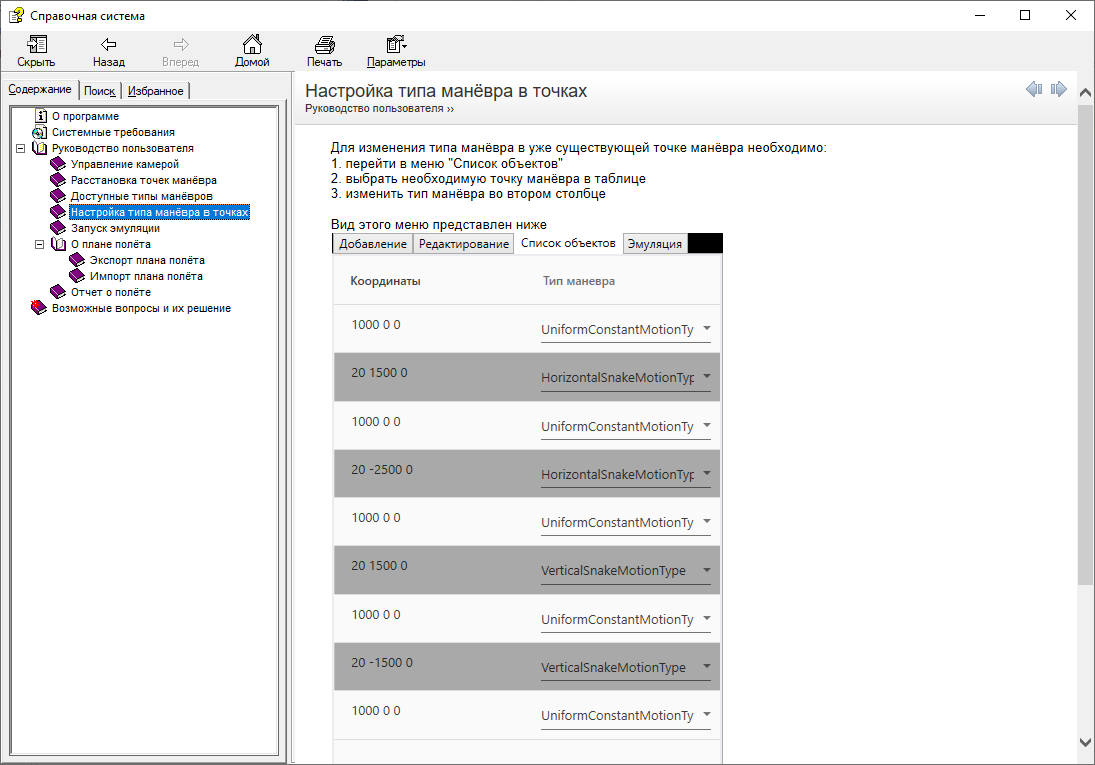


Рисунок 24 – Окно подраздела «Настройка типа манёвра в точках» раздела «Руководство пользователя» в справке

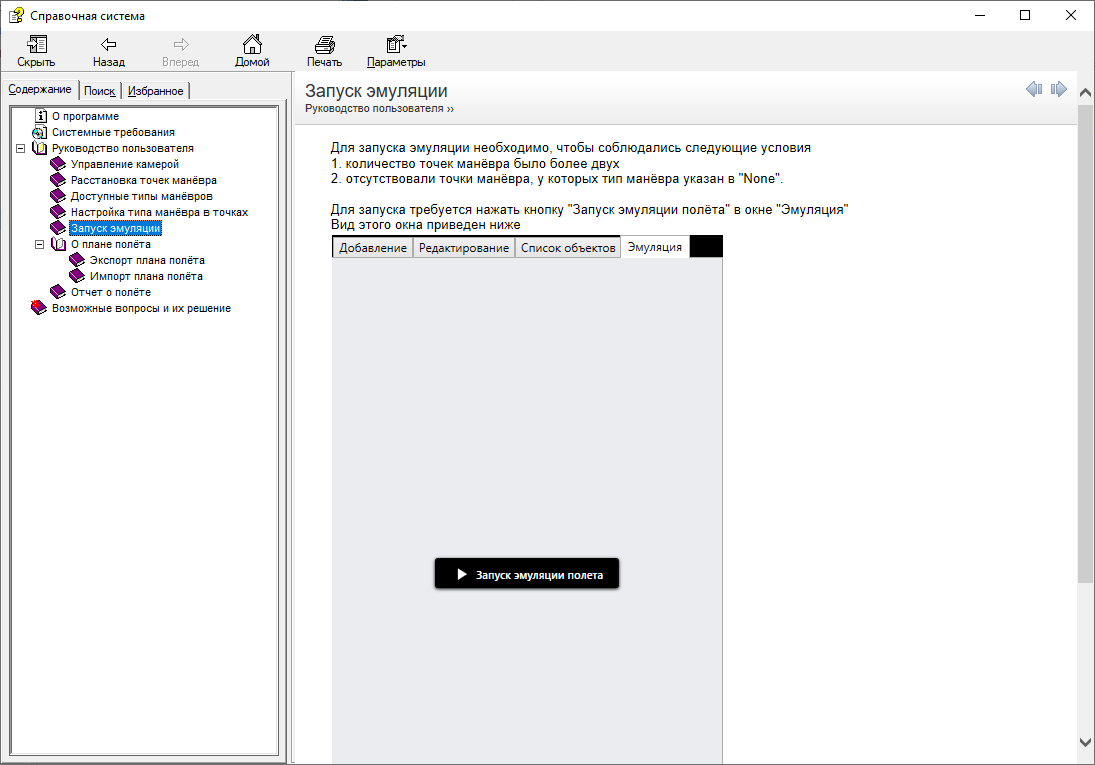


Рисунок 25 – Окно подраздела «Запуск эмуляции» раздела «Руководство пользователя» в справке

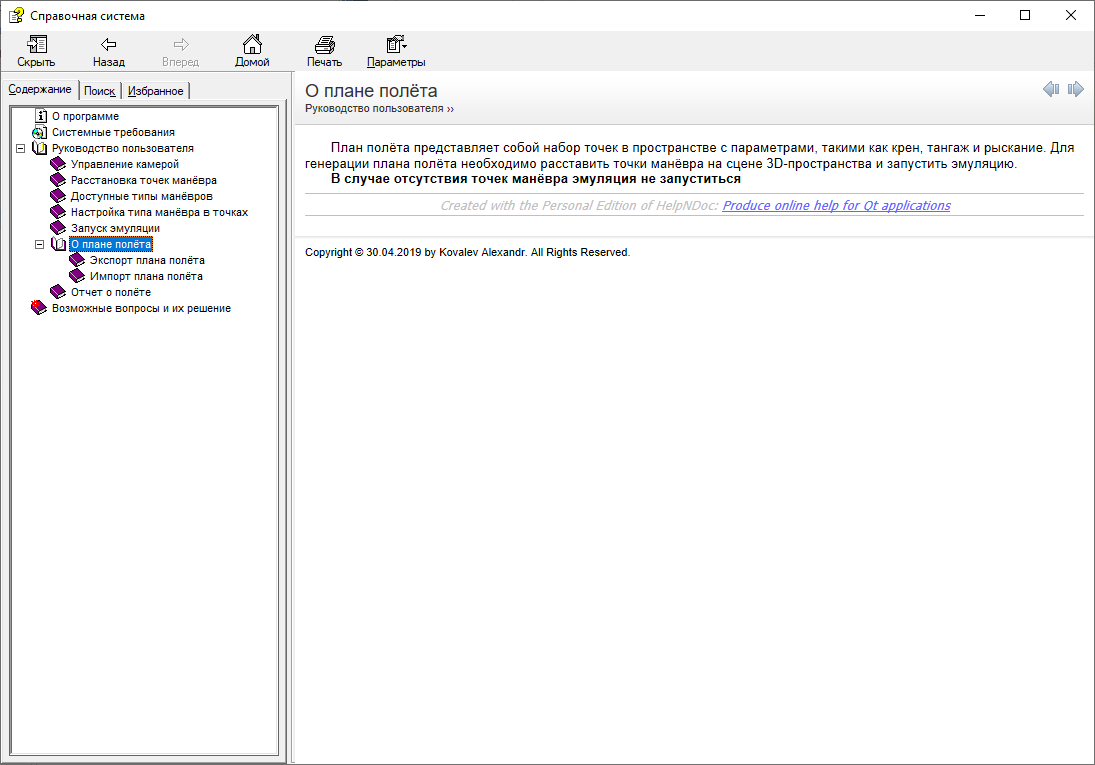


Рисунок 26 – Окно подраздела «О плане полёта» раздела «Руководство пользователя» в справке

7 Охрана труда и окружающей среды

7.1 Правовые, нормативные, социально-экономические и организационные вопросы охраны труда

В ОАО «АЛЕВКУРП» разработано и функционирует Система управления охраной труда (СУОТ). СУОТ разработан в соответствии с требованиями нормативных правовых актов Республики Беларусь в области охраны труда и СТБ 18001-2009 «Системы управления охраной труда. Требования» и направлено на создание на предприятии условий, обеспечивающих сохранение жизни и здоровья работников в процессе трудовой деятельности, соблюдение законодательства в области охраны труда.

Главный инженер предприятия организует выполнение мероприятий по устранению недостатков и нарушений по охране труда. О нарушениях, которые не могут быть устранены работниками структурного подразделения, сообщается генеральному директору предприятия.

Организация, оперативное руководство и координация работы по охране труда, общее руководство управлением охраной труда осуществляется генеральным директором ОАО «АЛЕВКУРП». Ответственным за реализацию функций СУОТ являются: главный инженер, заместитель генерального директора, начальники отделов и структурных подразделений.

Инженер по охране труда осуществляет организационно-методическую работу по управлению охраной труда, подготовку управленческих решений, контроль за их реализацией под руководством главного инженера.

Для организации работы и осуществления контроля по охране труда на предприятии создана служба охраны труда в лице инженера по охране труда.

Основными задачами службы охраны труда являются:

* совершенствование СУОТ;
* координация деятельности структурных подразделений организации по обеспечению здоровых и безопасных условий труда;
* внедрение передового опыта и новейших научных разработок по безопасности и гигиене труда, пропаганда охраны труда;
* осуществление контроля за соблюдением требований актов законодательства, технических нормативных правовых актов и локальных нормативных правовых актов, содержащих требования охраны труда.
* информирование и консультирование работников организации, в том числе ее руководителя, по вопросам охраны труда.

Функции инженера по охране труда:

* анализ причин производственного травматизма и профессиональной заболеваемости в организациях, расположенных на подведомственной территории, разработка мер по их профилактике;
* участие в организации разработки и реализации территориальных целевых программ улучшения условий и охраны труда;
* организация обучения, повышения квалификации и проверки знаний по вопросам охраны труда руководителей и специалистов организаций, расположенных на подведомственной территории, которые не находятся в подчинении республиканских органов государственного управления, иных государственных организаций, подчиненных Правительству Республики Беларусь, и других организаций;
* организация и руководство работой кабинета охраны труда в соответствии с Типовым положением о кабинете охраны труда, утвержденным постановлением Министерства труда Республики Беларусь от 8 ноября 1999 г;
* подготовка государственной статистической отчетности по охране труда;
* пропаганда и распространение передового опыта в области охраны труда в организациях, расположенных на подведомственной территории, в том числе через средства массовой информации, организацию выставок, смотров-конкурсов, семинаров, совещаний и других мероприятий;
* участие в установленном законодательством порядке в расследовании несчастных случаев на производстве;
* информационное обеспечение организаций, расположенных на подведомственной территории, по вопросам охраны труда;
* консультирование, подготовка ответов на запросы организаций, рассмотрение в установленном порядке писем, обращений граждан по вопросам охраны труда.

Обучение и повышение уровня знаний руководителей и специалистов по вопросам охраны труда осуществляется при всех видах обучения в соответствии с Законом РБ «Об охране труда», утвержденным Постановлением Кабинета Министров РБ от 23.07.2008 №356-3 (в редакции от 12.07.2013 №61-3).

Проверка знаний по вопросам охраны труда руководителей и специалистов проводится с учетом их должностных обязанностей и характера производственной деятельности, а также требований нормативных правовых актов, технических нормативных правовых актов по охране труда, соблюдение которых входит в их должностные обязанности.

Согласно «Правилам обучения безопасным методам и приемом работы, проведения инструктажа и проверки знаний по вопросам охраны труда», утвержденным постановлением Министерством труда и социальной защиты РБ 28.11.2008 № 175 [14] (в ред. Постановления Минтруда и соцзащиты от 27.06.2011 №50) в организациях (независимо от форм собственности, видов деятельности и ведомственной подчиненности) должны проводиться:

* вводный инструктаж (проводит инженер по охране труда);
* первичный инструктаж (проводится главными инженерами, начальниками отделов и подразделений);
* повторный инструктаж (проводится как первичный);
* внеплановый инструктаж (проводится как первичный);
* целевой инструктаж (проводится как первичный).

Расследование несчастных случаев и профессиональных заболеваний на предприятии осуществляется в соответствии с Правилами расследования и учета несчастных случаев на производстве и профессиональных заболеваний, утвержденными Постановлением Совета Министров Республики Беларусь от 10.01.2004 г. №30.

Расследование несчастного случая проводится уполномоченным должностным лицом предприятия, с участием уполномоченного представителя профсоюза, специалиста по охране труда. Расследование несчастного случая на производстве должно быть проведено в срок не более трех рабочих дней. В указанный срок не включается время, необходимое для проведения экспертиз, получения заключений правоохранительных органов, организаций здравоохранения и других органов и организаций. Акт формы НП составляется в четырех экземплярах.

Основными целями расследования несчастных случаев на производстве являются:

* установление причин, вызвавших несчастный случай;
* установление виновного в несчастном случае;
* организация мероприятий, направленных на предотвращение подобных несчастных случаев.

Контроль за состоянием охраны труда может осуществляться в форме проверок, обследований, осмотров, рассмотрения жалоб, заявлений.

Основными видами контроля на предприятии являются:

* общественный контроль за соблюдением законодательства об охране труда;
* периодический контроль за соблюдением законодательства об охране труда;
* производственный контроль за соблюдением требований промышленной безопасности на опасных объектах;
* контроль по охране труда, осуществляемый службой охраны труда;
* контроль за соблюдением законодательства об охране труда.

Периодический контроль, осуществляемый представителями нанимателя с участием общественных инспекторов профсоюзов по охране труда, проводится:

* ежедневно – в структурных подразделениях (ИВЦ, ОМТО, лаборатория, и др.) – руководителями структурных подразделений с участием общественного инспектора профсоюза по охране труда;
* ежемесячно – в цехе, отделе, иных аналогичных структурных подразделениях организации – начальником цеха с участием общественного инспектора профсоюза по охране труда, руководителей служб цеха и представителей службы охраны труда (инженера по охране труда);
* ежеквартально – в предприятии в целом – руководителем организации (его заместителями) с участием руководителей служб, отделов, общественного инспектора профсоюза по охране труда.

7.2 Обеспечение защиты от рентгеновского излучения при эксплуатации ПЭВМ

Рентгеновское излучение представляет собой электромагнитные волны, энергия фотонов которых лежит на шкале электромагнитных волн между ультрафиолетовым излучением и гамма-излучением (от ~100 эВ до ~1 МэВ), что соответствует длинам волн от ~103,1 до ~10−2 Å (от ~10 до ~10−3 нм). Фотоны характеристического (то есть испускаемого при переходах в электронных оболочках атомов) рентгеновского излучения имеют энергию от 10 эВ до 250 кэВ, что соответствует излучению с частотой от 2⋅1015 до 6⋅1019 Гц и длиной волны 0,005—100 нм (общепризнанного определения нижней границы диапазона рентгеновских лучей в шкале длин волн не существует) [8].

В настоящее время в практике рентгеновской интроскопии существуют рентгеновские излучатели и рентгеновские моноблоки. Среди бытовых источников рентгеновского излучения являются дисплеи старых телевизоров и мониторов, сотовые телефоны, маршрутизаторы WiFi и Bluetooth. Однако, лишь первые оказывали хоть какое-то влияние на здоровье и состояние человека. В текущих реалиях пострадать от какого бы то ни было излучения на современных устройствах попросту невозможно, стандарты, вводимые во всём мире, не позволяют устройствам работать на повышенной мощности пример тому WiFi адаптер.

Перед вводом новых технологий или продукции в продажу проводятся тщательные исследования, и особенно в вопросе техники. В следствие приведенных выше фактов, нет причины беспокоится о причинах и о последствиях облучения персонала в случаях работы не в опасных условиях.

Согласно статье 8 Закона Республики Беларусь от 5 января 1998 года № 122-3 «О радиационной безопасности населения» установлены следующие основные пределы доз облучения на территории Республики Беларусь в результате воздействия источников ионизирующего излучения:

* для населения средняя годовая эффективная доза равна 0,001 зиверта или эффективная доза за период жизни (70 лет) - 0,07 зиверта; в отдельные годы допустимы большие значения эффективной дозы при условии, что средняя годовая эффективная доза, исчисленная за пять последовательных лет, не превысит 0,001 зиверта;
* для работников (персонала) средняя годовая эффективная доза равна 0,02 зиверта или эффективная доза за период трудовой деятельности (50 лет) - 1 зиверту; допустимо облучение в размере годовой эффективной дозы до 0,05 зиверта при условии, что средняя годовая эффективная доза, исчисленная за пять последовательных лет, не превысит 0,02 зиверта.

Для безопасности и защиты персонала от рентгеновского излучения необходимо принять следующие меры:

* не покупать устаревшее оборудование, например, мониторы на электро-лучевые трубках;
* не располагать персонал вблизи источников рентгеновского излучения;
* проводить регулярные замеры излучения в рабочих помещениях, а в случаях повышенных норм устранять эту проблему.

В случаях обнаружения неисправного оборудования следует предпринять меры по замене или починке данного оборудования.

7.3 Пожарная безопасность

По взрывопожарной и пожарной опасности помещения и здания ОАО «АЛЕВКУРП» относятся по ТКП 474-2013 к категории B1-B4, так как в помещениях обращаются горючие вещества и материалы, способные при взаимодействии с кислородом воспламенятся.

По классификации пожароопасных зон ПУЭ, организация относится к зоне П-IIa, так как имеют зоны, расположенные в помещениях, в которых обращаются твердые горючие вещества.

Здания для ВЦ и части зданий другого назначения, в которых предусмотрено размещение электронных вычислительных машин, относятся к II степени огнестойкости, поскольку здание имеет несущие и ограждающие конструкции из естественных или искусственных каменных материалов, бетона или железобетона с применением листовых и плитных негорючих материалов.

Строительные конструкции здания имеют следующий предел огнестойкости и класс пожарной опасности:

* несущие элементы здания (R45-K1) – предел огнестойкости 45 минут по потере несущей способности, мало пожароопасные;
* самонесущие стены (RE 30-K1) – предел огнестойкости 30 минут по потере несущей способности и потере целостности независимом от того, какое из двух предельных состояний наступит ранее, мало пожароопасные;
* наружные несущие стены (E 15-K2) – предел огнестойкости 15 минут по потере целостности, умеренно пожароопасные;
* перекрытия междуэтажные (RE1 45-K1) – предел огнестойкости 45 минут по потере несущей способности, целостности и теплоизолирующей способности независимо от того, какое из трех предельных состояний наступит ранее, мало пожароопасные;
* элементы бесчердачных покрытий – настилы, в том числе с утеплителем (RE 15-K1) – предел огнестойкости 15 минут по потере несущей способности и потере целостности независимо ото того, какое из двух предельных состояний наступит ранее, мало пожароопасные;
* элементы бечердачных покрытий – фермы, балки, прогоны (R 15-K1) – предел огнестойкости 15 минут по потере несущей способности, мало пожароопасные;
* лестничные клетки – внутренние стены (REI 60-KO) – предел огнестойкости 60 минут по потере несущей способности, целостности и теплоизолирующей способности независимо от того, какое их тех предельных состояний наступит ранее, не пожароопасные;
* лестничные клетки – марши и площадки лестниц (R 45-KO) – предел огнестойкости 45 минут по потере несущей способности, не пожароопасные.

Согласно требованиям пожарной безопасности, предприятие обеспеченна первичными средствами пожаротушения:

* огнетушители (водные (ОВ), углекислотные (ОУ-2));
* пожарные краны (ПК).

Первичные средства пожаротушения расположены в коридорах, проходах, не препятствуют безопасной эвакуации людей. Их располагают на видных местах вблизи от выходов из помещений на высоте не более 1,5 м.

Расстояние от возможного очага возгорания до места размещения огнетушителя не должно превышать 20 метров.

В каждом кабинете имеется один огнетушитель ОП-8 НЗГА с диоксидом углерода, расположен рядом с выходной дверью на высоте метра от уровня пола.

Так же во всех помещениях установлены тепловые пожарные извещатели ИП 101-01-BM. Метод действия заключается в периодическом измерении температуры пространства, и формирования электрического сигнала о возникшем пожаре и передачи его на приемно-контрольные приборы.

В ОАО «АЛЕВКУРП» для проведения профилактических мероприятий по предупреждению и тушению пожаров создана добровольная пожарная дружина в составе:

* командир добровольной пожарной дружины – 1чел;
* командир расчета – 2чел;
* старший боевого расчета – 4чел;
* член боевого расчета – 12чел.

На добровольную пожарную дружину возложены обязанности по контролю за соблюдением противопожарного режима в помещениях, проведение разъяснительной работы среди работников предприятия по соблюдению противопожарного режима на рабочих местах, вызов пожарной службы в случае возникновения пожара, принятие мер по его тушению имеющимися средствами пожаротушения.

Выход из здания осуществляется поэтапно:

* на первом этапе люди покидают внутренние помещения;
* второй этап — выбор пути движения к выходу;
* третий этап — перемещение людей на безопасное расстояние от горящего здания.

В высотных зданиях принят следующий порядок действий:

* первым эвакуируется этаж, на котором произошло возгорание, затем выводят людей с этажа выше и ниже, потом с оставшихся этажей, начиная с верхних;
* далее эвакуируются подвальные помещения.

7.4 Охрана окружающей среды

Процесс удаления из помещения отработанного воздуха крайне необходим. В ОАО «АЛЕВКУРП» для очистки воздуха служат вентиляционные системы. Они обеспечивают санитарно-гигиенические условия воздушной среды в помещении, благоприятные для здоровья и самочувствия человека. Главные принципы обслуживания вентиляционных систем: регулярность и своевременность, оперативность.

Каждые три месяца необходимо проделывать следующие работы:

* предварительно осматривать всю систему и проводить анализ издаваемых звуков;
* проверять степень эффективности очистки системой подаваемого воздуха в помещение;
* проверка состояния оборудования;
* проверка воздуховодов, и освобождение от скопившейся пыли и других различных отложений;
* проверка состояния воздушных фильтров;
* контроль показаний всех приборов, определение достоверности предоставляемых данных.

Каждые пол года необходимо проводить следующее техобслуживание вентиляционной системы:

* определение параметров силы тока в системе;
* крыльчатка вентилятора очищается от скопившихся отложений и обрабатывается дезинфицирующими средствами, специально для этого предназначенными.

Процесс очистки сточных вод делится на 4 стадии:

* механическая;
* биологическая;
* физико-химическая;
* дезинфекция сточных вод.

Механическая нужна для предварительной очистки сточных вод, которые поступают на очистные сооружения для последующей биологической очистки.

Биологическая необходима для очистки органических загрязнений. Далее во время физико-химической происходит очистка от растворенных примесей. И для окончательного обеззараживания вод происходит дезинфекция.

Утилизация твёрдых производственных отходов происходит с помощью их захоронения на специальных полигонах. Мусор уплотняется машинами по всей территории, а после оборудуется специальной системой труб для откачки «свалочного газа» и покрывается слое грунта.

Обезвреживание и захоронение токсичных промышленных отходов осуществляется на специальных инженерных сооружениях - полигонах захоронения токсичных промышленных отходов. Полигоны захоронения промышленных отходов обеспечивают изоляцию токсичных веществ от селитебной зоны и защиту окружающей среды от загрязнения за пределами санитарно-защитной зоны.

8 Экономический раздел

8.1 Технико-экономическое обоснование разработки программного средства

Необходимо рассчитать экономическую эффективность разработки программного средства.

В технико-экономическом обосновании будут рассмотрены следующие вопросы:

- составление плана по разработке программного средства;

- расчет стоимости разработки;

- экономическая эффективность разработки.

В первой части технико-экономического обоснования необходимо рассчитать срок разработки по созданию программного средства «Программа эмуляции полёта ракеты».

Во второй части технико-экономического обоснования необходимо рассчитать стоимость разработки программного средства с момента получения первого варианта технического задания и заканчивая оформлением документации и сдачей разработки.

В третьей части необходимо оценить экономическую эффективность программного средства.

8.2 Составление плана по разработке программного средства

План разработки программного средства представлен в таблице 8.1.

Таблица 8.1 - План разработки программного средства

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапов и видов работ | Исполнитель | Количество исполнителей | Трудоемкость, человеко-дни |
| Подготовительный | техник-программист | 1 | 5 |
| Выбор методов и средств | техник-программист | 1 | 2 |
| Разработка алгоритмов и программ | техник-программист | 1 | 10 |
| Отладка программ и анализ результатов | техник-программист | 1 | 16 |
| Оформление документации и подготовка к сдаче разработки | техник-программист | 1 | 7 |

8.3 Определение цены программного средства

В условиях рыночных отношений научно-техническая продукция также является товаром. Поэтому узловым вопросом технико-экономического обоснования выступает определение цены основного результата дипломного проекта.

Вместе с тем следует иметь в виду, что в отраслевых рекомендациях по установлению цены на научно-техническую продукцию фактически реализован вариант механической аналогии с материальным производством (издержек или сметной стоимости) и учет минимального уровня рентабельности.

Расчет цены основного результата дипломного проекта осуществляется в определенной последовательности.

Определяются материальные затраты на выполнение работ по теме, включая стоимость покупных комплектующих изделий и полуфабрикатов на изготовление макетов и опытных образцов.

Расчет осуществляется по формуле (8.1).

, (8.1)

где Ктзр - коэффициент, учитывающий транспортно-заготовительные расходы Ктзр≈от 1,05 до 1,10;

Hpi- норма расхода i-го вида материалов на макет или опытный образец (кг, м, и так далее);

Цi- действующая отпускная цена за единицу i-го вида материала, руб.;

Odi- возвратные отходы i-го вида материала (кг, м, и так далее);

Цdi- цена за единицу возвращенных отходов i-го вида материала, руб.;

n – количество применяемых видов материалов.

Расчет целесообразно представить в таблице 8.2.

Таблица 8.2 - Расчет затрат на материалы, покупные полуфабрикаты и комплектующие изделия

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Наименование материалов покупных полуфабрикатов и комплектующих изделий | Единица измерения | Количество | Цена при-обретения без НДС, руб. | НДС, руб. | Цена с НДС, руб. |
| Бумага | пачка | 1 | 14,00 | 2,80 | 16,80 |
| Ручка шариковая | штука | 1 | 1,00 | 0,20 | 1,20 |
| Папка-скоросшиватель | штука | 1 | 2,00 | 0,40 | 2,40 |
| Диск | штука | 1 | 2,00 | 0,40 | 2,40 |
| Всего расходов | | | | | 22,80 |
| Всего с транспортно-заготовительными расходами | | | | | 24,00 |

Затраты на электроэнергию находятся исходя из продолжительности периода разработки программного обеспечения, количества кВт/ч, затраченных на его проектирование и тарифа за 1 кВт/ч. по следующей формуле:

, (8.2)

гдеКэ– стоимость одного кВт/ч,руб;

Тр – количество кВт/ч.

Тарифы на электроэнергию применяются согласно приложения к Декларации *«*Об уровне тарифов на электроэнергию, отпускаемую РУП Электроэнергетики ГПО “Белэнерго” для юридических лиц и ИП» на соответствующий период времени, когда разрабатывается программное обеспечение.

Базовый тариф для прочих потребителей с 01.01.2019 г. составляет 0,3199 руб. за 1 кВт/ч. Время реализации проекта 40 дней, среднее потребление энергии в месяц составило 75 кВт/ч, то есть было потреблено 143 кВт/ч. Исходя из вышеизложенного получаем, что на электроэнергию было затрачено:

Рэ=0,3199 × 143 = 45,75 руб.

Определяется основная заработная плата научно-технического персонала, непосредственно занятого выполнением работ.

К этой статье относятся основная заработная плата работников, а также премии, входящие в фонд заработной платы. Среднее количество рабочих дней в месяце равно 21, а средняя продолжительность рабочего дня составляет восемь часов. Следовательно, часовая заработная плата определяется делением размера оклада на количество рабочих часов в месяце (то есть на 168 часов).

Тарифная ставка седьмого разряда на предприятии составляет 86 руб. Трудоемкость определяется исходя из данных, представленных в таблице 8.1.

Произведение трудоемкости на сумму часовой заработной платы определяет затраты по зарплате для работника на все время разработки.

Расчет основной заработной платы по теме приведен в таблице 8.3.

Таблица 8.3 - Расчет затрат на основную заработную плату научно-производственного персонала

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Наименование этапов работ | Исполнитель | Разряд | Тарифный  коэффициент | Часовая тарифная ставка, руб. | Трудоемкость,  (чел/час) | Затраты по заработной плате, руб. |
| Подготови-тельный | техник-программист | 7 | 2,03 | 0,51 | 40 | 41,41 |
| Выбор методов и средств | техник-  программист | 7 | 2,03 | 0,51 | 16 | 16,56 |
| Разработка алгоритмов и программ | техник-  программист | 7 | 2,03 | 0,51 | 80 | 82,82 |
| Отладка программ и анализ результатов | техник-  программист | 7 | 2,03 | 0,51 | 128 | 132,52 |
| Оформление документации и подготовка к сдаче разработки | техник-  программист | 7 | 2,03 | 0,51 | 56 | 57,98 |
| Всего | | | | | | 331,29 |
| Всего с коэффициентом премий | | | | | | 404,17 |

Определяется дополнительная заработная плата исполнителей, включающая разнообразные предусмотренные трудовым законодательством выплаты, по формуле (8.3).

, (8.3)

где Ндз - норматив дополнительной заработной платы, Ндз ≈ от 10 до 25 %.

Рдз=404,17 × 0,22 = 88,92 руб.

Рассчитываются отчисления органам социальной защиты по формуле (8.4):

, (8.4)

где Нос- норма отчислений на социальную защиту, Нос=34%.

Рос= (404,17 + 88,92) × 0,34 =167,65 руб.

Также рассчитываются отчисления на страхование от несчастных случаев на производстве и профессиональных заболеваний (Остр) по ставке действующего законодательства (Нбгс принимается равным от 0,3 до 0,9%). Для расчетов среднее значение Нбгс  принимается равным 0,6%. Остр рассчитывается по формуле (8.5).

. (8.5)

Остр= (404,17 + 88,92) × 0,006 = 2,96 руб.

Определяются прочие прямые расходы, связанные с амортизационными отчислениями на полное восстановление основных производственных фондов, арендная плата и лизинговые платежи, компенсация за износ (амортизацию) использованного в процессе создания научно-технической продукции оборудования по договоренности. Рпр рассчитывается по формуле (865).

, (8.6)

где Нпр– норматив прямых расходов, Нпр≈от 10 до 20 %.

Рпр= 404,17 × 0,11 = 44,46 руб.

Исчисляются косвенные (накладные) расходы по формуле (8.7).

, (8.7)

где Нкос - норматив косвенных расходов, Нкос≈от 50 до 100%.

Ркос= 404,17 × 0,88 = 355,67 руб.

Определяется полная себестоимость научно-технической продукции как сумма всех групп затрат по формуле (8.8).

(8.8)

Сп= 24,00 + 45,75 + 404,17 + 88,92 + 167,65 + 2,96 + 44,46 + 355,67 = 1133,58 руб.

По среднему уровню рентабельности в процентах от полной себестоимости определяется плановая прибыль единицы научно-технической продукции по формуле (8.9).

, (8.9)

где Ур - средний уровень рентабельности, Ур ≈от 10 до 30%.

П= 1133,58 × 0,22 = 249,37 руб.

Определяется приближенная (ориентировочная) отпускная цена научно-технической продукции по формуле (8.10).

. (8.10)

Цопт=1133,58 + 249,39 = 1382,97 руб.

Определяется налог на добавленную стоимость (НДС) по формуле (8.11).

, (8.11)

где ННДС - ставка налога НДС, ННДС=20 %.

НДС=1382,97 × 0,20 = 276,59 руб.

Определяется цена научно-технической продукции с учетом НДС по формуле (8.12).

( 8.12)

Цотп=1382,97 + 276,59 = 1659,56 руб.

Все приведенные выше расчеты целесообразно объединить в сводную таблицу 8.4.

Таблица 8.4 - Расчет ориентировочной цены научно-технической продукции

|  |  |  |
| --- | --- | --- |
| Статьи затрат | Условные обозначения | Сумма, руб. |
| Материалы, покупные полуфабрикаты и комплектующие изделия | Рм | 24,00 |
| Электроэнергия | Рэ | 45,75 |
| Основная заработная плата научно-производственного персонала | Роз | 404,17 |
| Дополнительная заработная плата научно-производственного персонала | Рдз | 88,92 |
| Отчисления на социальную защиту и на страхование от несчастных случаев | Рос  Остр | 167,65  2,96 |
| Прочие прямые расходы | Рпр | 44,46 |
| Накладные расходы | Ркос | 355,67 |
| Полная себестоимость | Сп | 1133,58 |
| Плановые накопления (прибыль) | П | 249,39 |
| Отпускная цена (без НДС) | Ц | 1382,97 |
| Налог на добавленную стоимость |  | 276,59 |
| Отпускная цена с НДС | Цотп | 1659,56 |

8.4 Экономическая эффективность разработки

В данном разделе представлено экономическое обоснование для дипломного проекта по теме «Программа эмуляции полёта ракеты».

В технико-экономическом обосновании были рассмотрены следующие вопросы:

- составление плана по разработке программы;

- расчет стоимости разработки.

В первой части технико-экономического обоснования был рассчитан срок разработки по созданию программного средства. Он составил 320 часов.

Во второй части технико-экономического обоснования была рассчитана стоимость разработки программного средства, которая составила 1659,56 рублей.

Основное преимущество разработки программного средства в том, что программа в отличие от аналогов имеет современный и удобный интерфейс, низкие требования к аппаратному обеспечению и проста в использовании. Так же, в сравнении с аналогами, имеет меньшую стоимость.

Заключение

Расчет плана полёта ракеты при значительном количестве манёвров занимает много времени. В связи с этим в рамках дипломного проектирования было разработано программное средство «Maneuver.exe», эмулирующее полет ракеты. Программное средство было сделано для организации ОАО «АЛЕВКУРП» как один из программных модулей проверки работоспособности ракет или ракетных модулей. Для создания и реализации данного проекта ранее упомянутая организация предоставила математическую библиотеку для вычисления полёта ракеты.

Для достижения цели дипломного проектирования были решены следующие задачи:

– определена вычислительная система, необходимая для создания программного средства;

– разработаны логическая и физическая модели данных;

– по модели выполнена проектирование задачи;

– разработано программное средство;

– описано созданное программное средство;

– выбрана методика испытания;

– описан процесс функционального тестирования;

– приведены примеры в области применения.

Для разработки данного программного средства были применены и закреплены знания по уже ранее изученному материалу во время обучения в КБиП, были отработаны навыки владения методами надёжного программирования и эффективности разработки программного обеспечения.

В программе реализовано эмулирование полета ракеты, редактирование плана, вывод на экран, а также формирование отчетов, экспорт данных и вывод их на печать.

Программное средство разработано на языках C# и C++ в среде Visual Studio 2019.

Раздел «Охрана труда и окружающей среды» описывает систему управления охраны труда, структуру и функции охраны труда в ОАО «АЛЕВКУРП», содержит раздел об обеспечении защиты от рентгеновского излучения при эксплуатации ПЭВМ, описывает средства пожаротушения и класс помещений по ПУЭ П-IIa здания ОАО «АЛЕВКУРП», приводит мероприятия и процессы по очистке воздуха и сточных вод.

В экономическом разделе описывается и обосновывается стоимость разработки и реализации программного средства, рассчитываются экономические показатели при разработке программного средства.

Программное средство можно модифицировать, как того пожелает заказчик.

Программа готова к использованию.

Список информационных источников

1. Багласова Т.Г. Методические указания по выполнению дипломного проекта для учащихся по специальности 2-40 01 01 «Программное обеспечение технологий» / Т.Г.Багласова. – Минск: КБП, 2017. – 30 с.
2. Багласова Т.Г. Методические указания по оформлению курсовых и дипломных проектов / Т.Г.Багласова, К.О.Якимович. – Минск: КБП, 2015. – 29 с.
3. Гради Буч. Объектно - ориентированный анализ и проектирование / Гради Буч. – 2-е изд. – М.: Бином, 1998. – 720 c.
4. Крум, Э.В. Экономика предприятия. Практикум / Э. В. Крум. – Минск : Издательство Гревцова, 2009. – 360 с.
5. Экономика предприятия. Практикум / Э. В. Крум [и др.] ; под ред. Э. В. Крум. – Минск : Издательство Гревцова, 2009. – 355 с.
6. Михнюк, Т.Ф. Охрана труда / Т.Ф. Михнюк. – Минск : ИВЦ Минфина, 2009. – 365 с.
7. Постановление министерства труда и соц. защиты Республики Беларусь [Электронный ресурс] – Материал из Министерства сельского хозяйства и продовольствия Республики Беларусь – cop. 2019 – Режим доступа: <https://mshp.gov.by/print/ohrana/c495b42f27285cf5.html>
8. Рентгеновское излучение [Электронный ресурс] – femto, 2019 – Режим доступа: <http://www.femto.com.ua/articles/part_2/3422.html>. – Дата доступа : 25.05.2019.
9. ГОСТ 2.105-95 ЕСКД. Общие требования к текстовым документам. – М.: Из-во стандартов, 1995. – 10 c.
10. Руководство по WPF [Электронный ресурс] – metanit, 2019. – Режим доступа: <https://metanit.com/sharp/wpf/>. – Дата доступа : 25.05.2019.
11. Формула Циолковского [Электронный ресурс] – EpizidySpace, 2019. – Режим доступа: <http://epizodyspace.ru/bibl/ziv/2002/2/2-form-tsiol.html>. – Дата доступа : 25.05.2019.
12. Координаты вектора [Электронный ресурс] – webmath, 2019. – Режим доступа: <http://www.webmath.ru/poleznoe/formules_13_6.php>. – Дата доступа : 25.05.2019.
13. Длина вектора [Электронный ресурс] – webmath, 2019. – Режим доступа: <http://www.webmath.ru/poleznoe/formules_4_6.php>. – Дата доступа : 25.05.2019.
14. Сферические координаты [Электронный ресурс] – tutorial.math.lamar, 2019. – Режим доступа: <http://tutorial.math.lamar.edu/Classes/CalcIII/SphericalCoords.aspx>. – Дата доступа : 25.05.2019.
15. Диаграмма вариантов использования [Электронный ресурс] – anisimovkhv, 2019 – Режим доступа: <https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema12/tema12_2> – Дата доступа : 25.05.2019.
16. Диаграмма деятельности [Электронный ресурс] – it-gost, 2019 – Режим доступа: <http://it-gost.ru/articles/view_articles/96> – Дата доступа : 25.05.2019.
17. Диаграмма последовательности [Электронный ресурс] – it-gost, 2019 – Режим доступа: <http://it-gost.ru/articles/view_articles/94>. – Дата доступа : 25.05.2019.
18. Диаграмма компонентов [Электронный ресурс] – anisimovkhv, 2019 – Режим доступа: <https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema15/tema15_2>. – Дата доступа : 25.05.2019.
19. Диаграмма классов [Электронный ресурс] – informicus, 2019 – Режим доступа: <http://www.informicus.ru/default.aspx?SECTION=6&id=73&subdivisionid=3>. – Дата доступа : 25.05.2019.
20. Windows 10 [Электронный ресурс] – Microsoft, 2019 – Режим доступа: <https://www.microsoft.com/en-us/software-download/windows10>. – Дата доступа : 25.05.2019.
21. Microsoft Visual Studio [Электронный ресурс] – Microsoft 2019 – Режим доступа: <https://docs.microsoft.com/ru-ru/windows/uwp>. – Дата доступа : 25.05.2019.
22. C# [Электронный ресурс] – Microsoft, 2019 – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/>. – Дата доступа : 25.05.2019.
23. HelpNDoc [Электронный ресурс] – htmleditors, 2019 – Режим доступа: <http://htmleditors.ru/Rasnoe/help/list1/HelpNDoc.html>. – Дата доступа : 25.05.2019.
24. Umlet [Электронный ресурс] – freeanalogs, 2019 – Режим доступа: <https://freeanalogs.ru/UMLet>. – Дата доступа : 25.05.2019.
25. InstallShield [Электронный ресурс] – Microsoft 2019 – Режим доступа: <https://ru.wikipedia.org/wiki/InstallShield>. – Дата доступа : 25.05.2019.

Приложение А

(обязательное)

Текст программы

using System;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

using System.Collections.Generic;

using SharpGL;

using Maneuver.Scene;

using Maneuver.Scene.Components;

namespace Maneuver.Helpers

{

/// <summary>

/// Helper for color-pick method

/// </summary>

public static class ColorPickerHelper

{

private static OpenGLControl Canvas = new OpenGLControl();

private static Dictionary<System.Windows.Media.Color, SceneElement> SelectionObjectColorsPairs = null;

private static Camera camera = null;

private static void PickRenderOnPickRenderControl(object sender, RenderEventArgs args)

{

var objects\_and\_colors = SelectionObjectColorsPairs;

var gl = Canvas.OpenGL;

gl.Clear(OpenGL.GL\_COLOR\_BUFFER\_BIT | OpenGL.GL\_DEPTH\_BUFFER\_BIT);

gl.MatrixMode(SharpGL.Enumerations.MatrixMode.Projection);

gl.LoadIdentity();

int[] viewport\_size = new int[4];

gl.GetInteger(OpenGL.GL\_VIEWPORT, viewport\_size);

float x = (float)viewport\_size[2] / viewport\_size[3];

gl.Perspective(45, x, 0.1, double.MaxValue);

camera.LookAt(gl);

foreach (var key\_value in objects\_and\_colors)

key\_value.Value.ColorPickRender(gl, key\_value.Key);

gl.Flush();

}

/// <summary>

/// Perform selection by color from scene

/// </summary>

/// <param name="viewport">Original <see cref="OpenGLControl"/></param>

/// <param name="SelectionObjectColorsPairs">Dictionary of colors and array <see cref="SceneElement"/></param>

/// <param name="camera">Original <see cref="Camera"/> of <see cref="ManeuverScene"/></param>

/// <param name="mouse\_x">Mouse X position on click</param>

/// <param name="mouse\_y">Mouse Y position on click</param>

/// <exception cref="ArgumentNullException"></exception>

/// <exception cref="ArgumentOutOfRangeException"></exception>

/// <returns>Clicked point color</returns>

public static System.Windows.Media.Color PerformSelection(in OpenGLControl viewport, in Dictionary<System.Windows.Media.Color, SceneElement> SelectionObjectColorsPairs, in Camera camera, int mouse\_x, int mouse\_y)

{

if (viewport is null || SelectionObjectColorsPairs is null || camera is null)

throw new ArgumentNullException();

if (mouse\_x < 0 || mouse\_y < 0)

throw new ArgumentOutOfRangeException();

Canvas.Height = viewport.Height;

Canvas.Width = viewport.Width;

Canvas.Location = viewport.Location;

ColorPickerHelper.camera = camera;

ColorPickerHelper.SelectionObjectColorsPairs = SelectionObjectColorsPairs;

Canvas.OpenGLDraw += PickRenderOnPickRenderControl;

Canvas.DoRender();

Canvas.OpenGLDraw -= PickRenderOnPickRenderControl;

byte[] color = new byte[4];

int[] viewport\_sizies = new int[4];

Canvas.OpenGL.GetInteger(OpenGL.GL\_VIEWPORT, viewport\_sizies); // получение размеров окна

Canvas.OpenGL.ReadPixels(mouse\_x, viewport\_sizies[3] - mouse\_y, 1, 1, OpenGL.GL\_RGBA, OpenGL.GL\_UNSIGNED\_BYTE, color);

System.Windows.Media.Color color\_result = new System.Windows.Media.Color()

{

R = color[0],

G = color[1],

B = color[2],

A = 1

};

return color\_result;

}

}

}

namespace Maneuver.Helpers

{

/// <summary>

/// Contains extended methods for classes

/// </summary>

public static class ExtendedMethodsHelper

{

public static bool IsFloat(this string str)

=> float.TryParse(str, out float value);

public static float ToFloat(this string str)

=> float.Parse(str);

}

}

using System;

using System.IO;

using System.Collections.Generic;

using ManeuverMath.Perform;

using Maneuver.Scene.Components;

using Maneuver.Scene.Components.Elements;

namespace Maneuver.Helpers

{

/// <summary>

/// Helper for computation

/// </summary>

public static class PerformHelper

{

/// <summary>

/// Fills the array with the specified value

/// </summary>

/// <typeparam name="T">Array and value type</typeparam>

/// <param name="arr">Fillable array</param>

/// <param name="fillValue">Value for fill</param>

public static void FillArray<T>(ref T[] arr, T fillValue)

{

int i = 0;

if (arr.Length > 16)

{

while (i < 16)

arr[i++] = fillValue;

while (i + 16 < arr.Length)

{

Array.Copy(arr, 0, arr, i, 16);

i += 16;

}

}

while (i < arr.Length)

arr[i++] = fillValue;

}

/// <summary>

/// Fills the array with the specified array

/// </summary>

/// <typeparam name="T">Arrays type</typeparam>

/// <param name="arr">Fillable array</param>

/// <param name="fillArr">Array for fill</param>

/// <exception cref="System.ArgumentException"></exception>

public static void FillArrayByArray<T>(ref T[] arr, T[] fillArr)

{

if (arr.Length < fillArr.Length)

throw new ArgumentException("fill arr is greater then ref arr");

else if (arr.Length % fillArr.Length != 0)

throw new ArgumentException("fill arr length is not multiple fillArr length");

Array.Copy(fillArr, arr, fillArr.Length);

int arrayToFillHalfLength = arr.Length / 2;

for (int i = fillArr.Length; i < arr.Length; i \*= 2)

{

int copyLength = i;

if (i > arrayToFillHalfLength)

{

copyLength = arr.Length - i;

}

Array.Copy(arr, 0, arr, i, copyLength);

}

}

public static string ReadConentFromFile(string path)

{

string content = "";

using (StreamReader reader = new StreamReader(path))

{

content = reader.ReadToEnd();

}

return content;

}

public static Func<float, float> CreateActionByButtonName(string buttonName)

{

int mask = 15;

string command = buttonName.Remove(0, mask);

int value = Convert.ToInt32(command.Remove(0, 1));

char type = buttonName[0];

switch (command[0])

{

case 'm': // mul

return x => x \* value;

case 'd': // div

return x => x / value;

case 'p': // plus

return x => x + value;

case 's': // sub

return x => x - value;

case 'e': // equal, set

return x => value;

default:

return null;

}

}

public static (Point3DFloat position, ulong time)[] CreateManeuverPlan(MathManeuverPoint[] maneuverPoints)

{

ManeuverSolution solution = new ManeuverSolution(maneuverPoints);

var flight\_points = new LinkedList<(Point3DFloat position, ulong time)>();

ulong currentTime = 1; // time in nanoseconds

const double nanoToMilliDivisor = 1000.0;

while (true)

{

var fly\_info = solution.PerformMotion(currentTime / nanoToMilliDivisor);

if (fly\_info is null) break; // is end

flight\_points.AddLast(

(new Point3DFloat((float)fly\_info.X, (float)fly\_info.Y, (float)fly\_info.Z),

currentTime)

);

currentTime += (ulong)nanoToMilliDivisor;

}

var plan = new (Point3DFloat position, ulong time)[flight\_points.Count];

flight\_points.CopyTo(plan, 0);

return plan;

}

}

}

using System.Windows.Forms;

using SharpGL;

namespace Maneuver.Helpers

{

/// <summary>

/// Helper to work with program

/// </summary>

public static class ProgramStateHelper

{

public static OpenGLControl OpenGLControl { get; set; }

/// <summary>

/// Mouse sensitivity

/// </summary>

public static float MouseKoef { get; set; } = 0.01f;

/// <summary>

/// Last cursor pressed position

/// </summary>

public static System.Windows.Point LastMouseCursorPressedPosition { get; set; } = default;

/// <summary>

/// Last mouse pressed key

/// </summary>

public static MouseButtons LastPressedKey { get; set; } = MouseButtons.None;

/// <summary>

/// Is mouse key pressed

/// </summary>

public static bool IsMouseKeyPressed { get; set; } = false;

public static void UpdateOpenGLView()

{

if (OpenGLControl is null)

return;

OpenGLControl.Refresh();

}

}

}

using Maneuver.Scene.Primitives;

using System.IO;

using ObjLoader;

using ObjLoader.Loader;

using ObjLoader.Loader.Loaders;

using ObjLoader.Loader.Data.VertexData;

using Maneuver.Scene.Primitives.ExtendedPrimitives;

using System.Collections.Generic;

namespace Maneuver.Helpers

{

public static class RocketLoader

{

public static Rocket LoadRocketFromFile(string file)

{

var objLoaderFactory = new ObjLoaderFactory();

var objLoader = objLoaderFactory.Create();

var fileStream = new FileStream(file, FileMode.Open);

var result = objLoader.Load(fileStream);

var vertices = result.Vertices.ToArray();

var normals = result.Normals.ToArray();

var texcoords = result.Textures.ToArray();

List<uint> indices = new List<uint>();

var findedVertices = new Dictionary<Vertex, uint>();

uint index = 0;

#region create indices and sort vertices

for (int i = 0; i < vertices.Length; i++)

{

if (findedVertices.ContainsKey(vertices[i]))

indices.Add(findedVertices[vertices[i]]);

else

{

findedVertices.Add(vertices[i], index);

indices.Add(index++);

}

}

Vertex[] vBuffer = new Vertex[findedVertices.Count];

foreach (var item in findedVertices)

vBuffer[item.Value] = item.Key;

vertices = vBuffer;

#endregion

float[] fVertices = new float[vertices.Length \* RocketLoader.SizeOf<Vertex>()],

fNormals = new float[normals.Length \* RocketLoader.SizeOf<Normal>()],

fTexcoords = new float[texcoords.Length \* RocketLoader.SizeOf<Texture>()];

uint[] uIndices = indices.ToArray();

index = 0;

for(int i = 0; i < fVertices.Length; i += 3, index++)

{

fVertices[i] = vertices[index].X;

fVertices[i + 1] = vertices[index].Y;

fVertices[i + 2] = vertices[index].Z;

}

index = 0;

for (int i = 0; i < fNormals.Length; i += 3, index++)

{

fNormals[i] = normals[index].X;

fNormals[i + 1] = normals[index].Y;

fNormals[i + 2] = normals[index].Z;

}

index = 0;

for (int i = 0; i < fTexcoords.Length; i += 2, index++)

{

fTexcoords[i] = texcoords[index].X;

fTexcoords[i + 1] = texcoords[index].Y;

}

Rocket rocket = new Rocket(fVertices, fNormals, fTexcoords, uIndices);

return rocket;

}

private static int SizeOf<T>() where T : struct

{

return System.Runtime.InteropServices.Marshal.SizeOf<T>() / 4;

}

}

}

using System;

namespace Maneuver.Helpers

{

/// <summary>

/// Helper for spherical system computation

/// </summary>

static class SphericalSystemHelper

{

/// <summary>

/// Calculate X position in 3 dimensions

/// </summary>

/// <param name="len">radius</param>

/// <param name="theta">angle "theta"</param>

/// <param name="phi">angle "phi"</param>

/// <returns></returns>

public static double Calculate3DXPointPosition(double len, double theta, double phi)

=> len \* Math.Sin(theta) \* Math.Cos(phi);

/// <summary>

/// Calculate Y position in 3 dimensions

/// </summary>

/// <param name="len">radius</param>

/// <param name="theta">angle "theta"</param>

/// <param name="phi">angle "phi"</param>

/// <returns></returns>

public static double Calculate3DYPointPosition(double len, double theta, double phi)

=> len \* Math.Sin(theta) \* Math.Sin(phi);

/// <summary>

/// Calculate Z position in 3 dimensions

/// </summary>

/// <param name="len">radius</param>

/// <param name="theta">angle "theta"</param>

/// <param name="phi">angle "phi"</param>

/// <returns></returns>

public static double Calculate3DZPointPosition(double len, double theta)

=> len \* Math.Cos(theta);

/// <summary>

/// Read the name, idiot

/// </summary>

/// <param name="positionCoordinate">position</param>

/// <param name="targetCoordinate">target position</param>

/// <param name="scale">scale value</param>

/// <returns></returns>

public static float CalculateScaledNormalizedCoordinate(float positionCoordinate, float targetCoordinate, float scale)

=> positionCoordinate > targetCoordinate ?

(positionCoordinate - targetCoordinate) \* scale + targetCoordinate :

(targetCoordinate + positionCoordinate) \* scale + targetCoordinate;

/// <summary>

/// Read the name, idiot

/// </summary>

/// <param name="positionCoordinate">position</param>

/// <param name="targetCoordinate">target position</param>

/// <param name="scale">scale value</param>

/// <returns></returns>

public static float CalculateScaledCoordinate(float positionCoordinate, float targetCoordinate, float scale)

=> (targetCoordinate - positionCoordinate) \* scale + positionCoordinate;

}

}

using System;

using System.Diagnostics;

using System.Collections.Generic;

using System.Collections;

namespace Maneuver.Helpers

{

public static class Utility

{

public static ulong ElapsedNanoSeconds(this Stopwatch stopwatch)

{

ulong nanoseconds = (ulong)Math.Pow(10, 9) \* (ulong)stopwatch.ElapsedTicks / (ulong)Stopwatch.Frequency;

return nanoseconds;

}

public static bool Contains<T>(this LinkedList<T> list, Func<T, bool> rule)

{

foreach (var item in list)

{

if (rule(item) == true)

return true;

}

return false;

}

public static int Count<T>(this LinkedList<T> list, Func<T, bool> rule)

{

int totalCount = 0;

foreach (var item in list)

{

if (rule(item) == true)

totalCount++;

}

return totalCount;

}

public static T ElementAt<T>(this LinkedList<T> \_list, int position)

{

LinkedListNode<T> mark = \_list.First;

int i = 0;

while (i < position)

{

mark = mark.Next;

i++;

}

return mark.Value;

}

public static T[] ToArray<T>(this IList<T> list) where T : new()

{

T[] arr = new T[list.Count];

list.CopyTo(arr, 0);

return arr;

}

}

}

using System;

using Maneuver.Scene.Primitives;

using Maneuver.Scene.Primitives.ExtendedPrimitives;

namespace Maneuver.Scene.Components.Elements

{

/// <summary>

/// Represents the kompas of <see cref="ManeuverScene"/>/>

/// </summary>

public class Kompas : SceneElement

{

/// <summary>

/// Set <see cref="SceneElement.ViewObject"/> in <paramref name="viewObject"/> and <see cref="SceneElement.Location"/> in (x, y, z) = (0, 0, 0)

/// </summary>

/// <param name="viewObject">Renderable object of <see cref="Kompas"/></param>

/// /// <exception cref="ArgumentNullException"></exception>

public Kompas(IRenderablePrimitive viewObject) : base(viewObject)

{

if (viewObject is Circle2DArray circle)

this.\_radius = circle.Radius;

}

/// <summary>

/// Set <see cref="SceneElement.ViewObject"/> in <paramref name="viewObject"/> and <see cref="SceneElement.Location"/> in <paramref name="location"/>

/// </summary>

/// <param name="viewObject"><see cref="SceneElement.ViewObject"/> of <see cref="Kompas"/></param>

/// <param name="location"><see cref="SceneElement.Location"/> of <see cref="Kompas"/></param>

/// <exception cref="ArgumentNullException"></exception>

public Kompas(IRenderablePrimitive viewObject, Point3DFloat location) : base(viewObject, location)

{

if (viewObject is Circle2DArray circle)

this.\_radius = circle.Radius;

}

/// <summary>

/// Create <see cref="Kompas"/> with

/// </summary>

public Kompas() : base(new Circle2DArray(100, 10), new Point3DFloat(0, 0, 0)) { \_radius = 100; }

private float \_radius = 100;

public float Radius

{

get => \_radius;

}

}

}

using System;

using System.ComponentModel;

using Maneuver.Scene.Primitives;

using Maneuver.Scene.ManeuverManagement;

namespace Maneuver.Scene.Components.Elements

{

/// <summary>

/// Represents the object-maneuver-point/>

/// </summary>

public class ManeuverPoint : SceneElement

{

// public event PropertyChangedEventHandler PropertyChanged;

// public void OnPropetyChanged(PropertyChangedEventArgs e) => PropertyChanged?.Invoke(this, e);

private ManeuverType maneuver = ManeuverType.None;

/// <summary>

/// Set <see cref="SceneElement.ViewObject"/> in <paramref name="viewObject"/> and <see cref="SceneElement.Location"/> in (x, y, z) = (0, 0, 0)

/// </summary>

/// <param name="viewObject">Renderable object of <see cref="ManeuverPoint"/></param>

/// /// <exception cref="ArgumentNullException"></exception>

public ManeuverPoint(IRenderablePrimitive viewObject) : base(viewObject) { }

/// <summary>

/// Set <see cref="SceneElement.ViewObject"/> in <paramref name="viewObject"/> and <see cref="SceneElement.Location"/> in <paramref name="location"/>

/// </summary>

/// <param name="viewObject"><see cref="SceneElement.ViewObject"/> of <see cref="ManeuverPoint"/></param>

/// <param name="location"><see cref="SceneElement.Location"/> of <see cref="ManeuverPoint"/></param>

/// <exception cref="ArgumentNullException"></exception>

public ManeuverPoint(IRenderablePrimitive viewObject, Point3DFloat location) : base(viewObject, location) { }

public ManeuverPoint() : base() { }

/// <summary>

/// <see cref="ManeuverType"/> of <see cref="ManeuverPoint"/>

/// </summary>

public ManeuverType Maneuver

{

get => maneuver;

set

{

if (maneuver == value)

return;

maneuver = value;

OnPropetyChanged(new PropertyChangedEventArgs("Maneuver"));

}

}

}

}

using System;

using SharpGL;

namespace Maneuver.Scene.Components

{

/// <summary>

/// Represents the camera of <seealso cref="ManeuverScene"/>

/// </summary>

public class Camera

{

/// <summary>

/// Is eye position

/// </summary>

public Point3DFloat Position { get; set; }

/// <summary>

/// Is target position

/// </summary>

public Point3DFloat Target { get; set; }

/// <summary>

/// Create default camera

/// </summary>

public Camera()

{

Position = new Point3DFloat();

Target = new Point3DFloat(5, 5, 5);

}

/// <summary>

/// Create camera with with given <see cref="Camera.Position"/> and <seealso cref="Camera.Target"/>

/// </summary>

/// <param name="position"><see cref="Camera.Position"/> value</param>

/// <param name="target"><see cref="Camera.Target"/> value</param>

public Camera(Point3DFloat position, Point3DFloat target)

{

if (position is null || target is null)

throw new ArgumentNullException();

this.Position = position;

this.Target = target;

}

/// <summary>

/// Peform method <seealso cref="OpenGL.LookAt"/> with <see cref="Camera.Position"/> and <see cref="Camera.Target"/> values

/// </summary>

/// <param name="context">It is <see cref="OpenGL"/> context</param>

public void LookAt(OpenGL context)

=> context.LookAt(Position.X, Position.Y, Position.Z, Target.X, Target.Y, Target.Z, 0, 0, 1);

/// <summary>

/// Set <see cref="Camera.Position"/> with given values

/// </summary>

/// <param name="x"></param>

/// <param name="y"></param>

/// <param name="z"></param>

public void SetPosition(float x, float y, float z)

{

Position.X = x;

Position.Y = y;

Position.Z = z;

}

/// <summary>

/// Set <see cref="Camera.Target"/> with given values

/// </summary>

/// <param name="x"></param>

/// <param name="y"></param>

/// <param name="z"></param>

public void SetTarget(float x, float y, float z)

{

Target.X = x;

Target.Y = y;

Target.Z = z;

}

public Camera Clone()

{

Camera camera = new Camera(this.Position.Clone(), this.Target.Clone());

return camera;

}

}

}

using System.ComponentModel;

using Maneuver.Helpers;

namespace Maneuver.Scene.Components

{

/// <summary>

/// Represents the point of 3 dimensions (x, y, z) with <see cref="System.Single"/> values

/// </summary>

public class Point3DFloat : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

public void OnPropetyChanged(PropertyChangedEventArgs e) => PropertyChanged?.Invoke(this, e);

float x, y, z;

public float X

{

get => x;

set

{

if (x == value)

return;

x = value;

OnPropetyChanged(new PropertyChangedEventArgs("X"));

}

}

public float Y

{

get => y;

set

{

if (y == value)

return;

y = value;

OnPropetyChanged(new PropertyChangedEventArgs("Y"));

}

}

public float Z

{

get => z;

set

{

if (z == value)

return;

z = value;

OnPropetyChanged(new PropertyChangedEventArgs("Z"));

}

}

public Point3DFloat() => x = y = z = 0;

public Point3DFloat(float x, float y, float z)

{

this.x = x;

this.y = y;

this.z = z;

}

/// <summary>

/// Moves <seealso cref="Point3DFloat.X"/>, <seealso cref="Point3DFloat.Y"/> and <seealso cref="Point3DFloat.Z"/> to given values

/// </summary>

/// <param name="x\_offset">movement <seealso cref="Point3DFloat.X"/> value</param>

/// <param name="y\_offset">movement <seealso cref="Point3DFloat.Y"/> value</param>

/// <param name="z\_offset">movement <seealso cref="Point3DFloat.Z"/> value</param>

public void Move(float x\_offset, float y\_offset, float z\_offset)

{

x += x\_offset;

y += y\_offset;

z += z\_offset;

}

/// <summary>

/// Set <seealso cref="Point3DFloat.X"/>, <seealso cref="Point3DFloat.Y"/> and <seealso cref="Point3DFloat.Z"/> to given values

/// </summary>

/// <param name="x\_offset"><seealso cref="Point3DFloat.X"/> value</param>

/// <param name="y\_offset"><seealso cref="Point3DFloat.Y"/> value</param>

/// <param name="z\_offset"><seealso cref="Point3DFloat.Z"/> value</param>

public void Set(float x, float y, float z)

{

this.x = x;

this.y = y;

this.z = z;

}

public Point3DFloat Clone()

{

Point3DFloat point = new Point3DFloat(this.x, this.y, this.z);

return point;

}

}

/// <summary>

/// Represents the point of 3 dimensions (x, y, z) with <see cref="System.Int32"/> values

/// </summary>

public class Point3DInt

{

int x, y, z;

public int X => x;

public int Y => y;

public int Z => z;

public Point3DInt() => x = y = z = 0;

public Point3DInt(int x, int y, int z)

{

this.x = x;

this.y = y;

this.z = z;

}

/// <summary>

/// Moves <seealso cref="Point3DFloat.X"/>, <seealso cref="Point3DFloat.Y"/> and <seealso cref="Point3DFloat.Z"/> to given values

/// </summary>

/// <param name="x\_offset">movement <seealso cref="Point3DFloat.X"/> value</param>

/// <param name="y\_offset">movement <seealso cref="Point3DFloat.Y"/> value</param>

/// <param name="z\_offset">movement <seealso cref="Point3DFloat.Z"/> value</param>

public void Move(int x\_offset, int y\_offset, int z\_offset)

{

x += x\_offset;

y += y\_offset;

z += z\_offset;

}

/// <summary>

/// Set <seealso cref="Point3DFloat.X"/>, <seealso cref="Point3DFloat.Y"/> and <seealso cref="Point3DFloat.Z"/> to given values

/// </summary>

/// <param name="x\_offset"><seealso cref="Point3DFloat.X"/> value</param>

/// <param name="y\_offset"><seealso cref="Point3DFloat.Y"/> value</param>

/// <param name="z\_offset"><seealso cref="Point3DFloat.Z"/> value</param>

public void Set(int x, int y, int z)

{

this.x = x;

this.y = y;

this.z = z;

}

}

}

using System;

using System.ComponentModel;

using SharpGL;

using Maneuver.Scene.Primitives;

namespace Maneuver.Scene.Components

{

/// <summary>

/// Represents the element of <seealso cref="ManeuverScene"/>

/// </summary>

public abstract class SceneElement : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

public void OnPropetyChanged(PropertyChangedEventArgs e) => PropertyChanged?.Invoke(this, e);

private Point3DFloat location = default;

private IRenderablePrimitive viewObject = default;

public SceneElement() { }

/// <summary>

/// Set <see cref="SceneElement.ViewObject"/> in <paramref name="viewObject"/> and <see cref="SceneElement.Location"/> in (x, y, z) = (0, 0, 0)

/// </summary>

/// <param name="viewObject">Renderable object of <see cref="SceneElement"/></param>

/// /// <exception cref="ArgumentNullException"></exception>

public SceneElement(IRenderablePrimitive viewObject)

{

if (viewObject is null)

throw new ArgumentNullException();

this.ViewObject = viewObject;

this.Location = new Point3DFloat(0, 0, 0);

}

/// <summary>

/// Set <see cref="SceneElement.ViewObject"/> in <paramref name="viewObject"/> and <see cref="SceneElement.Location"/> in <paramref name="location"/>

/// </summary>

/// <param name="viewObject"><see cref="SceneElement.ViewObject"/> of <see cref="SceneElement"/></param>

/// <param name="location"><see cref="SceneElement.Location"/> of <see cref="SceneElement"/></param>

/// <exception cref="ArgumentNullException"></exception>

public SceneElement(IRenderablePrimitive viewObject, Point3DFloat location)

{

if (viewObject is null || location is null)

throw new ArgumentNullException();

this.ViewObject = viewObject;

this.Location = location;

}

/// <summary>

/// The location of element

/// </summary>

[TypeConverter(typeof(ExpandableObjectConverter))]

public Point3DFloat Location

{

get => location;

set

{

if (location == value)

return;

location = value;

OnPropetyChanged(new PropertyChangedEventArgs("Location"));

}

}

/// <summary>

/// The rendered primitive of the scene element

/// </summary>

[TypeConverter(typeof(ExpandableObjectConverter))]

public IRenderablePrimitive ViewObject

{

get => viewObject;

set

{

if (viewObject == value)

return;

viewObject = value;

OnPropetyChanged(new PropertyChangedEventArgs("ViewObject"));

}

}

/// <summary>

/// Perform the render of element. Call <see cref="IRenderablePrimitive.Draw"/> of <see cref="SceneElement.ViewObject"/> object

/// </summary>

/// <param name="context">It is <see cref="OpenGL"/> context</param>

public void Render(OpenGL context)

{

if (viewObject != null)

viewObject.Draw(context, Location);

}

/// <summary>

/// Perform color-pick render

/// </summary>

/// <param name="context">It is <see cref="OpenGL"/> context</param>

/// <param name="color">Color-pick color</param>

public void ColorPickRender(OpenGL context, System.Windows.Media.Color color)

{

byte[] rgba = { color.R, color.G, color.B, color.A };

this.ColorPickRender(context, rgba, Location);

}

/// <summary>

/// Perform color-pick render

/// </summary>

/// <param name="context">It is <see cref="OpenGL"/> context</param>

/// <param name="rgba">It is array {r, g, b, a} array</param>

public void ColorPickRender(OpenGL context, byte[] rgba, Point3DFloat location)

{

if(ViewObject is ISelectedPrimitive selectModeView)

selectModeView.ColorPickDraw(context, rgba, Location);

}

}

}

namespace Maneuver.Scene.ManeuverManagement

{

public enum ManeuverType

{

/// <summary>

/// Равномерное постоянное движение

/// </summary>

UniformConstantMotionType = 1,

/// <summary>

/// Ускоренное движение

/// </summary>

UniformAcceleratedMotionType = 2,

/// <summary>

/// Горизонтальный поворот

/// </summary>

HorizontalTurnMotionType = 3,

/// <summary>

/// Вертикальный поворот

/// </summary>

VerticalTurnMotionType = 4,

/// <summary>

/// Горизонтальное движение змейкой

/// </summary>

HorizontalSnakeMotionType = 5,

/// <summary>

/// Вертикальное движение змейкой

/// </summary>

VerticalSnakeMotionType = 6,

/// <summary>

/// Отсутствующий маневр

/// </summary>

None = 7

}

}

using System;

using System.ComponentModel;

using SharpGL;

using Maneuver.Scene.Components;

namespace Maneuver.Scene.Primitives

{

public class SolidSphere : IRenderablePrimitive, ISelectedPrimitive, INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

public void OnPropetyChanged(PropertyChangedEventArgs e) => PropertyChanged?.Invoke(this, e);

private float[] vertices;

private float[] normals;

private float[] texcoords;

private uint[] indices;

private float[] colors;

private float \_radius;

private int \_rings, \_sectors;

public float Radius

{

get => \_radius;

set

{

\_radius = value;

this.Generate(\_radius, \_rings, \_sectors);

OnPropetyChanged(new PropertyChangedEventArgs("Radius"));

}

}

public int Rings

{

get => \_rings;

set

{

\_rings = value;

this.Generate(\_radius, \_rings, \_sectors);

OnPropetyChanged(new PropertyChangedEventArgs("Rings"));

}

}

public int Sectors

{

get => \_sectors;

set

{

\_sectors = value;

this.Generate(\_radius, \_rings, \_sectors);

OnPropetyChanged(new PropertyChangedEventArgs("Sectors"));

}

}

/// <summary>

/// Init sphere with preset radius, ringss and sectors

/// </summary>

/// <param name="radius"></param>

/// <param name="rings"></param>

/// <param name="sectors"></param>

/// <param name="location"></param>

public SolidSphere(float radius, int rings, int sectors)

{

\_sectors = sectors;

\_rings = rings;

\_radius = radius;

Generate(radius, rings, sectors);

}

/// <summary>

/// Perform color-only draw

/// </summary>

/// <param name="context">It is <see cref="OpenGL"/> context</param>

/// <param name="rgba">It is {r, g, b, a} array</param>

/// <param name="location">Location for render</param>

public void ColorPickDraw(OpenGL context, byte[] rgba, Point3DFloat location)

{

var gl = context;

byte[] pick\_colors = new byte[colors.Length];

Maneuver.Helpers.PerformHelper.FillArrayByArray(ref pick\_colors, rgba);

gl.PolygonMode(OpenGL.GL\_FRONT\_AND\_BACK, OpenGL.GL\_FILL);

gl.PushMatrix();

gl.Translate(location.X, location.Y, location.Z);

gl.Rotate(90, 0, 0);

gl.EnableClientState(OpenGL.GL\_VERTEX\_ARRAY);

gl.EnableClientState(OpenGL.GL\_NORMAL\_ARRAY);

gl.EnableClientState(OpenGL.GL\_TEXTURE\_COORD\_ARRAY);

gl.EnableClientState(OpenGL.GL\_COLOR\_ARRAY);

unsafe

{

fixed (byte\* p = pick\_colors)

{

IntPtr ptr = (IntPtr)p;

gl.ColorPointer(4, OpenGL.GL\_UNSIGNED\_BYTE, 0, ptr);

}

}

gl.VertexPointer(3, 0, vertices);

gl.NormalPointer(OpenGL.GL\_FLOAT, 0, normals);

gl.TexCoordPointer(2, OpenGL.GL\_FLOAT, 0, vertices);

// gl.DrawElements(OpenGL.GL\_QUADS, indices.Length, indices); - в идеале, но почему-то баги

gl.DrawElements(OpenGL.GL\_QUADS, indices.Length - 50, indices);

gl.DisableClientState(OpenGL.GL\_VERTEX\_ARRAY);

gl.DisableClientState(OpenGL.GL\_NORMAL\_ARRAY);

gl.DisableClientState(OpenGL.GL\_TEXTURE\_COORD\_ARRAY);

gl.PopMatrix();

}

/// <summary>

/// Renders an object in the specified <see cref="OpenGL"/> context

/// </summary>

/// <param name="context">It is <see cref="OpenGL"/> context</param>

/// <param name="location">Location for rende</param>

public void Draw(OpenGL context, Point3DFloat location)

{

var gl = context;

gl.PolygonMode(OpenGL.GL\_FRONT\_AND\_BACK, OpenGL.GL\_LINE);

gl.PushMatrix();

gl.Translate(location.X, location.Y, location.Z);

gl.Rotate(90, 0, 0);

gl.EnableClientState(OpenGL.GL\_VERTEX\_ARRAY);

gl.EnableClientState(OpenGL.GL\_NORMAL\_ARRAY);

gl.EnableClientState(OpenGL.GL\_TEXTURE\_COORD\_ARRAY);

gl.EnableClientState(OpenGL.GL\_COLOR\_ARRAY);

unsafe

{

fixed (float\* p = colors)

{

IntPtr ptr = (IntPtr)p;

gl.ColorPointer(4, OpenGL.GL\_FLOAT, 0, ptr);

}

}

gl.VertexPointer(3, 0, vertices);

gl.NormalPointer(OpenGL.GL\_FLOAT, 0, normals);

gl.TexCoordPointer(2, OpenGL.GL\_FLOAT, 0, vertices);

// gl.DrawElements(OpenGL.GL\_QUADS, indices.Length, indices); - в идеале, но почему-то баги

gl.DrawElements(OpenGL.GL\_QUADS, indices.Length - 50, indices);

gl.DisableClientState(OpenGL.GL\_VERTEX\_ARRAY);

gl.DisableClientState(OpenGL.GL\_NORMAL\_ARRAY);

gl.DisableClientState(OpenGL.GL\_TEXTURE\_COORD\_ARRAY);

gl.PopMatrix();

}

/// <summary>

/// Perform generate

/// </summary>

/// <param name="radius"></param>

/// <param name="rings"></param>

/// <param name="sectors"></param>

private void Generate(float radius, int rings, int sectors)

{

float m\_pi\_2 = (float)Math.PI / 2f;

float R = 1f / (rings - 1);

float S = 1f / (sectors - 1);

int r = 0, s = 0;

vertices = new float[rings \* sectors \* 3];

normals = new float[rings \* sectors \* 3];

texcoords = new float[rings \* sectors \* 2];

int v\_index = 0, n\_index = 0, t\_index = 0;

for (r = 0; r < rings; r++)

{

for (s = 0; s < sectors; s++)

{

float y = (float)Math.Sin(-m\_pi\_2 + Math.PI \* r \* R);

float x = (float)(Math.Cos(2f \* Math.PI \* s \* S) \* Math.Sin(Math.PI \* r \* R));

float z = (float)(Math.Sin(2f \* Math.PI \* s \* S) \* Math.Sin(Math.PI \* r \* R));

texcoords[t\_index++] = s \* S;

texcoords[t\_index++] = r \* R;

vertices[v\_index++] = x \* radius;

vertices[v\_index++] = y \* radius;

vertices[v\_index++] = z \* radius;

normals[n\_index++] = -x;

normals[n\_index++] = -y;

normals[n\_index++] = -z;

}

}

indices = new uint[rings \* sectors \* 4];

int in\_index = 0;

for (r = 0; r < rings; r++)

{

for (s = 0; s < sectors; s++)

{

indices[in\_index++] = (uint)((r + 1) \* sectors + s);

indices[in\_index++] = (uint)((r + 1) \* sectors + (s + 1));

indices[in\_index++] = (uint)(r \* sectors + (s + 1));

indices[in\_index++] = (uint)(r \* sectors + s);

}

}

colors = new float[vertices.Length \* 4];

for (int i = 0; i < colors.Length; i += 4)

{

/\*colors[i] = 0f;

colors[i + 1] = 1f;

colors[i + 2] = 1f;

colors[i + 3] = 1f;\*/

colors[i] = 1f;

colors[i + 1] = 0.4f;

colors[i + 2] = 0.6f;

colors[i + 3] = 1f;

}

}

}

}

using System;

using SharpGL;

using Maneuver.Scene.Components;

namespace Maneuver.Scene.Primitives

{

/// <summary>

/// Represents the circle of <see cref="ManeuverScene"/>/>

/// </summary>

public class Circle : IRenderablePrimitive

{

/// <summary>

/// Vertex array

/// </summary>

private Point3DFloat[] vertexes = null;

public Circle() { }

/// <summary>

/// Creates a circle with a given radius and number of sides

/// </summary>

/// <param name="radius">radius of circle</param>

/// <param name="slides">number of sides</param>

public Circle(float radius, int slides)

=> Generate(radius, slides);

/// <summary>

/// Generate a circle with a given radius and number of sides and set to this object

/// </summary>

/// <param name="radius">radius of circle</param>

/// <param name="slides">number of sides</param>

public void Generate(float radius, int slides)

{

vertexes = new Point3DFloat[slides];

for (int i = 0; i < slides; i++)

{

float theta = 2.0f \* (float)Math.PI \* i / (float)slides;

float x = radius \* (float)Math.Cos(theta);

float y = radius \* (float)Math.Sin(theta);

Point3DFloat vertex = new Point3DFloat()

{

X = x,

Y = y,

Z = 0

};

vertexes[i] = vertex;

}

}

public void Draw(OpenGL context, Point3DFloat location)

{

if (vertexes == null)

return;

var gl = context;

gl.Translate(location.X, location.Y, location.Z);

gl.Begin(OpenGL.GL\_LINE\_LOOP);

for (int i = 0; i < vertexes.Length; i++)

{

gl.Color(1f, 1f, 1f);

gl.Vertex(vertexes[i].X, vertexes[i].Y, vertexes[i].Z);

}

gl.End();

}

}

}

using System.Collections.Generic;

using SharpGL;

using Maneuver.Scene.Components;

namespace Maneuver.Scene.Primitives.ExtendedPrimitives

{

public class Circle2DArray : IRenderablePrimitive

{

/// <summary>

/// <see cref="Circle"/> array of <see cref="Circle2DArray"/>

/// </summary>

public List<Circle> Circles { get; set; } = new List<Circle>();

public Circle2DArray() { }

/// <summary>

/// Creates a <see cref="Circle2DArray"/> with a given radius and number of cirlcles

/// </summary>

/// <param name="radius"></param>

/// <param name="circleCount"></param>

public Circle2DArray(float radius, int circleCount)

{

this.Generate(radius, circleCount);

this.Radius = radius;

}

/// <summary>

/// Generate new <see cref="Circle2DArray"/>

/// </summary>

/// <param name="radius"></param>

/// <param name="circleCount"></param>

public void Generate(float radius, int circleCount)

{

float step = radius / circleCount;

float step\_point = step;

for (int i = 0; i < circleCount; i++)

{

Circle circle = new Circle();

circle.Generate(step\_point, 30);

Circles.Add(circle);

step\_point += step;

}

}

public float Radius { get; private set; }

/// <summary>

/// Perform draw of <see cref="Circle2DArray"/>

/// </summary>

/// <param name="context"></param>

/// <param name="location"></param>

public void Draw(OpenGL context, Point3DFloat location)

{

foreach (var circle in this.Circles)

circle.Draw(context, location);

}

}

}

using System;

using System.ComponentModel;

using SharpGL;

using Maneuver.Scene.Components;

namespace Maneuver.Scene.Primitives.ExtendedPrimitives

{

public class Rocket : IRenderablePrimitive, INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

public void OnPropetyChanged(PropertyChangedEventArgs e) => PropertyChanged?.Invoke(this, e);

private float[] vertices;

private float[] normals;

private float[] texcoords;

private uint[] indices;

private float[] colors;

private float scale\_value;

private float course, pitch, azimuth;

public float Scale

{

get => scale\_value;

set

{

if (scale\_value < 0.1f)

throw new ArgumentException("Scale value can not be latter 0.1");

scale\_value = value;

OnPropetyChanged(new PropertyChangedEventArgs("Scale"));

}

}

/// <summary>

/// Рыскание

/// </summary>

public float Course

{

get => course;

set

{

course = value;

OnPropetyChanged(new PropertyChangedEventArgs("Yaw"));

}

}

/// <summary>

/// Тангаж

/// </summary>

public float Pitch

{

get => pitch;

set

{

pitch = value;

OnPropetyChanged(new PropertyChangedEventArgs("Pitch"));

}

}

/// <summary>

/// Крен

/// </summary>

public float Azimuth

{

get => azimuth;

set

{

azimuth = value;

OnPropetyChanged(new PropertyChangedEventArgs("Roll"));

}

}

public Rocket() { }

public Rocket(float[] vertices, float[] normals, float[] texcoords, uint[] indices, float[] rgba = null)

{

rgba = rgba ?? new float[4] { 0.4f, 1f, 0.4f, 0.2f };

this.vertices = vertices;

this.normals = normals;

this.texcoords = texcoords;

this.indices = indices;

this.scale\_value = 10;

course = azimuth = pitch = 0;

colors = new float[vertices.Length \* 4];

Helpers.PerformHelper.FillArrayByArray(ref colors, rgba);

}

public void Draw(OpenGL context, Point3DFloat location)

{

var gl = context;

gl.PolygonMode(OpenGL.GL\_FRONT\_AND\_BACK, OpenGL.GL\_FILL);

gl.PushMatrix();

// gl.Rotate(pitch \* 180, course \* 180, azimuth \* 180);

gl.Translate(location.X, location.Y, location.Z);

gl.Scale(this.scale\_value, this.scale\_value, this.scale\_value);

gl.EnableClientState(OpenGL.GL\_VERTEX\_ARRAY);

gl.EnableClientState(OpenGL.GL\_NORMAL\_ARRAY);

gl.EnableClientState(OpenGL.GL\_TEXTURE\_COORD\_ARRAY);

gl.EnableClientState(OpenGL.GL\_COLOR\_ARRAY);

unsafe

{

fixed (float\* p = colors)

{

IntPtr ptr = (IntPtr)p;

gl.ColorPointer(4, OpenGL.GL\_FLOAT, 0, ptr);

}

}

gl.VertexPointer(3, 0, vertices);

gl.NormalPointer(OpenGL.GL\_FLOAT, 0, normals);

gl.TexCoordPointer(2, OpenGL.GL\_FLOAT, 0, vertices);

gl.DrawElements(OpenGL.GL\_POLYGON, indices.Length, indices);

// gl.DrawArrays(OpenGL.GL\_QUADS, 0, vertices.Length / 3);

gl.DisableClientState(OpenGL.GL\_VERTEX\_ARRAY);

gl.DisableClientState(OpenGL.GL\_NORMAL\_ARRAY);

gl.DisableClientState(OpenGL.GL\_TEXTURE\_COORD\_ARRAY);

gl.DisableClientState(OpenGL.GL\_COLOR\_ARRAY);

gl.PopMatrix();

}

}

}

using System;

using System.Collections.Generic;

using Maneuver.Helpers;

using Maneuver.Scene.Components;

using Maneuver.Scene.Components.Elements;

using Maneuver.Scene.Primitives.ExtendedPrimitives;

using SharpGL;

namespace Maneuver.Scene

{

public class ManeuverScene

{

public OpenGL OpenGLContext { get; set; } = default;

/// <summary>

/// Camera of <see cref="ManeuverScene"/>

/// </summary>

public Camera Camera { get; set; } = new Camera(new Point3DFloat(100, 100, 100), new Point3DFloat(0, 0, 0));

/// <summary>

/// <see cref="ManeuverScene"/> elements

/// </summary>

public List<SceneElement> SceneElements { get; set; } = new List<SceneElement>();

public Kompas Kompas { get; set; } = new Kompas(new Circle2DArray(100, 10), new Point3DFloat(0, 0, 0));

/// <summary>

/// Init <see cref="ManeuverScene"/> with <see cref="OpenGL"/> context

/// </summary>

/// <param name="openGLContext">It is <see cref="OpenGL"/> context</param>

/// <exception cref="ArgumentNullException"></exception>

public ManeuverScene(OpenGL openGLContext)

=> this.OpenGLContext = openGLContext != null ? openGLContext : throw new ArgumentNullException("OpenGL context is null");

/// <summary>

/// Perform the render of <see cref="ManeuverScene"/>

/// </summary>

public void Render()

{

var gl = OpenGLContext;

// производится очистка экрана (очистка буфферов глубины и цвета)

gl.Clear(OpenGL.GL\_COLOR\_BUFFER\_BIT | OpenGL.GL\_DEPTH\_BUFFER\_BIT);

gl.MatrixMode(SharpGL.Enumerations.MatrixMode.Projection);

gl.LoadIdentity();

// получение размера вьюпорта

int[] viewport\_size = new int[4];

gl.GetInteger(OpenGL.GL\_VIEWPORT, viewport\_size);

// настройка отображения матрицы (объектов на экране) с заданием угла обзора, дальностью отрисовки

// (значения буфера глубины)

float x = (float)viewport\_size[2] / viewport\_size[3];

gl.Perspective(45, x, 0.1, double.MaxValue);

// задание точки отображения (наблюдатель, цель)

Camera.LookAt(gl);

// отрисовка всех объектов в сцене

foreach (SceneElement element in SceneElements)

element.Render(gl);

this.Kompas?.Render(gl);

gl.Flush();

}

/// <summary>

/// Perform color-selection

/// </summary>

/// <param name="mouse\_x">mouse X position</param>

/// <param name="mouse\_y">mouse Y position</param>

/// <param name="viewport">Original <see cref="OpenGLControl"/></param>

/// <returns>Selected <see cref="SceneElement"/></returns>

public SceneElement DoHitTest(int mouse\_x, int mouse\_y, OpenGLControl viewport)

{

var colorsElements = this.CreateSelectColors();

var clickedColor = ColorPickerHelper.PerformSelection(viewport, colorsElements, this.Camera, mouse\_x, mouse\_y);

return colorsElements.ContainsKey(clickedColor) ? colorsElements[clickedColor] : null;

}

/// <summary>

/// Create ditionary of scene elements and colors

/// </summary>

/// <returns></returns>

private Dictionary<System.Windows.Media.Color, SceneElement> CreateSelectColors()

{

var pairs = new Dictionary<System.Windows.Media.Color, SceneElement>();

byte r = 0, g = 0, b = 1;

const byte alpha = 1;

foreach (var element in this.SceneElements)

{

if (++b == 0)

if (++g == 0)

if (++r == 0)

// происходит только в случае 16.5 млн объектов-сфер в сцене

throw new Exception("you have reached the limit of colors");

pairs.Add(new System.Windows.Media.Color() { R = r, G = g, B = b, A = alpha }, element);

}

return pairs;

}

public ManeuverScene Clone()

{

ManeuverScene scene = new ManeuverScene(this.OpenGLContext);

scene.Camera = this.Camera.Clone();

scene.SceneElements = new List<SceneElement>(this.SceneElements.ToArray());

return scene;

}

}

}

using System;

using System.Collections.Generic;

using SharpGL;

using Maneuver.Helpers;

using Maneuver.Scene.Components;

using Maneuver.Scene.Primitives;

using Maneuver.Scene.Primitives.ExtendedPrimitives;

namespace Maneuver.Scene

{

public class ManeuverSceneWithFly

{

private OpenGL OpenGLContext = default;

/// <summary>

/// Camera of <see cref="ManeuverScene"/>

/// </summary>

public Camera Camera { get; set; } = new Camera(new Point3DFloat(100, 100, 100), new Point3DFloat(0, 0, 0));

/// <summary>

/// <see cref="ManeuverScene"/> elements

/// </summary>

public List<SceneElement> SceneElements { get; set; } = new List<SceneElement>();

/// <summary>

/// Is rocket in scene. Default - <see cref="SolidSphere"/>

/// </summary>

public Rocket Rocket { get; set; } = RocketLoader.LoadRocketFromFile("flyObject.obj");

// public Rocket Rocket { get; set; } = RocketLoader.LoadRocketFromFile("ballistic\_rocket.obj");

public Point3DFloat RocketLocation { get; set; } = new Point3DFloat(0, 0, 0);

/// <summary>

/// Init <see cref="ManeuverScene"/> with <see cref="OpenGL"/> context

/// </summary>

/// <param name="openGLContext">It is <see cref="OpenGL"/> context</param>

/// <exception cref="ArgumentNullException"></exception>

public ManeuverSceneWithFly(OpenGL openGLContext)

=> this.OpenGLContext = openGLContext != null ? openGLContext : throw new ArgumentNullException("OpenGL context is null");

/// <summary>

/// Perform the render of <see cref="ManeuverScene"/>

/// </summary>

public void Render()

{

var gl = OpenGLContext;

gl.Clear(OpenGL.GL\_COLOR\_BUFFER\_BIT | OpenGL.GL\_DEPTH\_BUFFER\_BIT);

gl.MatrixMode(SharpGL.Enumerations.MatrixMode.Projection);

gl.LoadIdentity();

int[] viewport\_size = new int[4];

gl.GetInteger(OpenGL.GL\_VIEWPORT, viewport\_size);

float x = (float)viewport\_size[2] / viewport\_size[3];

gl.Perspective(45, x, 0.1, double.MaxValue);

Camera.LookAt(gl);

foreach (SceneElement element in SceneElements)

element.Render(gl);

Rocket.Draw(gl, RocketLocation);

gl.Flush();

}

public ManeuverSceneWithFly(List<SceneElement> elements, OpenGL openGLContext)

{

this.SceneElements = new List<SceneElement>(elements);

this.OpenGLContext = openGLContext;

}

}

}

using System;

using System.Windows.Data;

using System.Globalization;

namespace Maneuver.UIExtensions

{

public class CoordinateValueConverter : IMultiValueConverter

{

public object Convert(object[] values, Type targetType, object parameter, System.Globalization.CultureInfo culture)

{

return $"{values[0]} {values[1]} {values[2]}";

}

public object[] ConvertBack(object value, Type[] targetTypes, object parameter, CultureInfo culture)

{

return null;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using System.Windows.Threading;

namespace Maneuver.UIWindows

{

/// <summary>

/// Логика взаимодействия для FlyInfoWindow.xaml

/// </summary>

public partial class FlyInfoWindow : Window

{

public FlyInfoWindow((ManeuverMath.Perform.SphericalCoordinate info, double time)[] flightPoints)

{

InitializeComponent();

List<PointInfo> pointInfos = new List<PointInfo>();

for (int i = 0; i < flightPoints.Length; i++)

{

pointInfos.Add(new PointInfo(flightPoints[i]));

}

PointsDataGrid.ItemsSource = pointInfos;

// PointsDataGrid.Columns[0].CanUserSort = false;

}

public class PointInfo

{

public string Координаты { get; set; }

public string Время { get; set; }

public string Крен { get; set; }

public string Тангаж { get; set; }

public string Рыскание { get; set; }

public PointInfo((ManeuverMath.Perform.SphericalCoordinate info, double time) point)

{

this.Координаты = $"x:{point.info.X}, y:{point.info.Y}, z:{point.info.Z}";

this.Время = $"{point.time} ms";

this.Тангаж = point.info.Pitch.ToString();

this.Крен = point.info.Course.ToString();

this.Рыскание = point.info.Azimuth.ToString();

}

}

#region window ui activity and animation

bool isMenuDragMove = false;

void CloseCommandExecute(object sender, RoutedEventArgs e)

{

this.Close();

}

void MinimizeCommandExecute(object sender, RoutedEventArgs e)

{

WindowState = WindowState.Minimized;

}

void ChangeStateCommandExecute(object sender, RoutedEventArgs e)

{

if (this.WindowState == WindowState.Maximized)

{

this.WindowState = WindowState.Normal;

}

else

{

this.WindowState = WindowState.Maximized;

}

}

private void Menu\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

if (WindowState == WindowState.Maximized)

WindowState = WindowState.Normal;

else WindowState = WindowState.Maximized;

}

private void Menu\_MouseMove(object sender, MouseEventArgs e)

{

if (e.LeftButton == MouseButtonState.Pressed && isMenuDragMove)

{

if (WindowState == WindowState.Maximized)

WindowState = WindowState.Normal;

this.DragMove();

}

}

private void Menu\_MouseDown(object sender, MouseButtonEventArgs e) => isMenuDragMove = true;

private void Menu\_MouseUp(object sender, MouseButtonEventArgs e) => isMenuDragMove = false;

private void MainAppWindow\_StateChanged(object sender, EventArgs e)

=> ChangeWindowStateButton.Content = this.WindowState != WindowState.Maximized ? this.Resources["MaximizeWindowSimbol"] : this.Resources["MinimizeWindowSimbol"];

private void MainAppWindow\_KeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.System)

e.Handled = true;

}

#endregion

}

}

using System;

using System.IO;

using System.Windows;

using System.Threading;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Controls;

using System.Windows.Threading;

using System.Collections.Generic;

using System.Text.RegularExpressions;

using Microsoft.Win32;

using SharpGL;

using Maneuver.Helpers;

using ManeuverMath.Perform;

using Maneuver.Scene.Primitives;

using Maneuver.Scene.Components;

using Maneuver.Scene.Components.Elements;

using Maneuver.Scene.Primitives.ExtendedPrimitives;

using Newtonsoft.Json;

namespace Maneuver.UIWindows

{

/// <summary>

/// Логика взаимодействия для ManeuverView.xaml

/// </summary>

public partial class ManeuverView : Window

{

const int rDeley = 100;

private Scene.ManeuverSceneWithFly scene;

private bool isMenuDragMove = false;

private (SphericalCoordinate info, double time)[] FlightPoints;

private Thread playerThead = null;

public ManeuverView(MathManeuverPoint[] mathManeuverPoints, List<SceneElement> sceneElements )

{

InitializeComponent();

ManeuverSolution \_solution = new ManeuverSolution(mathManeuverPoints);

this.scene = new Scene.ManeuverSceneWithFly(sceneElements, this.OpenGLView.OpenGL);

var flight\_points = new LinkedList<(SphericalCoordinate info, double time)>();

double currentTime = 1; // time in milliseconds

const double milliDivisor = 0.1;

while (true)

{

var fly\_info = \_solution.PerformMotion(currentTime);

if (fly\_info is null) break; // is end

flight\_points.AddLast(

(fly\_info, currentTime)

);

currentTime += milliDivisor;

}

FlightPoints = new (SphericalCoordinate info, double time)[flight\_points.Count];

flight\_points.CopyTo(FlightPoints, 0);

RocketFlySlider.Maximum = flight\_points.Count;

this.scene.Camera.Position.Set(15000, 15000, 15000);

this.scene.Camera.Target.Set(0, 0, 0);

}

public ManeuverView((SphericalCoordinate info, double time)[] plan, Kompas kompas, Camera camera)

{

InitializeComponent();

scene = new Scene.ManeuverSceneWithFly(this.OpenGLView.OpenGL);

scene.Camera = camera;

scene.SceneElements.Add(kompas);

this.FlightPoints = plan;

this.scene.Camera.Position.Set(15000, 15000, 15000);

this.scene.Camera.Target.Set(0, 0, 0);

RocketFlySlider.Maximum = this.FlightPoints.Length;

}

private void OpenGLView\_OpenGLDraw(object sender, RenderEventArgs args)

{

var number = (int) this.RocketFlySlider.Value;

if (number > FlightPoints.Length - 1)

number = FlightPoints.Length - 1;

var fly\_info = FlightPoints[number];

//this.scene.RocketLocation.Set((float)fly\_info.info.Y, (float)fly\_info.info.Z, (float)fly\_info.info.X);

//this.scene.RocketLocation.Set((float)fly\_info.info.Z, (float)fly\_info.info.Y, (float)fly\_info.info.X);

this.scene.RocketLocation.Set((float)fly\_info.info.Z, (float)fly\_info.info.X, (float)fly\_info.info.X);

this.scene.Rocket.Pitch = (float)fly\_info.info.Pitch;

this.scene.Rocket.Azimuth = (float)fly\_info.info.Azimuth;

this.scene.Rocket.Course = (float)fly\_info.info.Course;

this.scene.Render();

}

#region camera

private void RocketFlySlider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

OpenGLView.DoRender();

}

private void OpenGLView\_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)

{

switch (e.Button)

{

case System.Windows.Forms.MouseButtons.Left:

case System.Windows.Forms.MouseButtons.Right:

case System.Windows.Forms.MouseButtons.Middle:

ProgramStateHelper.IsMouseKeyPressed = true;

ProgramStateHelper.LastPressedKey = e.Button;

ProgramStateHelper.LastMouseCursorPressedPosition = new Point(e.X, e.Y);

break;

}

}

private void OpenGLView\_MouseUp(object sender, System.Windows.Forms.MouseEventArgs e)

{

// не нужно реагировать на случай, когда какой-то дурак решил перетаскивание с другого UI-элемента сделать

if (ProgramStateHelper.LastPressedKey != e.Button) return;

switch (e.Button)

{

case System.Windows.Forms.MouseButtons.Left:

// если хотя бы одна из ранее зафиксированных коордитнат мыши не совпадает с текущим, то и не надо производить выделение

if (ProgramStateHelper.LastMouseCursorPressedPosition.X != e.X || ProgramStateHelper.LastMouseCursorPressedPosition.Y != e.Y)

return;

ProgramStateHelper.IsMouseKeyPressed = false;

break;

case System.Windows.Forms.MouseButtons.Middle:

ProgramStateHelper.LastPressedKey = System.Windows.Forms.MouseButtons.None;

ProgramStateHelper.IsMouseKeyPressed = false;

break;

}

}

private void OpenGLView\_MouseMove(object sender, System.Windows.Forms.MouseEventArgs e)

{

// если клавиша мышы не задействована, то и нечего нам тут делать

if (!ProgramStateHelper.IsMouseKeyPressed) return;

// объяснение

// если нажаты alt, ctrl и колесико мышки, то выполняется вращение самого target

// также, если нажаты только если нажаты alt и колесико мышки, то выполняется вокруг текущего target

if (Keyboard.IsKeyDown(Key.LeftAlt) && Keyboard.IsKeyDown(Key.LeftCtrl) && ProgramStateHelper.LastPressedKey == System.Windows.Forms.MouseButtons.Middle)

{

// повторение: этот блок выполняет вращение самого target

}

else if (Keyboard.IsKeyDown(Key.LeftAlt) && ProgramStateHelper.LastPressedKey == System.Windows.Forms.MouseButtons.Middle)

{

// повторение: этот блок выполняет вращение вокруг target

// получаем предыдущие координаты мыши

double last\_mouse\_x = ProgramStateHelper.LastMouseCursorPressedPosition.X;

double last\_mouse\_y = ProgramStateHelper.LastMouseCursorPressedPosition.Y;

// получаем текущее положение мыши

var current\_mouse\_x = e.X;

var current\_mouse\_y = e.Y;

/\* вычисления + изменение камеры \*/

// инициализация относитьельного вектора (camera.info -> cameta.target) и вычисление длинны относительного вектора

Point3DFloat vertex = new Point3DFloat()

{

X = this.scene.Camera.Position.X - this.scene.Camera.Target.X,

Y = this.scene.Camera.Position.Y - this.scene.Camera.Target.Y,

Z = this.scene.Camera.Position.Z - this.scene.Camera.Target.Z

};

vertex.X += this.scene.Camera.Target.X;

vertex.Y += this.scene.Camera.Target.Y;

vertex.Z += this.scene.Camera.Target.Z;

double radius = Math.Sqrt(vertex.X \* vertex.X + vertex.Y \* vertex.Y + vertex.Z \* vertex.Z);

// получение текущих theta и phi (подробнее: https://en.wikipedia.org/wiki/Spherical\_coordinate\_system)

double theta = Math.Acos(this.scene.Camera.Position.Z / radius); // угол наклона по оси z (упрощенное объяснение)

double phi = Math.Atan2(this.scene.Camera.Position.Y, this.scene.Camera.Position.X); // угол наклона по оси xy (упрощенное объяснение)

// изменение theta и phi

theta -= (float)(current\_mouse\_y - last\_mouse\_y) \* ProgramStateHelper.MouseKoef;

phi -= (float)(current\_mouse\_x - last\_mouse\_x) \* ProgramStateHelper.MouseKoef;

// вычисление новых позиций x, y, z

double newX = SphericalSystemHelper.Calculate3DXPointPosition(radius, theta, phi);

double newY = SphericalSystemHelper.Calculate3DYPointPosition(radius, theta, phi);

double newZ = SphericalSystemHelper.Calculate3DZPointPosition(radius, theta);

this.scene.Camera.Position.Set((float)newX, (float)newY, (float)newZ);

// апдейт текущего положения мыши

ProgramStateHelper.LastMouseCursorPressedPosition = new Point(e.X, e.Y);

OpenGLView.DoRender();

}

else if (ProgramStateHelper.LastPressedKey == System.Windows.Forms.MouseButtons.Middle)

{

// этот блок выполняет перемещение камеры и цели камеры (target-а) перпендикулярно

}

}

private void OpenGLView\_MouseWheel(object sender, System.Windows.Forms.MouseEventArgs e)

{

var delta\_value = e.Delta;

if (delta\_value > 0)

{

float newX = SphericalSystemHelper.CalculateScaledCoordinate(this.scene.Camera.Position.X, this.scene.Camera.Target.X, 0.2f);

float newY = SphericalSystemHelper.CalculateScaledCoordinate(this.scene.Camera.Position.Y, this.scene.Camera.Target.Y, 0.2f);

float newZ = SphericalSystemHelper.CalculateScaledCoordinate(this.scene.Camera.Position.Z, this.scene.Camera.Target.Z, 0.2f);

// если мы в плонтную станем к обекту на который смотрим то не надо становится

if (Math.Truncate(this.scene.Camera.Target.X) == Math.Truncate(newX) &&

Math.Truncate(this.scene.Camera.Target.Y) == Math.Truncate(newY) &&

Math.Truncate(this.scene.Camera.Target.Z) == Math.Truncate(newZ)

) return;

this.scene.Camera.Position.Set(newX, newY, newZ);

}

else

{

float newX = SphericalSystemHelper.CalculateScaledCoordinate(this.scene.Camera.Position.X, this.scene.Camera.Target.X, 2.2f);

float newY = SphericalSystemHelper.CalculateScaledCoordinate(this.scene.Camera.Position.Y, this.scene.Camera.Target.Y, 2.2f);

float newZ = SphericalSystemHelper.CalculateScaledCoordinate(this.scene.Camera.Position.Z, this.scene.Camera.Target.Z, 2.2f);

if (this.scene.Camera.Position.Z > 0)

{

if (newZ < 0)

newZ = -newZ;

}

else

{

if (newZ > 0)

newZ = -newZ;

}

if (this.scene.Camera.Position.X > 0)

{

if (newX < 0)

newX = -newX;

}

else

{

if (newX > 0)

newX = -newX;

}

if (this.scene.Camera.Position.Y > 0)

{

if (newY < 0)

newY = -newY;

}

else

{

if (newY > 0)

newY = -newY;

}

this.scene.Camera.Position.Set(newX, newY, newZ);

}

OpenGLView.DoRender();

}

#endregion

#region window ui activity and animation

private void NumberValidationTextBox(object sender, TextCompositionEventArgs e)

{

Regex regex = new Regex("[^0-9]+");

e.Handled = regex.IsMatch(e.Text);

}

private void RocketScaleTextBox\_TextChanged(object sender, System.Windows.Controls.TextChangedEventArgs e)

{

if (!this.IsLoaded) return;

if ((sender as TextBox).Text == string.Empty)

return;

try

{

int value = (int)(sender as TextBox).Text.ToFloat();

if (value == 0) return;

this.scene.Rocket.Scale = value;

OpenGLView.Refresh();

}

catch { }

}

void CloseCommandExecute(object sender, RoutedEventArgs e)

{

this.Close();

}

void MinimizeCommandExecute(object sender, RoutedEventArgs e)

{

WindowState = WindowState.Minimized;

}

void ChangeStateCommandExecute(object sender, RoutedEventArgs e)

{

if (this.WindowState == WindowState.Maximized)

{

this.WindowState = WindowState.Normal;

}

else

{

this.WindowState = WindowState.Maximized;

}

}

private void Menu\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

if (WindowState == WindowState.Maximized)

WindowState = WindowState.Normal;

else WindowState = WindowState.Maximized;

}

private void Menu\_MouseMove(object sender, MouseEventArgs e)

{

if (e.LeftButton == MouseButtonState.Pressed && isMenuDragMove)

{

if (WindowState == WindowState.Maximized)

WindowState = WindowState.Normal;

this.DragMove();

}

}

private void Menu\_MouseDown(object sender, MouseButtonEventArgs e) => isMenuDragMove = true;

private void Menu\_MouseUp(object sender, MouseButtonEventArgs e) => isMenuDragMove = false;

private void MainAppWindow\_StateChanged(object sender, EventArgs e)

=> ChangeWindowStateButton.Content = this.WindowState != WindowState.Maximized ? this.Resources["MaximizeWindowSimbol"] : this.Resources["MinimizeWindowSimbol"];

private void GridSplitter\_DragDelta(object sender, System.Windows.Controls.Primitives.DragDeltaEventArgs e)

{

this.Dispatcher.BeginInvoke(DispatcherPriority.Send,

(ThreadStart)delegate ()

{

this.OpenGLView.Refresh();

}

);

}

private void MainAppWindow\_KeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.System)

e.Handled = true;

}

#endregion

#region activity

private void SavePlanClick(object sender, RoutedEventArgs e)

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

var dialogResult = saveFileDialog.ShowDialog(this);

if (dialogResult != true)

return;

string filePath = saveFileDialog.FileName;

string ext = Path.GetExtension(filePath);

string fileContent = JsonConvert.SerializeObject(FlightPoints, Newtonsoft.Json.Formatting.Indented, new JsonSerializerSettings() { TypeNameHandling = TypeNameHandling.Auto });

switch (ext)

{

case ".json":

File.WriteAllText(filePath, fileContent);

break;

case ".xml":

var xmlDoc = JsonConvert.DeserializeXmlNode("{\"ManeuverPlanPoint\":" + fileContent + "}", "root");

xmlDoc.Save(filePath);

break;

default:

break;

}

}

private void PlayFlyClick(object sender, RoutedEventArgs e)

{

PlayPlanButton.Background = Brushes.Green;

PausePlanButton.Background = RestartPlanButton.Background = Brushes.Black;

if (playerThead is null)

{

playerThead = new Thread(() =>

{

double rocketStep = 0;

double rocketMax = 0;

Dispatcher.Invoke(() => rocketMax = RocketFlySlider.Maximum);

this.Dispatcher.InvokeAsync(() =>

{

if (RocketFlySlider.Value == rocketMax)

RocketFlySlider.Value = 0;

});

while (rocketStep < rocketMax)

{

Thread.Sleep(rDeley);

this.Dispatcher.InvokeAsync(() =>

{

RocketFlySlider.Value += 1;

});

rocketStep += 1;

}

this.Dispatcher.InvokeAsync(() =>

{

PausePlanButton.Background = PlayPlanButton.Background = RestartPlanButton.Background = Brushes.Black;

});

});

playerThead.Start();

}

else

{

if(playerThead.ThreadState != ThreadState.Running && playerThead.ThreadState != ThreadState.WaitSleepJoin)

playerThead.Resume();

}

}

private void StopFlyClick(object sender, RoutedEventArgs e)

{

PausePlanButton.Background = Brushes.Green;

PlayPlanButton.Background = RestartPlanButton.Background = Brushes.Black;

if (playerThead is null) return;

playerThead.Suspend();

}

private void RestartFlyClick(object sender, RoutedEventArgs e)

{

RestartPlanButton.Background = Brushes.Green;

PausePlanButton.Background = PlayPlanButton.Background = Brushes.Black;

if (playerThead != null)

{

if (playerThead.ThreadState == ThreadState.Running || playerThead.ThreadState == ThreadState.WaitSleepJoin)

if (playerThead.ThreadState != (ThreadState.AbortRequested | ThreadState.Suspended))

playerThead.Abort();

}

playerThead = new Thread(() =>

{

double rocketStep = 0;

double rocketMax = 0;

Dispatcher.Invoke(() => rocketMax = RocketFlySlider.Maximum);

this.Dispatcher.InvokeAsync(() =>

{

RocketFlySlider.Value = 0;

});

while (rocketStep < rocketMax)

{

Thread.Sleep(rDeley);

this.Dispatcher.InvokeAsync(() =>

{

RocketFlySlider.Value += 1;

});

rocketStep += 1;

}

this.Dispatcher.InvokeAsync(() =>

{

PausePlanButton.Background = PlayPlanButton.Background = RestartPlanButton.Background = Brushes.Black;

});

});

playerThead.Start();

}

private void MoreFlyInfoClick(object sender, RoutedEventArgs e)

{

FlyInfoWindow window = new FlyInfoWindow(FlightPoints);

window.ShowDialog();

}

#endregion

private void Window\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

{

if (playerThead != null)

switch (playerThead.ThreadState)

{

case ThreadState.AbortRequested:

break;

case ThreadState.Suspended:

playerThead.Resume();

playerThead.Abort();

break;

case ThreadState.WaitSleepJoin:

playerThead.Abort();

break;

default:

playerThead.Abort();

break;

}

}

}

}

using System;

using System.IO;

using System.Xml;

using System.Windows;

using System.Threading;

using System.Diagnostics;

using System.Windows.Media;

using System.Windows.Input;

using System.Windows.Controls;

using System.Windows.Threading;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Windows.Controls.Primitives;

using Microsoft.Win32;

using Maneuver.Scene;

using Maneuver.Helpers;

using Maneuver.UIWindows;

using ManeuverMath.Perform;

using Maneuver.Scene.Components;

using Maneuver.Scene.Primitives;

using Maneuver.Scene.ManeuverManagement;

using Maneuver.Scene.Components.Elements;

using Maneuver.Scene.Primitives.ExtendedPrimitives;

using Newtonsoft;

using Newtonsoft.Json;

namespace Maneuver

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public delegate Point GetDragDropPosition(IInputElement element);

private bool isMenuDragMove { get; set; } = false;

private bool isSceneOpenFromFile { get; set; } = false; string filePath = "";

ManeuverScene Scene { get; set; } = null;

ObservableCollection<ManeuverPoint> ManeuverPoints { get; set; } = null;

public MainWindow()

{

InitializeComponent();

Scene = new ManeuverScene(OpenGLView.OpenGL);

ManeuverPoints = new ObservableCollection<ManeuverPoint>();

// Scene.SceneElements.Add(new Kompas(new Circle2DArray(100, 10), new Point3DFloat(0, 0, 0)));

ProgramStateHelper.OpenGLControl = OpenGLView;

// show rocket view

// Scene.SceneElements.Add(new Kompas(RocketLoader.LoadRocketFromFile("ballistic\_rocket2.obj"), new Point3DFloat(0, 0, 0)));

SceneObjectsDataGrid.ItemsSource = ManeuverPoints;

}

#region drag drop m-points

private void SceneObjectsDataGrid\_Drop(object sender, DragEventArgs e)

{

if (prevRowIndex < 0)

return;

int index = this.GetDataGridItemCurrentRowIndex(e.GetPosition);

if (index < 0)

return;

if (index == prevRowIndex)

return;

if (index == SceneObjectsDataGrid.Items.Count - 1)

{

MessageBox.Show("This row-index cannot be drop");

return;

}

ObservableCollection<ManeuverPoint> productCollection = ManeuverPoints;

ManeuverPoint changedProduct = productCollection[prevRowIndex];

productCollection.RemoveAt(prevRowIndex);

productCollection.Insert(index, changedProduct);

}

private void SceneObjectsDataGrid\_PreviewMouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

prevRowIndex = GetDataGridItemCurrentRowIndex(e.GetPosition);

if (prevRowIndex < 0)

return;

SceneObjectsDataGrid.SelectedIndex = prevRowIndex;

ManeuverPoint selectedEmp = SceneObjectsDataGrid.Items[prevRowIndex] as ManeuverPoint;

if (selectedEmp == null)

return;

DragDropEffects dragdropeffects = DragDropEffects.Move;

if (DragDrop.DoDragDrop(SceneObjectsDataGrid, selectedEmp, dragdropeffects) != DragDropEffects.None)

{

SceneObjectsDataGrid.SelectedItem = selectedEmp;

}

}

private bool IsTheMouseOnTargetRow(Visual theTarget, GetDragDropPosition position)

{

Rect rect = VisualTreeHelper.GetDescendantBounds(theTarget);

Point point = position((IInputElement)theTarget);

return rect.Contains(point);

}

private DataGridRow GetDataGridRowItem(int index)

{

if (SceneObjectsDataGrid.ItemContainerGenerator.Status != GeneratorStatus.ContainersGenerated)

return null;

return SceneObjectsDataGrid.ItemContainerGenerator.ContainerFromIndex(index) as DataGridRow;

}

private int GetDataGridItemCurrentRowIndex(GetDragDropPosition pos)

{

int curIndex = -1;

for (int i = 0; i < SceneObjectsDataGrid.Items.Count; i++)

{

DataGridRow itm = GetDataGridRowItem(i);

if (IsTheMouseOnTargetRow(itm, pos))

{

curIndex = i;

break;

}

}

return curIndex;

}

int prevRowIndex = -1;

#endregion

#region window ui activity and animation

void CloseCommandExecute(object sender, RoutedEventArgs e)

{

this.Close();

}

void MinimizeCommandExecute(object sender, RoutedEventArgs e)

{

WindowState = WindowState.Minimized;

}

void ChangeStateCommandExecute(object sender, RoutedEventArgs e)

{

if (this.WindowState == WindowState.Maximized)

{

this.WindowState = WindowState.Normal;

}

else

{

this.WindowState = WindowState.Maximized;

}

}

private void Menu\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

if (WindowState == WindowState.Maximized)

WindowState = WindowState.Normal;

else WindowState = WindowState.Maximized;

}

private void Menu\_MouseMove(object sender, MouseEventArgs e)

{

if (e.LeftButton == MouseButtonState.Pressed && isMenuDragMove)

{

if (WindowState == WindowState.Maximized)

WindowState = WindowState.Normal;

this.DragMove();

}

}

private void Menu\_MouseDown(object sender, MouseButtonEventArgs e) => isMenuDragMove = true;

private void Menu\_MouseUp(object sender, MouseButtonEventArgs e) => isMenuDragMove = false;

private void MainAppWindow\_StateChanged(object sender, EventArgs e)

=> ChangeWindowStateButton.Content = this.WindowState != WindowState.Maximized ? this.Resources["MaximizeWindowSimbol"] : this.Resources["MinimizeWindowSimbol"];

private void GridSplitter\_DragDelta(object sender, System.Windows.Controls.Primitives.DragDeltaEventArgs e)

{

this.Dispatcher.BeginInvoke(DispatcherPriority.Send,

(ThreadStart)delegate ()

{

this.OpenGLView.Refresh();

}

);

}

private void MainAppWindow\_KeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.System)

e.Handled = true;

}

#endregion

#region Activity

private void AddNewMoveSphereClick(object sender, RoutedEventArgs e)

{

string x\_text = CreatePointXTextBox.Text.Replace('.', ','); // 12.5 -> 12,5

string y\_text = CreatePointYTextBox.Text.Replace('.', ','); // 12.5 -> 12,5

string z\_text = CreatePointZTextBox.Text.Replace('.', ','); // 12.5 -> 12,5

ManeuverType maneuver = (ManeuverType) PointManeuverTypeComboBox.SelectedItem;

// чуть ниже идут проверки на значения в полях

if(!x\_text.IsFloat())

{

MessageBox.Show("Значение в поле \"X\" не является числом", "Ошибка");

return;

}

if (!y\_text.IsFloat())

{

MessageBox.Show("Значение в поле \"Y\" не является числом", "Ошибка");

return;

}

if (!z\_text.IsFloat())

{

MessageBox.Show("Значение в поле \"Z\" не является числом", "Ошибка");

return;

}

float x = x\_text.ToFloat();

float y = y\_text.ToFloat();

float z = z\_text.ToFloat();

var maneuverPoint = new ManeuverPoint(new SolidSphere(7, 12, 12), new Point3DFloat(x, y, z));

maneuverPoint.Maneuver = maneuver;

Scene.SceneElements.Add(maneuverPoint); // add to scene

ManeuverPoints.Add(maneuverPoint); // add to UI

OpenGLView.Refresh();

}

private void OpenGLView\_DrawScene(object sender, SharpGL.RenderEventArgs args)

{

// draw scene

Scene.Render();

}

private void OpenGLView\_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)

{

switch (e.Button)

{

case System.Windows.Forms.MouseButtons.Left:

case System.Windows.Forms.MouseButtons.Right:

case System.Windows.Forms.MouseButtons.Middle:

ProgramStateHelper.IsMouseKeyPressed = true;

ProgramStateHelper.LastPressedKey = e.Button;

ProgramStateHelper.LastMouseCursorPressedPosition = new Point(e.X, e.Y);

break;

}

}

private void OpenGLView\_MouseUp(object sender, System.Windows.Forms.MouseEventArgs e)

{

// не нужно реагировать на случай, когда какой-то дурак решил перетаскивание с другого UI-элемента сделать

if (ProgramStateHelper.LastPressedKey != e.Button) return;

switch (e.Button)

{

case System.Windows.Forms.MouseButtons.Left:

// если хотя бы одна из ранее зафиксированных коордитнат мыши не совпадает с текущим, то и не надо производить выделение

if (ProgramStateHelper.LastMouseCursorPressedPosition.X != e.X || ProgramStateHelper.LastMouseCursorPressedPosition.Y != e.Y)

return;

// производим выделение объектов (hit test)

var selected = Scene.DoHitTest(e.X, e.Y, this.OpenGLView);

switch(selected)

{

case ManeuverPoint point:

CurrentSelectedObjectPropertyGrid.SelectedObject = point;

Scene.Camera.Target.Set(point.Location.X, point.Location.Y, point.Location.Z);

OpenGLView.Refresh();

break;

default:

CurrentSelectedObjectPropertyGrid.SelectedObject = null;

break;

}

ProgramStateHelper.IsMouseKeyPressed = false;

break;

case System.Windows.Forms.MouseButtons.Middle:

ProgramStateHelper.LastPressedKey = System.Windows.Forms.MouseButtons.None;

ProgramStateHelper.IsMouseKeyPressed = false;

break;

}

}

private void OpenGLView\_MouseMove(object sender, System.Windows.Forms.MouseEventArgs e)

{

// если клавиша мышы не задействована, то и нечего нам тут делать

if (!ProgramStateHelper.IsMouseKeyPressed) return;

// объяснение

// если нажаты alt, ctrl и колесико мышки, то выполняется вращение самого target

// также, если нажаты только если нажаты alt и колесико мышки, то выполняется вокруг текущего target

if (Keyboard.IsKeyDown(Key.LeftAlt) && Keyboard.IsKeyDown(Key.LeftCtrl) && ProgramStateHelper.LastPressedKey == System.Windows.Forms.MouseButtons.Middle)

{

// повторение: этот блок выполняет вращение самого target

}

else if (Keyboard.IsKeyDown(Key.LeftAlt) && ProgramStateHelper.LastPressedKey == System.Windows.Forms.MouseButtons.Middle)

{

// повторение: этот блок выполняет вращение вокруг target

// получаем предыдущие координаты мыши

double last\_mouse\_x = ProgramStateHelper.LastMouseCursorPressedPosition.X;

double last\_mouse\_y = ProgramStateHelper.LastMouseCursorPressedPosition.Y;

// получаем текущее положение мыши

var current\_mouse\_x = e.X;

var current\_mouse\_y = e.Y;

/\* вычисления + изменение камеры \*/

// инициализация относитьельного вектора (camera.position -> cameta.target) и вычисление длинны относительного вектора

Point3DFloat vertex = new Point3DFloat()

{

X = Scene.Camera.Position.X - Scene.Camera.Target.X,

Y = Scene.Camera.Position.Y - Scene.Camera.Target.Y,

Z = Scene.Camera.Position.Z - Scene.Camera.Target.Z

};

vertex.X += Scene.Camera.Target.X;

vertex.Y += Scene.Camera.Target.Y;

vertex.Z += Scene.Camera.Target.Z;

double radius = Math.Sqrt(vertex.X \* vertex.X + vertex.Y \* vertex.Y + vertex.Z \* vertex.Z);

// получение текущих theta и phi (подробнее: https://en.wikipedia.org/wiki/Spherical\_coordinate\_system)

double theta = Math.Acos(Scene.Camera.Position.Z / radius); // угол наклона по оси z (упрощенное объяснение)

double phi = Math.Atan2(Scene.Camera.Position.Y, Scene.Camera.Position.X); // угол наклона по оси xy (упрощенное объяснение)

// изменение theta и phi

theta -= (float)(current\_mouse\_y - last\_mouse\_y) \* ProgramStateHelper.MouseKoef;

phi -= (float)(current\_mouse\_x - last\_mouse\_x) \* ProgramStateHelper.MouseKoef;

// вычисление новых позиций x, y, z

double newX = SphericalSystemHelper.Calculate3DXPointPosition(radius, theta, phi);

double newY = SphericalSystemHelper.Calculate3DYPointPosition(radius, theta, phi);

double newZ = SphericalSystemHelper.Calculate3DZPointPosition(radius, theta);

Scene.Camera.Position.Set((float)newX, (float)newY, (float)newZ);

// апдейт текущего положения мыши

ProgramStateHelper.LastMouseCursorPressedPosition = new Point(e.X, e.Y);

OpenGLView.Refresh();

}

else if (ProgramStateHelper.LastPressedKey == System.Windows.Forms.MouseButtons.Middle)

{

// этот блок выполняет перемещение камеры и цели камеры (target-а) перпендикулярно

}

}

private void OpenGLView\_MouseWheel(object sender, System.Windows.Forms.MouseEventArgs e)

{

var delta\_value = e.Delta;

if (delta\_value > 0)

{

float newX = SphericalSystemHelper.CalculateScaledCoordinate(Scene.Camera.Position.X, Scene.Camera.Target.X, 0.2f);

float newY = SphericalSystemHelper.CalculateScaledCoordinate(Scene.Camera.Position.Y, Scene.Camera.Target.Y, 0.2f);

float newZ = SphericalSystemHelper.CalculateScaledCoordinate(Scene.Camera.Position.Z, Scene.Camera.Target.Z, 0.2f);

// если мы в плонтную станем к обекту на который смотрим то не надо становится

if (Math.Truncate(Scene.Camera.Target.X) == Math.Truncate(newX) &&

Math.Truncate(Scene.Camera.Target.Y) == Math.Truncate(newY) &&

Math.Truncate(Scene.Camera.Target.Z) == Math.Truncate(newZ)

) return;

Scene.Camera.Position.Set(newX, newY, newZ);

OpenGLView.Refresh();

}

else

{

float newX = SphericalSystemHelper.CalculateScaledCoordinate(Scene.Camera.Position.X, Scene.Camera.Target.X, 2.2f);

float newY = SphericalSystemHelper.CalculateScaledCoordinate(Scene.Camera.Position.Y, Scene.Camera.Target.Y, 2.2f);

float newZ = SphericalSystemHelper.CalculateScaledCoordinate(Scene.Camera.Position.Z, Scene.Camera.Target.Z, 2.2f);

if (Scene.Camera.Position.Z > 0)

{

if (newZ < 0)

newZ = -newZ;

}

else

{

if (newZ > 0)

newZ = -newZ;

}

if (Scene.Camera.Position.X > 0)

{

if (newX < 0)

newX = -newX;

}

else

{

if (newX > 0)

newX = -newX;

}

if (Scene.Camera.Position.Y > 0)

{

if (newY < 0)

newY = -newY;

}

else

{

if (newY > 0)

newY = -newY;

}

Scene.Camera.Position.Set(newX, newY, newZ);

OpenGLView.Refresh();

}

}

private void CurrentSelectedObjectPropertyGrid\_PropertyValueChanged(object s, System.Windows.Forms.PropertyValueChangedEventArgs e)

=> OpenGLView.Refresh();

private void PointManeuverTypeComboBox\_SelectionChanged(object sender, System.Windows.Controls.SelectionChangedEventArgs e)

{

if (!(sender as ComboBox).IsVisible)

return;

var point = (sender as ComboBox).DataContext as ManeuverPoint;

point.Maneuver = (ManeuverType) e.AddedItems[0];

}

private void PerformEmulationClick(object sender, RoutedEventArgs e)

{

IEnumerable<ManeuverPoint> obsCollection = ManeuverPoints;

var list = new List<ManeuverPoint>(obsCollection);

if(list.Exists(x => x.Maneuver == ManeuverType.None))

{

MessageBox.Show("необходимо задать типы манёвров для всех точек в пространстве");

return;

}

if(list.Count < 2)

{

MessageBox.Show("Необходимо минимум 2 точки в пространстве");

return;

}

var maneuverPoints = new List<MathManeuverPoint>();

foreach (var item in this.ManeuverPoints)

{

MathManeuverPoint point = new MathManeuverPoint()

{

X = item.Location.X,

Y = item.Location.Y,

Z = item.Location.Z,

ManeuverType = (MathManeuversTypes)item.Maneuver

};

maneuverPoints.Add(point);

}

List<SceneElement> elements = new List<SceneElement>(this.Scene.SceneElements.ToArray());

elements.Add(this.Scene.Kompas);

ManeuverView view = new ManeuverView(maneuverPoints.ToArray(), elements);

view.ShowDialog();

}

private void TestSceneFillClick(object sender, RoutedEventArgs e)

{

this.Scene.SceneElements.Clear();

this.ManeuverPoints.Clear();

ManeuverPoint[] points =

{

new ManeuverPoint(new SolidSphere(7, 12, 12), new Point3DFloat(1000f, 0f, 0f)) {Maneuver = ManeuverType.UniformConstantMotionType},

new ManeuverPoint(new SolidSphere(7, 12, 12), new Point3DFloat(20.0f, 1500.0f, 0f)) {Maneuver = ManeuverType.HorizontalSnakeMotionType},

new ManeuverPoint(new SolidSphere(7, 12, 12), new Point3DFloat(1000f,0f,0f)) {Maneuver = ManeuverType.UniformConstantMotionType},

new ManeuverPoint(new SolidSphere(7, 12, 12), new Point3DFloat(20.0f, -2500.0f, 0f)) {Maneuver = ManeuverType.HorizontalSnakeMotionType},

new ManeuverPoint(new SolidSphere(7, 12, 12), new Point3DFloat(1000f, 0f, 0f)) {Maneuver = ManeuverType.UniformConstantMotionType},

new ManeuverPoint(new SolidSphere(7, 12, 12), new Point3DFloat(20.0f, 1500.0f, 0f)) {Maneuver = ManeuverType.VerticalSnakeMotionType},

new ManeuverPoint(new SolidSphere(7, 12, 12), new Point3DFloat(1000f, 0f, 0f)) {Maneuver = ManeuverType.UniformConstantMotionType},

new ManeuverPoint(new SolidSphere(7, 12, 12), new Point3DFloat(20.0f, -1500.0f, 0f)) {Maneuver = ManeuverType.VerticalSnakeMotionType},

new ManeuverPoint(new SolidSphere(7, 12, 12), new Point3DFloat(1000f, 0f, 0f)) {Maneuver = ManeuverType.UniformConstantMotionType},

};

for (int i = 0; i < points.Length; i++)

{

this.Scene.SceneElements.Add(points[i]);

this.ManeuverPoints.Add(points[i]);

}

this.OpenGLView.Refresh();

}

private void RemoveSelectedManeuverPointClick(object sender, RoutedEventArgs e)

{

var selectedPoint = SceneObjectsDataGrid.SelectedItem as ManeuverPoint;

if (selectedPoint is null)

return;

ManeuverPoints.Remove(selectedPoint);

this.Scene.SceneElements.Remove(selectedPoint);

this.OpenGLView.Refresh();

}

#endregion

#region Save and open scene

// выполняет открытие файла

private void OpenSceneClick(object sender, RoutedEventArgs e)

{

if (isSceneOpenFromFile)

{

var dialogResult = MessageBox.Show("Сохранить изменения в текущей сцене?", "Внимание",

MessageBoxButton.YesNoCancel, MessageBoxImage.Warning);

switch (dialogResult)

{

case MessageBoxResult.Yes:

var pnts = new ManeuverPoint[ManeuverPoints.Count];

ManeuverPoints.CopyTo(pnts, 0);

string fileContent = JsonConvert.SerializeObject(pnts, Newtonsoft.Json.Formatting.Indented, new JsonSerializerSettings() { TypeNameHandling = TypeNameHandling.Auto });

string exton = Path.GetExtension(filePath);

switch (exton)

{

case ".json":

File.WriteAllText(filePath, fileContent);

break;

case ".xml":

var xmlDoc = JsonConvert.DeserializeXmlNode("{\"ManeuverPoint\":" + fileContent + "}", "root");

xmlDoc.Save(filePath);

break;

default:

break;

}

MessageBox.Show($"Старая сцена сохранена по пути {filePath} " +

$"{Environment.NewLine} " +

$"Сейчас откроется выбор другого файла для открытия");

break;

case MessageBoxResult.No:

break;

case MessageBoxResult.Cancel:

return;

}

}

var saveFileDialog = new OpenFileDialog();

var fileDialogResult = saveFileDialog.ShowDialog(this);

if (fileDialogResult != true)

return;

var fileName = saveFileDialog.FileName;

string ext = Path.GetExtension(fileName);

ManeuverPoint[] points = null;

switch (ext)

{

case ".json":

string content = PerformHelper.ReadConentFromFile(fileName);

points = JsonConvert.DeserializeObject<ManeuverPoint[]>(content, new JsonSerializerSettings() { TypeNameHandling = TypeNameHandling.Auto });

break;

case ".xml":

content = PerformHelper.ReadConentFromFile(fileName);

var xmlDoc = new XmlDocument();

xmlDoc.LoadXml(content);

content = JsonConvert.SerializeXmlNode(xmlDoc, Newtonsoft.Json.Formatting.Indented, true);

content = content.Remove(0, content.IndexOf('[')); // костыль

content = content.Remove(content.LastIndexOf(']') + 1); // костыль

points = JsonConvert.DeserializeObject<ManeuverPoint[]>(content, new JsonSerializerSettings() { TypeNameHandling = TypeNameHandling.Auto });

break;

default:

break;

}

ManeuverPoints = new ObservableCollection<ManeuverPoint>();

Scene.SceneElements.RemoveAll(x => x.GetType() == typeof(ManeuverPoint));

for (int i = 0; i < points.Length; i++)

{

if (points[i].ViewObject is SolidSphere sphere)

sphere.Radius = sphere.Radius; // костыль

ManeuverPoints.Add(points[i]);

Scene.SceneElements.Add(points[i]);

}

isSceneOpenFromFile = true;

filePath = fileName;

this.SceneObjectsDataGrid.ItemsSource = ManeuverPoints;

this.OpenGLView.DoRender();

}

// выполняет сохранение сцены в файл

private void SaveSceneClick(object sender, RoutedEventArgs e)

{

var points = new ManeuverPoint[ManeuverPoints.Count];

ManeuverPoints.CopyTo(points, 0);

string fileContent = JsonConvert.SerializeObject(points, Newtonsoft.Json.Formatting.Indented, new JsonSerializerSettings() { TypeNameHandling = TypeNameHandling.Auto });

SaveJsonContentWithFileConditions(fileContent);

this.OpenGLView.DoRender();

}

// выполняет сохранение сцены в файл через диалог с пользователем

private void SaveSceneAsClick(object sender, RoutedEventArgs e)

{

var saveFileDialog = new SaveFileDialog();

var dialogResult = saveFileDialog.ShowDialog(this);

if (dialogResult != true)

return;

var fileName = saveFileDialog.FileName;

string ext = Path.GetExtension(fileName);

var points = new ManeuverPoint[ManeuverPoints.Count];

ManeuverPoints.CopyTo(points, 0);

string fileContent = JsonConvert.SerializeObject(points, Newtonsoft.Json.Formatting.Indented, new JsonSerializerSettings() { TypeNameHandling = TypeNameHandling.Auto });

switch (ext)

{

case ".json":

File.WriteAllText(fileName, fileContent);

break;

case ".xml":

var xmlDoc = JsonConvert.DeserializeXmlNode("{\"ManeuverPoint\":" + fileContent + "}", "root");

xmlDoc.Save(fileName);

break;

default:

break;

}

isSceneOpenFromFile = true;

filePath = fileName;

this.OpenGLView.DoRender();

}

// закрыть без сохранений всё

private void NotSaveSceneAndCloseClick(object sender, RoutedEventArgs e)

{

this.Close();

}

private void SaveJsonContentWithFileConditions(string jsonFileContent)

{

if (isSceneOpenFromFile)

{

string ext = Path.GetExtension(filePath);

switch (ext)

{

case ".json":

File.WriteAllText(filePath, jsonFileContent);

break;

case ".xml":

var xmlDoc = JsonConvert.DeserializeXmlNode("{\"ManeuverPoint\":" + jsonFileContent + "}", "root");

xmlDoc.Save(filePath);

break;

default:

break;

}

}

else

{

var saveFileDialog = new SaveFileDialog();

var dialogResult = saveFileDialog.ShowDialog(this);

if (dialogResult != true)

return;

var fileName = saveFileDialog.FileName;

string ext = Path.GetExtension(fileName);

switch (ext)

{

case ".json":

File.WriteAllText(filePath, jsonFileContent);

break;

case ".xml":

var xmlDoc = JsonConvert.DeserializeXmlNode(jsonFileContent);

xmlDoc.Save(filePath);

break;

default:

break;

}

isSceneOpenFromFile = true;

filePath = fileName;

this.OpenGLView.DoRender();

}

}

private void ImportFlyPlanClick(object sender, RoutedEventArgs e)

{

OpenFileDialog fileDialog = new OpenFileDialog();

var dialogResult = fileDialog.ShowDialog(this);

if (dialogResult != true)

return;

var fileName = fileDialog.FileName;

var ext = Path.GetExtension(fileName);

(SphericalCoordinate info, double time)[] plan = null;

switch (ext)

{

case ".json":

string content = PerformHelper.ReadConentFromFile(fileName);

plan = JsonConvert.DeserializeObject<(SphericalCoordinate info, double time)[]>(content, new JsonSerializerSettings() { TypeNameHandling = TypeNameHandling.Auto });

break;

case ".xml":

content = PerformHelper.ReadConentFromFile(fileName);

var xmlDoc = new XmlDocument();

xmlDoc.LoadXml(content);

content = JsonConvert.SerializeXmlNode(xmlDoc, Newtonsoft.Json.Formatting.Indented, true);

content = content.Remove(0, content.IndexOf('[')); // костыль

content = content.Remove(content.LastIndexOf(']') + 1); // костыль

plan = JsonConvert.DeserializeObject<(SphericalCoordinate position, double time)[]>(content, new JsonSerializerSettings() { TypeNameHandling = TypeNameHandling.Auto });

break;

default:

break;

}

ManeuverView maneuverView = new ManeuverView(plan, this.Scene.Kompas, this.Scene.Camera.Clone());

maneuverView.ShowDialog();

}

#endregion

#region menu activity

private void SetCameraToCenterClick(object sender, RoutedEventArgs e)

{

this.Scene.Camera.SetTarget(0, 0, 0);

this.OpenGLView.DoRender();

}

private void ResetCameraClick(object sender, RoutedEventArgs e)

{

this.Scene.Camera.SetTarget(0, 0, 0);

this.Scene.Camera.SetPosition(100, 100, 100);

this.OpenGLView.DoRender();

}

private void SetSphereRadiusSizeClick(object sender, RoutedEventArgs e)

{

var func = PerformHelper.CreateActionByButtonName((sender as MenuItem).Name);

for (int i = 0; i < this.Scene.SceneElements.Count; i++)

if (this.Scene.SceneElements[i] is ManeuverPoint point)

if (point.ViewObject is SolidSphere sphere)

sphere.Radius = func(sphere.Radius);

this.OpenGLView.DoRender();

}

private void SetKompasRadiusSizeClick(object sender, RoutedEventArgs e)

{

var func = PerformHelper.CreateActionByButtonName((sender as MenuItem).Name);

var radius = this.Scene.Kompas.Radius;

radius = func(radius);

this.Scene.Kompas = new Kompas(new Circle2DArray(radius, 10), new Point3DFloat(0, 0, 0));

this.OpenGLView.DoRender();

}

private void RemoveAllManeuverPointsClick(object sender, RoutedEventArgs e)

{

this.Scene.SceneElements.RemoveAll(x => x.GetType() == typeof(ManeuverPoint));

this.ManeuverPoints.Clear();

this.OpenGLView.DoRender();

}

private void OpenHelpClick(object sender, RoutedEventArgs e)

{

System.Windows.Forms.Help.ShowHelp(null, "help.chm");

}

#endregion

}

}