МИНОБРНАУКИ РОССИИ
ВЛАДИВОСТОКСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И АНАЛИЗА ДАННЫХ
КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СИСТЕМ

ОТЧЁТ
ПО УЧЕБНОЙ ОЗНАКОМИТЕЛЬНОЙ ПРАКТИКЕ

Студент
гр. БИС-24-01 _____ Н.М. Шевцов

Руководитель
ассистент _____ О.О. Соколов

Владивосток 2026

МИНОБРНАУКИ РОССИИ

ВЛАДИВОСТОКСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И АНАЛИЗА ДАННЫХ

КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СИСТЕМ

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на учебную практику по получению первичных профессиональных умений и навыков

Студенту: гр. БИС-24-01 Шевцову Никите Михайловичу

Срок сдачи работы:18.07.2026

Задание 1. Используя произвольные язык программирования и среду разработки создайте программу, которая отображает на экране монитора график кривой «Гипотрохоида».

Задание 2. Используя результаты предыдущего задания создайте анимацию примитива, движущегося по траектории построенной кривой.

Задание 3. Представление собранных материалов руководителю практики, оформление отчета по практике по стандарту вуза.

Структура отчета по практике:

- **Введение;**
- **Описание выбранной фигуры;**
- **Описание языка и среды разработки;**
- **Описание принципов работы программа и ее алгоритмов;**
- **Заключение;**
- **Список использованных источников;**
- **Приложение** (программный код).

Отчет по практике оформляется в соответствии с СК-СТО-ТР-04-1.005-2015 «Требования к оформлению текстовой части выпускных квалификационных работ, курсовых работ (проектов), рефератов, контрольных работ, отчётов по практикам, лабораторным работам».

Руководитель практики _____ Соколов О.О.

Задание получил: _____ Шевцов Н.М.

Содержание

Введение

Учебная ознакомительная практика – важный этап учебного процесса, направленный на закрепление и углубление знаний, полученных в ходе теоретического обучения, на подготовку к изучению последующих дисциплин. В процессе прохождения практики приобретаются практические навыки и компетенции в сфере профессиональной деятельности.

Параметрические кривые, описывающие сложные циклические движения, находят широкое применение в механике, робототехнике, компьютерной графике и системах автоматизированного проектирования. Возможность гибкого управления параметрами такой кривой, как гипотрохоида, и визуализации её изменений в реальном времени позволит не только изучить её геометрические свойства, но и применить полученные знания в инженерной практике.

Для практического изучения и использования гипотрохоиды пользователям часто приходится обращаться к громоздким математическим пакетам (MATLAB, MathCAD) или недостаточно гибким веб-инструментам. В связи с этим актуальна задача создания собственного интерактивного приложения, которое сочетало бы простоту использования, наглядность и достаточную гибкость для исследования гипотрохоиды.

Цель работы – закрепление и углубление знаний и умений в области алгоритмизации и программирования, полученных в ходе аудиторных занятий. Для этого будет разработано интерактивное кроссплатформенное приложение с оконным графическим интерфейсом для построения гипотрохоиды с возможностью изменения её параметров, переключения систем координат и анимации движения примитива по её траектории.

Объект исследования – параметрическая кривая «Гипотрохоида».

Для достижения поставленной цели необходимо решить следующие задачи:

- изучить теоретические основы гипотрохоиды;
- провести анализ ТЗ, разработать блок-схему алгоритма работы программы;
- произвести и обосновать выбор языка программирования и среды разработки;
- разработать программу, реализующую отрисовку гипотрохоиды;
- реализовать анимацию движения примитива (закрашенного квадрата синего цвета со стороной a > 2) по траектории построенной кривой;
- провести тестирование приложения на корректность работы;
- обобщить материалы практики и оформить отчёт с необходимыми документами.

Выполнение работы позволит закрепить навыки алгоритмизации и объектно-ориентированного программирования, освоить методы работы с графическими библиотеками для построения динамических графиков, получить практический опыт создания интерактивных приложений с оконным интерфейсом и анимацией, а также углубить понимание параметрических кривых и способов их практического применения.

1 Анализ технического задания

1.1 Гипотрохоида: основные сведения и история

Гипотрохоида – плоская кривая, образуемая точкой, жёстко связанной с окружностью, которая катится без скольжения по внутренней стороне другой, неподвижной окружности. Название кривой происходит от греческого hypo (под, внутри) и латинского trochus (обруч, колесо), что отражает способ её построения: точка описывает траекторию внутри направляющей окружности [1].

Частным случаем гипотрохоиды является гипоциклоида – кривая, которую описывает точка, зафиксированная на ободе катящейся окружности (то есть при равенстве расстояния от центра катящейся окружности до точки радиусу этой окружности). Если же точка находится снаружи или внутри катящейся окружности, получаются соответственно удлинённая или укороченная гипотрохоида, увидеть примеры построения которых можно на рисунке 1.

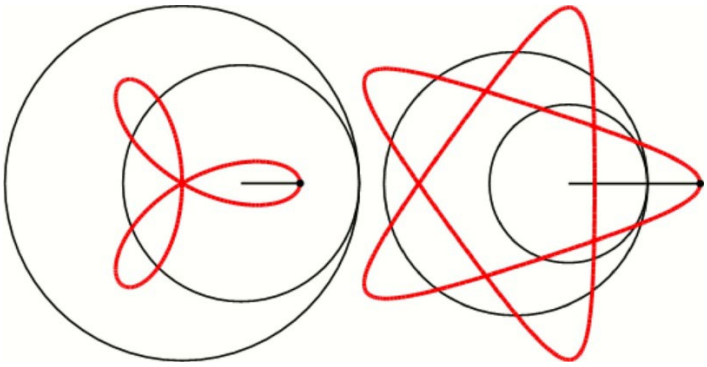

Рисунок 1 – Построение укороченной (слева) и удлиненной (справа) гипотрохоиды

Одним из первых, кто исследовал свойства кривых, получаемых при качении окружности по окружности, был персидский астроном и математик Насир ад-Дин ат-Туси (1201–1274). В своём комментарии к «Альмагесту» Птолемея (1247 год) он описал механизм, позже получивший название «Пара Туси» [2]. Схема данного механизма представлена на рисунке 2.

Рисунок 2 – Схема пары Туси, сделанная ат-Туси в XIII веке

Согласно этому механизму, если окружность радиуса r катится внутри окружности радиуса 2r, то точка на ободе меньшей окружности движется по прямой линии – диаметру большой окружности. Этот результат, известный также как теорема Коперника, использовался для объяснения попятного движения планет и оказал влияние на развитие астрономии.

В XVIII веке было установлено, что придание боковым поверхностям зубьев шестерён и впадин между ними трохоидальной формы снижает момент трения вращающихся колёс и позволяет им вращаться более эффективно [3]. В конце XIX – начале XX века, немецкая фирма Мартина Шиллинга выпускала серию кинематических моделей для демонстрации образования гипотрохоид в учебных целях [4]. Одну из таких моделей можно увидеть на рисунке 3.

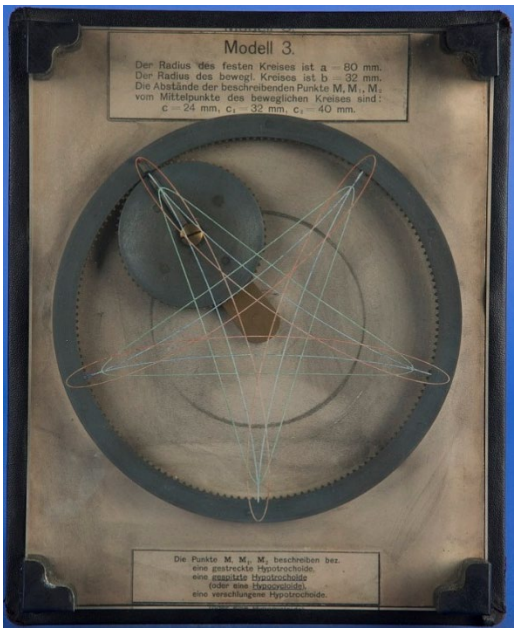

Рисунок 3 – Третья кинематическая модель гипотрохоиды Мартина Шиллинга

В XX веке гипотрохоида получила широкую известность благодаря игрушке Spirograph («Спирограф»), которая позволяет рисовать сложные узоры при помощи зубчатых колёс [5]. Современный спирограф представлен на рисунке 4.

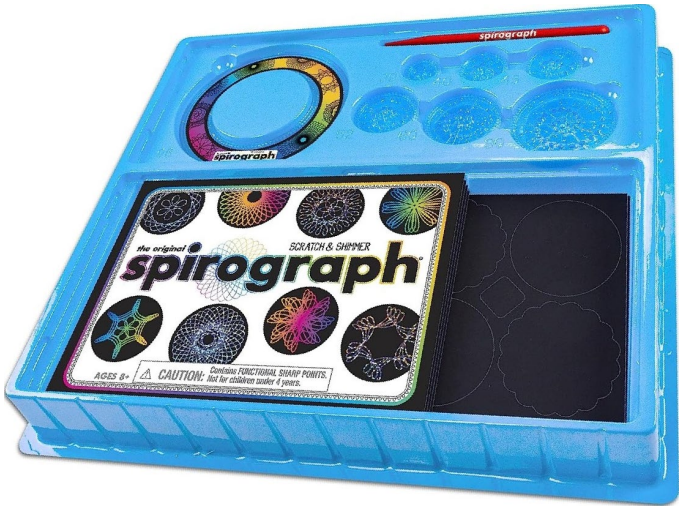

Рисунок 4 – Современный спирограф

Области практического применения гипотрохоиды разнообразны. Наиболее известное инженерное применение – профилирование рабочей камеры роторно-поршневого двигателя Ванкеля. Корпус двигателя имеет внутреннюю поверхность, выполненную по форме эпитрохоиды или гипотрохоиды, а ротор, совершающий планетарное движение, обеспечивает герметичность и изменение объёма камер [6]. Устройство двигателя можно увидеть на рисунке 5.

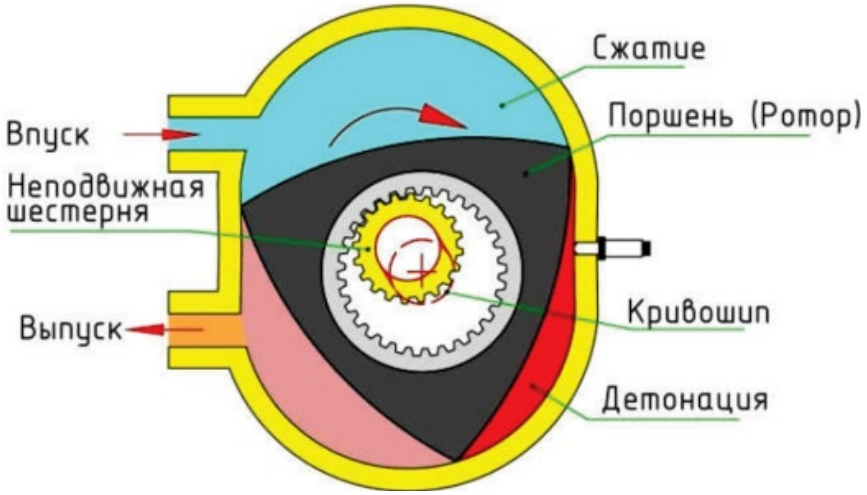

Рисунок 5 – Устройство роторно-поршневого двигателя Ванкеля

Кроме того, трохоидальные профили применяются при проектировании зубчатых передач с внутренним зацеплением, а также в качестве декоративных элементов и для создания орнаментов благодаря их эстетической привлекательности.

1.2 Математический аппарат для построения графика

Для программной реализации построения гипотрохоиды необходимо определить её математическое представление, пригодное для дискретного вычисления координат точек и последующей отрисовки на экране. Гипотрохоида является параметрической кривой, то есть координаты x и y каждой точки выражаются через независимый параметр φ. В декартовой системе координат данная функция задаётся системой параметрических уравнений, представленных в формуле 1.

$$\begin{cases} x(\varphi) = (R - r)\cos\varphi + \mathrm{d}\cos(\dfrac{R - r}{r}\varphi) \\ y(\varphi) = (R - r)\sin\varphi - \mathrm{d}\sin(\dfrac{R - r}{r}\varphi) \end{cases}, \quad (1)$$

где R – радиус направляющей (неподвижной) окружности;

r – радиус катящейся (производящей) окружности;

d – расстояние от центра катящейся окружности до вычерчивающей точки;

φ – параметр (угол поворота производящей окружности), изменяющийся в пределах от 0 до значения, указанного в формуле 2, для получения замкнутой кривой.

Значение, до которого может изменяться параметр φ, представлено ниже в формуле 2.

$$2\pi \frac{R}{\text{НОД}(R, r)}. \quad (2)$$

где R – радиус направляющей окружности;

r – радиус катящейся окружности;

$\text{НОД}(R, r)$ – наибольший общий делитель радиусов двух окружностей.

В полярной системе координат уравнение гипотрохоиды имеет более сложный вид и на практике используется редко. В рамках данной работы построение в полярных координатах будет выполняться путём пересчёта координат из параметрических уравнений в формуле 1 с использованием формул перехода 3 и 4. Вычисление координаты ρ показано в формуле 3.

$$\rho = \sqrt{x^2 + y^2}, \quad (3)$$

где ρ – полярный радиус;

x, y – значения декартовых координат точки кривой.

Вычисление координаты θ показано в формуле 4.

$$\theta = \arctan(\frac{y}{x}), \quad (4)$$

где θ – полярный угол;

x, y – значения декартовых координат точки кривой.

Параметры R, r и d определяют характеристики гипотрохоиды и задают её поведение.

Соотношение R/r определяет количество лепестков (узлов) кривой: после сокращения количество лепестков равно знаменателю полученной дроби (например, при R/r = 5/2 кривая будет иметь 2 лепестка), а если это отношение иррационально, то кривая никогда не замкнётся.

Параметр d определяет положение вычерчивающей точки:

- при d = r точка лежит на ободе катящейся окружности, получается гипоциклоида;
- при d < r точка находится внутри катящейся окружности, получается укороченная гипотрохоида;
- при d > r точка находится за пределами катящейся окружности, получается удлинённая гипотрохоида.

Частный случай, когда R = 2r при d = r, даёт прямую линию, являющуюся диаметром направляющей окружности (пара Туси).

1.3 Блок-схема и логика работы программы

Блок-схема – это графическое представление алгоритма: каждый этап выполнения программы отображается в виде геометрических фигур, соединённых стрелками, которые указывают на порядок выполнения операций [7].

Блок-схемы широко используются при проектировании ПО, так как позволяют наглядно представить логику работы приложения, выявить ошибки на ранних этапах разработки и упростить процесс написания кода.

Для упрощенного представления алгоритма работы будущего приложения была создана блок-схема, которую можно увидеть разделенной на две части на рисунках А.1 и А.2 в Приложении А. При запуске программы происходит инициализация главного окна, содержащего панель управления с кнопками и область для отображения графика. После инициализации программа переходит в режим ожидания действий пользователя.

Пользователю доступны две основные кнопки управления: «Настройки параметров» и «Обновить график». При нажатии на кнопку «Настройки параметров» открывается диалоговое окно, в котором пользователь вводит параметры кривой: радиус направляющей окружности (R), радиус катящейся окружности (r) и расстояние от центра катящейся окружности до вычерчивающей точки (d). Также в этом окне пользователь выбирает систему координат (декартову или полярную) и определяет, необходимо ли включить анимацию движения примитива.

После ввода данных выполняется проверка их корректности: все параметры должны быть положительными числами, при этом радиус катящейся окружности должен быть меньше радиуса направляющей. Если данные некорректны, то пользователю выводится сообщение об ошибке, и окно настроек остаётся открытым для повторного ввода. Если данные корректны, то окно настроек закрывается, а кнопка «Обновить график» становится активной, позволяя пользователю в любой момент перестроить график с текущими параметрами.

При нажатии на кнопку «Обновить график» выполняется очистка области графика в главном окне (с принудительной остановкой таймера анимации, если он был запущен). Затем производится расчёт точек кривой по параметрическим уравнениям гипотрохоиды.

После расчёта пиксельных координат точек для их корректного отображения на экране, программа определяет выбранную пользователем систему координат и отрисовывает соответствующую координатную сетку: для декартовой системы – прямоугольную сетку с осями X и Y, для полярной – концентрические окружности и радиальные лучи. По рассчитанным точкам на области графика строится кривая.

После построения гипотрохоиды программа проверяет, запрашивал ли пользователь анимацию. Если анимация включена, запускается таймер, который с заданным интервалом обновляет положение примитива (закрашенного квадрата) на траектории кривой. Периодическая перерисовка происходит асинхронно по отношению к основному потоку управления. Если анимация отключена, то отображается только статичный график.

В процессе работы приложения программа отслеживает изменение размера главного окна: выполняет перерасчёт масштаба и перерисовывает все элементы графика с учётом новых размеров области отображения. При закрытии главного окна программа завершается.

2 Анализ и выбор средств разработки

2.1 Языки программирования

Для реализации приложения с оконным графическим интерфейсом и возможностью построения параметрических кривых с автоматическим масштабированием требуется язык программирования, обладающий следующими характеристиками: поддержка объектно-ориентированной парадигмы, кроссплатформенность, наличие развитых библиотек для создания графического интерфейса и работы с двухмерной графикой (включающие также возможность реализации с их помощью анимации).

В результате анализа были рассмотрены два языка программирования Python и Java, обладающие перечисленными ранее характеристиками и подходящими для данной работы.

Python – интерпретируемый язык программирования с динамической типизацией, поддерживающий объектно-ориентированную, процедурную и функциональную парадигмы. Отличается простым и читаемым синтаксисом, что ускоряет разработку, а также имеет обширную экосистему библиотек для различных задач, включая создание графических приложений и научные вычисления [8].

Java – компилируемый объектно-ориентированный язык программирования со строгой статической типизацией, работающий на виртуальной машине JVM, что обеспечивает кроссплатформенность. Имеет средства для создания графических интерфейсов и широко применяется в корпоративной разработке [9].

Сравнение рассмотренных языков программирования кратко представлено в таблице 1.

Таблица 1 – Сравнение языков программирования

Характеристика	Python	Java
Кроссплатформенность	Одна программа может быть запущена на Windows, macOS и Linux при наличии необходимых библиотек, для запуска требуется установленный интерпретатор.	Поддерживает Windows, macOS и Linux через виртуальную машину JVM: программа компилируется в байт-код, который исполняется на любой платформе с установленной JRE.
Наличие GUI-библиотек	Широкий выбор библиотек для создания графических интерфейсов как встроенных (Tkinter), так и внешних (PySide6, PyQt, wxPython и др.)	Основные библиотеки для создания графических интерфейсов: Swing (стандартная поставка Java) и JavaFX (начиная с Java 11, требует отдельного подключения).
Масштабирование графики	Библиотеки со встроенными механизмами масштабирования графических сцен (например, QGraphicsView в PySide6 или PyQt) и ручными механизмами управления, в Tkinter масштабирование реализуется вручную через пересчёт координат.	Swing требует ручного пересчёта координат всех элементов и их перерисовки при изменении размеров окна, увеличивая сложность реализации. JavaFX предоставляет более гибкие средства для масштабирования, но требует отдельной настройки.

Продолжение таблицы 1

Характеристика	Python	Java
Скорость разработки	Высокая скорость разработки благодаря простому синтаксису, динамической типизации и интерактивной среде выполнения. Написание прототипов и их отладка занимают меньше времени по сравнению с компилируемыми языками.	Средняя скорость разработки. Статическая типизация требует более детального описания структур данных, а компиляция перед запуском замедляет итерации при отладке. Создание графического интерфейса на Swing требует большего объёма кода.
Поддержка ООП	Поддерживает объектно-ориентированное программирование: классы, наследование, инкапсуляцию и полиморфизм. ООП является нестрогим – допускается смешение с процедурным и функциональным стилями.	Поддерживает строгую объектно-ориентированную парадигму. Всё является объектами, код организуется исключительно в виде классов – это требует более формального подхода к проектированию.

В результате сравнения для реализации приложения был выбран язык Python. На его выбор оказали влияние следующие факторы:

− кроссплатформенность и открытая экосистема библиотек, обеспечивающие свободу выбора инструментов и возможность запуска готового приложения на различных операционных системах без существенных доработок;

− наличие широкого выбора библиотек для создания графического интерфейса;

− высокая скорость разработки благодаря простому синтаксису и динамической типизации, что важно для выполнения работы в ограниченные сроки учебной практики;

− возможность быстро создавать прототипы и тестировать отдельные компоненты программы без необходимости полной компиляции проекта.

Таким образом, Python обеспечивает наиболее эффективное решение поставленной задачи с минимальными затратами времени на разработку и отладку.

2.2 Графические библиотеки

После выбора языка программирования Python появилась необходимость в определении того, какая именно библиотека будет использоваться для создания графического интерфейса и отрисовки графиков. Ключевыми требованиями к библиотеке являются: наличие средств для работы с двухмерной графикой, возможность отображения параметрических кривых, поддержка масштабирования при изменении размеров окна и возможность реализации анимации движения примитива.

В результате анализа были отобраны две библиотеки Tkinter и PySide6, соответствующие перечисленным требованиям. Tkinter – стандартная библиотека для создания графических интерфейсов, входящая в состав поставки Python. Она предоставляет базовые виджеты для построения оконных приложений с холстом для рисования графических примитивов [10].

PySide6 – официальная бесплатная привязка библиотеки Qt для Python, разрабатываемая Qt Group. Предоставляет широкий набор компонентов для создания интерфейсов, включая систему графических сцен QGraphicsView со встроенными механизмами масштабирования и трансформации объектов. Позволяет реализовывать собственные алгоритмы отрисовки [11].

Сравнение рассмотренных библиотек кратко представлено в таблице 2.

Таблица 2 – Сравнение графических библиотек

Характеристика	Tkinter	PySide6 (Qt)
Установка и распространение	Входит в стандартную поставку Python, не требует дополнительной установки. Доступна сразу после установки интерпретатора.	Требует отдельной установки через менеджер пакетов pip. Занимает значительный объём дискового пространства (≈ 600 МБ).
Масштабирование графики	Масштабирование содержимого холста при изменении размеров окна требует ручного пересчёта координат всех элементов и их перерисовки.	Встроенные механизмы масштабирования (QGraphicsView) и возможность реализации собственной логики через переопределение методов отрисовки.
Возможности анимации	Реализация движения объектов требует ручного управления таймерами (метод after()) и перерисовки объектов на холсте для каждого кадра.	Имеет встроенные средства для анимации (классы QPropertyAnimation, QTimer и QGraphicsScene) с поддержкой перемещения объектов по сцене.
Производительность	Подходит для простых графических задач, при большом количестве объектов или частой перерисовке производительность может снижаться.	Использует аппаратное ускорение через OpenGL, обеспечивая высокую производительность при работе с большим количеством объектов.

В результате сравнения для выполнения работы была выбрана библиотека PySide6, на что оказали влияние следующие факторы:

− наличие гибких средств для работы с графикой, включая встроенные механизмы масштабирования (QGraphicsView) и возможность реализации собственной логики отрисовки, что позволяет адаптировать поведение программы под конкретные требования;

− встроенные средства для реализации анимации (QPropertyAnimation, QTimer), упрощающие создание плавного движения примитива по траектории кривой;

− высокая производительность за счёт аппаратного ускорения, а также современный внешний вид интерфейса, настраиваемый через таблицы стилей (QSS).

Использование PySide6 позволяет реализовать все функциональные требования с минимальными трудозатратами, возможностью выбора подходящего способа масштабирования.

2.3 Среды разработки

Для написания программного кода и отладки приложения необходима также среда разработки (IDE), обеспечивающая удобство написания кода, поддержку языка Python, интеграцию с VCS (при необходимости), а также наличие инструментов для отладки и тестирования.

В соответствии с этими требованиями были отобраны и в дальнейшем рассмотрены популярные среды разработки Visual Studio Code и IntelliJ IDEA.

Visual Studio Code – бесплатный кроссплатформенный редактор, разрабатываемый Microsoft. Поддерживает широкий спектр языков программирования через систему расширений. Имеет встроенный терминал, отладчик, интеграцию с Git и библиотеку плагинов [12].

IntelliJ IDEA – IDE от JetBrains, ориентированная на языки Java и Kotlin, но поддерживающая другие языки через плагины. Существует в двух редакциях: бесплатной Community и платной Ultimate. Предоставляет инструменты анализа кода, рефакторинга и отладки [13].

Сравнение рассмотренных сред разработки кратко представлено в таблице 3.

Таблица 3 – Сравнение сред разработки

Характеристика	Visual Studio Code	IntelliJ IDEA
Языки программирования	Поддерживает большое количество языков через систему расширений. Python поддерживается официальным расширением от Microsoft.	Основная направленность – Java и Kotlin. Поддержка Python реализована через плагин, который требует установки и настройки.
Инструменты отладки	Встроенный отладчик с поддержкой точек останова, пошагового выполнения, просмотра переменных и стека вызовов. Интеграция с терминалом.	Мощные инструменты отладки и профилирования, глубокий анализ кода, автодополнение и рефакторинг. Отладчик интегрирован с интерфейсом.
Расширяемость	Огромный каталог расширений для различных языков, инструментов и функций. Позволяет произвести настройку под конкретные задачи.	Поддержка плагинов через JetBrains Marketplace. Основной упор на расширение возможностей для разработки на языке программирования Java.
Производительность	Лёгкий и быстрый, потребляет мало ресурсов, запускается практически мгновенно даже на устройствах с ограниченной производительностью.	Более тяжёлый, требует значительного объёма ОЗУ, запуск может занимать время. Оптимизирован для работы с большими проектами на Java.

В результате сравнения, для дальнейшего создания приложения была выбрана среда разработки Visual Studio Code. На её выбор оказали влияние факторы:

- бесплатность и кроссплатформенность, доступность на различных ОС;
- поддержка Python через официальное расширение от Microsoft, обеспечивающее также автодополнение, отладку, запуск и тестирование кода;
- лёгкость и высокая скорость работы, что важно при разработке учебных проектов, поскольку позволяет сосредоточиться на написании кода, а не на ожидании загрузки среды;
- наличие встроенного терминала, широкие возможности кастомизации через расширения – можно легко и быстро добавить поддержку дополнительных инструментов.

Таким образом, Visual Studio Code предоставляет все необходимые инструменты для разработки приложения на Python с использованием графической библиотеки PySide6 и является отличным выбором для выполнения работы.

3 Разработка программы

3.1 Реализация графического интерфейса и структуры программы

Для создания графического интерфейса в разработанном приложении использовалась библиотека PySide6 – официальная привязка Qt для Python. Qt предоставляет набор готовых компонентов (виджетов): кнопки, поля ввода, выпадающие списки, чек-боксы и другие элементы управления, настраиваемые целиком через написание кода.

Взаимодействие между компонентами Qt осуществляется через механизм сигналов и слотов: при возникновении события (например, нажатии кнопки) генерируется сигнал, который вызывает соответствующий метод-обработчик. Для размещения виджетов в окне используются менеджеры компоновки (layout), автоматически управляющие расположением и размерами элементов при изменении размеров окна.

Разработанное приложение построено с использованием объектно-ориентированной парадигмы и состоит из четырёх основных классов, каждый из которых отвечает за определённую функциональность [14]:

- MainWindow – главное окно приложения;
- SettingsDialog – диалоговое окно для ввода параметров кривой;
- HypoTrochoidWidget – виджет для отрисовки графика;
- CalculationThread – поток для асинхронного расчёта точек кривой.

Взаимодействие классов организовано следующим образом: MainWindow содержит экземпляр HypoTrochoidWidget для отображения графика и управляет открытием SettingsDialog. После ввода параметров в диалоговом окне данные передаются в HypoTrochoidWidget, который, в свою очередь, запускает CalculationThread для вычисления точек кривой в отдельном потоке. Это позволяет сохранить отзывчивость интерфейса во время расчётов.

3.1.1 Класс MainWindow

Класс MainWindow наследуется от QMainWindow и является главным окном приложения. Он содержит панель управления с кнопками «Настройки параметров» и «Обновить график», информационную надпись с отображением текущих параметров программы, а также виджет для отображения графика гипотрохоиды. Кнопка «Обновить график» изначально отключена и активируется только после первого успешного ввода параметров.

Метод show_settings_dialog() открывает диалоговое окно настроек. Если пользователь подтверждает ввод, то введенные параметры сохраняются, активируется кнопка «Обновить график». Информационная надпись при этом обновляется с отображением текущих значений.

Фрагмент кода, отвечающего за настройку главного окна приложения MainWindow, можно увидеть ниже на рисунке 6.

```
class MainWindow(QMainWindow): # главное окно приложения
    def __init__(self): # конструктор…

    def init_ui(self): # метод для создания элементов интерфейса
        central_widget = QWidget() # создание центрального виджета-подложки
        self.setCentralWidget(central_widget) # установка его как центрального для QMainWindow

        main_layout = QVBoxLayout() # создание вертикального макета
        central_widget.setLayout(main_layout) # привязка макета к центральному виджету

        control_layout = QHBoxLayout() # горизонтальный макет для панели управления

        self.settings_button = QPushButton("Настройки параметров") # кнопка для открытия диалогового окна настроек
        self.settings_button.clicked.connect(self.show_settings_dialog) # привязка сигнала к методу открытия окна
        control_layout.addWidget(self.settings_button) # добавление кнопки в макет

        self.redraw_button = QPushButton("Обновить график") # кнопка ручного обновления графика
        self.redraw_button.clicked.connect(self.redraw_graph) # привязка сигнала к методу обновления
        self.redraw_button.setEnabled(False) # отключение кнопки обновления до первого ввода параметров
        control_layout.addWidget(self.redraw_button) # добавление кнопки в макет

        control_layout.addStretch() # добавление растягивающегося пустого пространства для смещения элементов влево

        self.info_label = QLabel("Нажмите 'Настройки параметров' для ввода данных") # информационная надпись
        control_layout.addWidget(self.info_label) # добавление надписи в макет
```

Рисунок 6 – Фрагмент кода из класса MainWindow

Как было упомянуто ранее, класс MainWindow управляет открытием окна SettingsDialog, где производится ввод параметров гипотрохоиды.

3.1.2 Класс SettingsDialog

Класс SettingsDialog наследуется от QDialog и представляет собой модальное диалоговое окно, интерфейс которого разделён на две группы: «Параметры кривой» (поля ввода R, r, d) и «Настройки отображения» (выпадающий список систем координат и чек-бокс для включения анимации). В нижней части окна находятся кнопки «Построить график» и «Закрыть».

Статические переменные класса хранят последние введённые пользователем значения, что позволяет при следующем открытии окна отобразить предыдущие параметры.

Метод validate_and_accept() выполняет проверку корректности введённых данных: все параметры должны быть положительными числами и радиус катящейся окружности должен быть меньше радиуса направляющей. При обнаружении ошибки выводится соответствующее сообщение, и окно остаётся открытым.

3.1.3 Класс HypoTrochoidWidget

Класс HypoTrochoidWidget наследуется от QWidget и является основным виджетом для отрисовки графика. Он отвечает за хранение точек кривой, перевод их математических координат в пиксельные с последующим отображением на экране, а также за управление анимацией. Внутри класса хранятся коэффициент масштабирования для перевода координат в пиксели, таймер для анимации и различные флаги состояния программы.

Метод set_parameters() принимает параметры кривой, останавливает таймер анимации (если он был запущен) и запускает асинхронный расчёт точек.

3.1.4 Класс CalculationThread

Класс CalculationThread наследуется от QThread и предназначен для асинхронного расчёта точек кривой в отдельном потоке. Это позволяет не блокировать основной поток интерфейса во время выполнения вычислений. Класс принимает параметры кривой и выбранную систему координат, а по завершении расчёта отправляет сигнал со списком точек, который обрабатывается в основном потоке для отрисовки.

3.2 Реализация алгоритма построения кривой и анимации

3.2.1 Расчёт точек кривой

Для построения гипотрохоиды необходимо вычислить множество точек, удовлетворяющих параметрическим уравнениям, представленных в формуле 1, и отобразить их на экране. Расчёт выполняется в отдельном потоке с помощью класса CalculationThread.

На основе формулы 2, для вычисления количества оборотов, необходимых для замыкания кривой (periods), используется наибольший общий делитель (НОД) радиусов R и r, умноженных на 100 для их перевода в целые числа. Это позволяет корректно обрабатывать случаи, когда радиусы заданы с плавающей точкой. Кривая замыкается, когда параметр φ достигает значения 2π * periods.

При использовании полярной системы координаты пересчитываются по формулам 3 и 4. Общее количество точек определяется как произведение числа оборотов на количество точек на один оборот (300). Это обеспечивает достаточную гладкость кривой при приемлемой производительности. Фрагмент кода, отвечающего за расчёт математических координат точек гипотрохоиды, представлен на рисунке 7.

```
# предварительное вычисление коэффициентов вне цикла для оптимизации расчёта
R_minus_r = self.R - self.r # разность радиусов (R - r)
ratio = R_minus_r / self.r # коэффициент (R - r) / r для косинуса и синуса

for i in range(total_steps + 1): # цикл, проходящий по всем точкам (включая последнюю)
    fi = (i / total_steps) * 2 * math.pi * periods # вычисление параметра угла (fi) для текущей точки

    # параметрические уравнения гипотрохоиды в декартовой системе координат
    x = R_minus_r * math.cos(fi) + self.d * math.cos(ratio * fi)
    y = R_minus_r * math.sin(fi) - self.d * math.sin(ratio * fi)

    points.append((x, y)) # добавление рассчитанной точки в список

if self.coord_system == 1: # если выбрана полярная система координат
    polar_points = [] # создается временный список для полярных координат
    for x, y in points: # перебираются все точки и вычисляются:
        rho = math.sqrt(x * x + y * y) # полярный радиус
        theta = math.atan2(y, x) # и полярный угол (в радианах) через atan2 (для учёта знаков перед x и y)
        polar_points.append((rho, theta)) # сохраняются результаты в полярном формате
    points = polar_points # декартовые координаты заменяются на полярные

self.finished.emit(points) # отправка рассчитанного списка точек
```

Рисунок 7 – Фрагмент кода из метода run() класса CalculationThread

Для предотвращения излишней нагрузки на систему, максимальное число оборотов кривой ограничено числом 50.

3.2.2 Масштабирование и отрисовка

После завершения расчёта математических координат точек в отдельном потоке вызывается метод paintEvent() класса HypoTrochoidWidget: он выполняет отрисовку графика. Масштабирование реализовано вручную через пересчёт математических координат в пиксельные с учётом текущих размеров виджета для, способного растягиваться при изменении размеров окна, причем центр гипотрохоиды всегда будет совпадать с текущим центром виджета.

Сначала вычисляются границы графика на основе минимальных и максимальных значений координат точек. Затем определяется коэффициент масштабирования, который показывает, сколько пикселей на экране соответствует одной единице математической координаты.

Для этого вычисляются два значения: отношение ширины (w) виджета к разности между максимальным и минимальным значениями по оси X (то есть к ширине графика в математических единицах) и отношение высоты (h) виджета к разности между максимальным и минимальным значениями по оси Y (к высоте графика).

Из этих двух значений выбирается меньшее, чтобы вся кривая гарантированно поместилась в области отображения. Полученный коэффициент scale уменьшается на 5%, благодаря чему по краям остаются небольшие отступы, и график выглядит аккуратно. После этого каждая точка переводится в пиксельные координаты по формулам 5 и 6. В формуле 5 продемонстрировано то, как происходит перевод в пиксели по оси X.

$$x_{px} = \frac{w}{2} + (x - center_x) * scale, \tag{5}$$

где w – текущая ширина виджета для отрисовки графика (w/2 – середина виджета);

x – значение математической координаты точки кривой;

$center_x$ – математическая координата центра гипотрохоиды, вычисляемая как середина диапазона значений точек кривой по оси X;

$scale$ – коэффициент масштабирования, определяющий, сколько пикселей соответствует одной единице математической координаты.

В формуле 6 продемонстрировано то, как происходит перевод в пиксельные координаты на оси Y: знак минуса здесь обусловлен тем, что в компьютерной графике ось Y направлена вниз, когда в математике она направлена вверх. В остальном формула аналогична формуле 5, только здесь отдельные параметры связаны уже с осью Y.

$$y_{px} = \frac{h}{2} - \left(y - center_y\right) * scale, \tag{6}$$

где h – текущая высота виджета для отрисовки графика (h/2 – середина виджета);

y – значение математической координаты точки кривой;

$center_y$ – математическая координата центра гипотрохоиды по оси Y;

$scale$ – коэффициент масштабирования.

В зависимости от выбранной системы координат вызываются методы draw_axes() для декартовой сетки или draw_polar_grid() для полярной. В обоих случаях размер шрифта для подписей и меток также рассчитывается динамически в зависимости от размеров окна. Это обеспечивает читаемость текста на графике при любом масштабе.

3.2.3 Анимация движения примитива

Анимация движения реализована с использованием таймера QTimer, который с интервалом 30 мс обновляет положение примитива на траектории кривой. При включении анимации прогресс (от 0.0 до 1.0) увеличивается на величину animation_speed, которая рассчитывается как отношение количества точек, проходимых за кадр (4), к общему числу точек. Отвечающий за это фрагмент кода из класса HypoTrochoidWidget можно увидеть на рисунке 8.

```
# отрисовка примитива (квадрата) для анимации
if self.animating and len(self.points_px) > 1: # если анимация включена и точек достаточно
    # использование плавной интерполяции вместо дискретного переключения по индексу
    progress = self.anim_progress * (len(self.points_px) - 1) # вещественный индекс текущей точки криво
    idx_floor = int(math.floor(progress)) # целая часть от текущего индекса (ближайший индекс слева)
    idx_ceil = min(idx_floor + 1, len(self.points_px) - 1) # следующий индекс (ближайший индекс справа
    frac = progress - idx_floor # дробная часть для интерполяции (насколько квадрат смещен от левой точ

    # линейная интерполяция позиции между двумя соседними точками
    x1, y1 = self.points_px[idx_floor] # пиксельные координаты первой точки
    x2, y2 = self.points_px[idx_ceil] # пиксельные координаты второй точки

    # позиция = точка_слева + (точка_справа - точка_слева) * дробная_часть
    pos_x = x1 + (x2 - x1) * frac # интерполированная пиксельная X-координата
    pos_y = y1 + (y2 - y1) * frac # интерполированная пиксельная Y-координата

    self.primitive_pos = (pos_x, pos_y) # сохранение полученной позиции квадрата для возможного использ
```

Рисунок 8 – Фрагмент кода из метода paintEvent() класса HypoTrochoidWidget

Для плавного движения примитива по траектории гипотрохоиды используется линейная интерполяция между её соседними точками в пиксельных координатах, что позволяет избежать резких дискретных скачков. При достижении конца траектории прогресс зацикливается, в результате чего примитив снова начинает перемещение из начала в конец.

Размер примитива (квадрата, закрашенного синим цветом) тоже адаптируется под размеры окна: сторона квадрата вычисляется как max(12, min(w, h) / 30), что гарантирует его видимость при любом масштабе. Минимальный возможный размер стороны равен числу 12.

3.2.4 Обработка изменения размера окна

При изменении размера главного окна виджет графика автоматически перерисовывается, при этом коэффициент масштабирования рассчитывается заново. Это обеспечивается вызовом метода update() при событии изменения размера, что приводит к повторному выполнению paintEvent(). Благодаря тому, что все координаты пересчитываются относительно текущих размеров виджета, масштабирование происходит корректно при любых размерах окна.

4 Тестирование программы

4.1 Работа программы при вводе корректных данных

Для проверки корректности работы приложения были выполнены тестовые запуски с различными значениями параметров кривой. Тестирование проводилось на примерах, которые демонстрируют основные режимы работы программы: построение гипотрохоиды в декартовой и полярной системах координат с проверкой работы анимации.

При запуске программы пользователю предлагается ввести параметры кривой в диалоговом окне настроек, открывающемся при нажатии кнопки «Настройки параметров». Для первого теста были выбраны предлагаемые программой по умолчанию значения: радиус направляющей окружности R = 5.0, радиус катящейся окружности r = 3.0, расстояние от центра до точки d = 1.5, система координат – декартовая. Анимация пока что будет отключена. Увидеть диалоговое окно с указанными параметрами можно на рисунке 9.

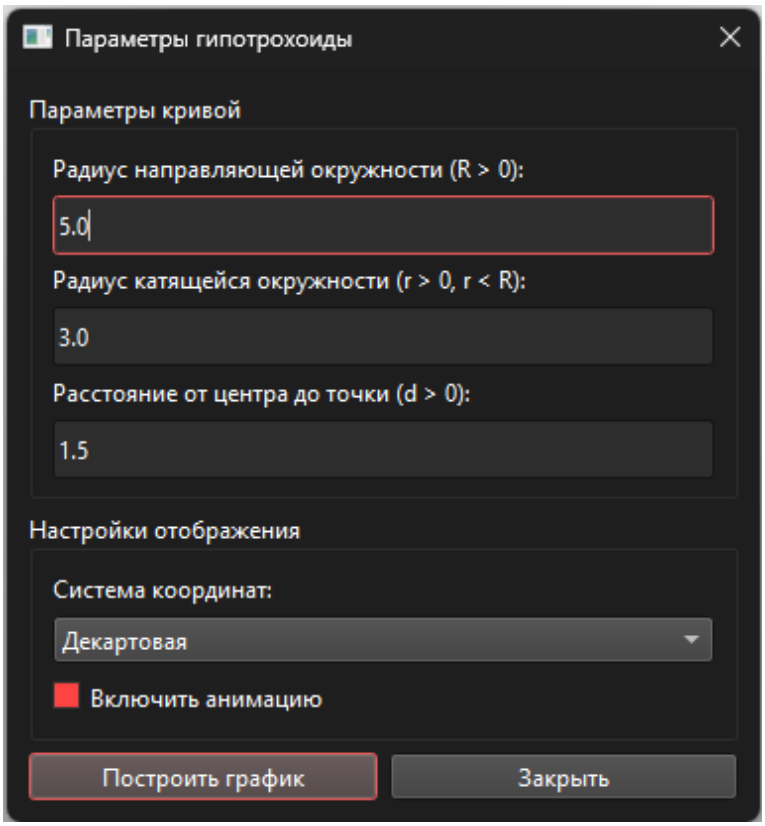

Рисунок 9 – Диалоговое окно SettingsDialog со значениями по умолчанию

После нажатия кнопки «Построить график» программа выполнила расчёт точек и отобразила гипотрохоиду внутри виджета главного окна. Как показано на рисунке 10, кривая отобразилась корректно – форма соответствует ожидаемой для выбранных параметров. Координатная сетка с осями X и Y и числовыми подписями отображается в соответствии с выбранной декартовой системой координат, информационная надпись на панели управления отображает текущие параметры.

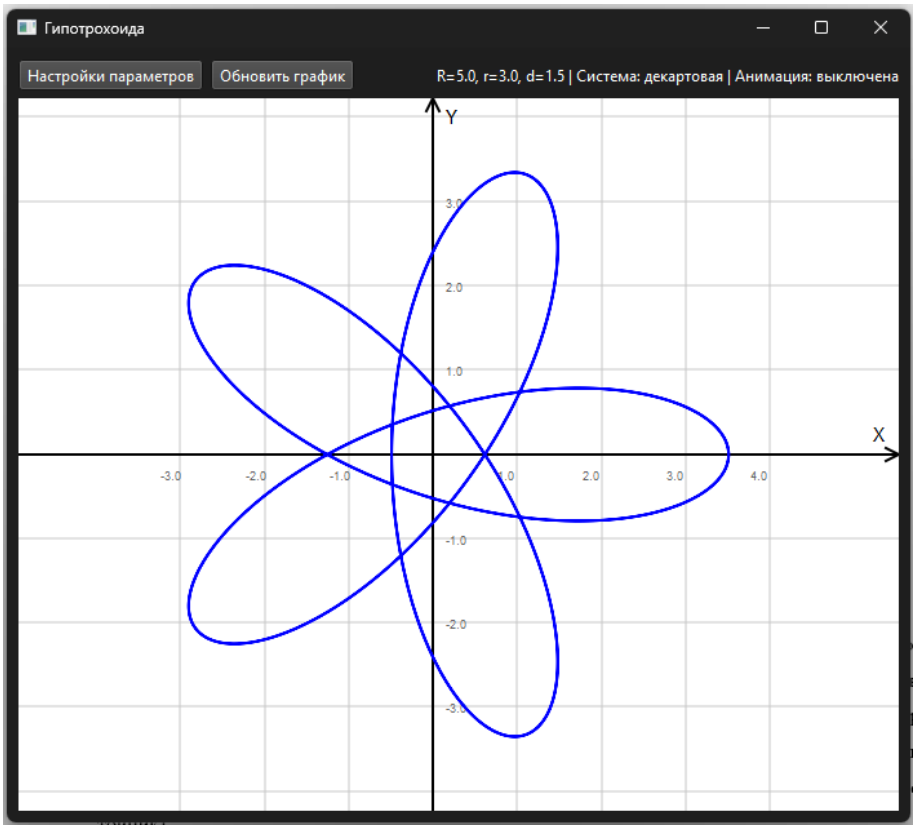

Рисунок 10 – Главное окно с гипотрохоидой и декартовой сеткой

Для проверки работы в полярной системе координат были выбраны те же параметры R = 5.0, r = 3.0, d = 1.5, но система координат была изменена на полярную. Вместе с этим была сразу проведена ещё и проверка работы анимации примитива, так как анимация для одной кривой здесь будет работать одинаково как в декартовой, так и в полярной системе координат.

Теперь после построения в главном окне отображается та же самая кривая, что и раньше, но уже с полярной координатной сеткой на заднем плане, состоящей из концентрических окружностей и радиальных лучей с градусными метками. Центр системы координат обозначен точкой с подписью «О».

После построения графика был также автоматически запущен таймер: закрашенный синий квадрат начал своё движение по траектории гипотрохоиды. Движение квадрата происходит плавно, без рывков, и зацикливается по достижении конца траектории. Размер квадрата адаптировался под окно: примитив виден хорошо. Полученный результат после изменения размеров главного окна представлен на рисунке 11.

Как можно увидеть на рисунке, масштабирование тоже сработало правильно: при изменении размеров окна оно выполнилось автоматически для всех элементов графика, включая координатную сетку, подписи и саму кривую. Элементы перерисовываются с новым коэффициентом масштабирования, сохраняя пропорции и читаемость, а график не вылезает за пределы окна. Была также проведена проверка работы активировавшейся кнопки «Обновить график»: она правильно перестроила кривую с графиком без повторного открытия окна настроек.

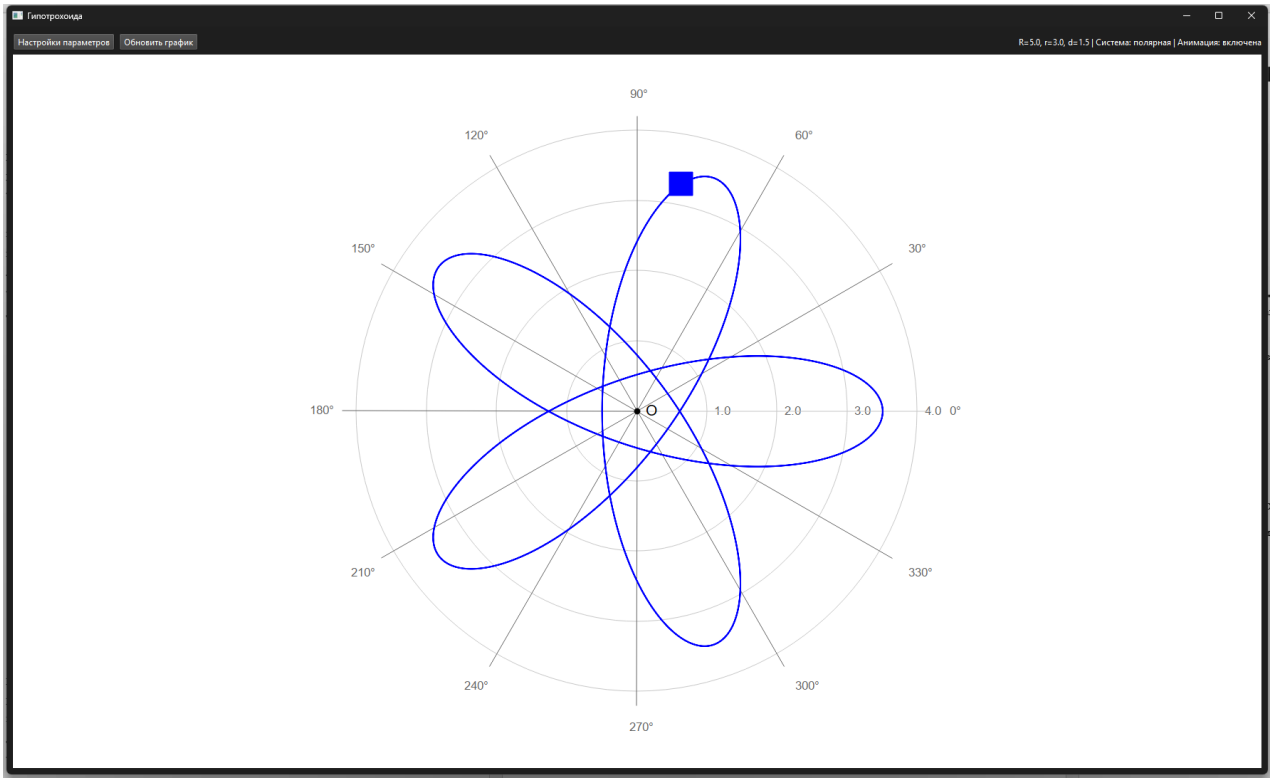

Рисунок 11 – Главное окно измененного размера с полярной сеткой и анимацией

В дополнение к этому тесту, было проведено ещё несколько тестов с различными наборами корректных параметров. Для некоторых наборов была замечена ситуация, когда при достижении конца траектории примитив совершал резкий скачок (телепортировался) в начало кривой. Это не является ошибкой программы, а объясняется особенностями гипотрохоиды.

Как было показано в пункте 1.2, количество лепестков и оборотов кривой определяется соотношением радиусов R и r. При R = 2r кривая вырождается в прямую линию (пара Туси при d = r) или в эллипс (при d > r). В таких случаях траектория замкнута, но из-за дискретного представления кривой в виде набора точек переход с последней точки на первую может визуально быть воспринят как скачок примитива.

Аналогичная ситуация наблюдается при несократимых дробях, например при R = 10.0 и r = 3.0 (соотношение 10/3), где кривая имеет 3 лепестка и замыкается после 10 оборотов. Математически кривая замкнута, но из-за дискретного представления траектории в виде набора точек последняя вычисленная точка может не совпадать с первой. Это связано с тем, что кривая может проходить через общие точки пересечения лепестков, и при переходе с одного участка траектории на другой примитив перескакивает с одной точки на другую.

Например, при параметрах R = 10.0, r = 3.0, d = 1.5 примитив в конце анимации телепортируется из одной точки на оси X в другую, что объясняется особенностями замыкания кривой при данном соотношении параметров. Программа корректно обрабатывает анимацию, зацикливая её и обеспечивая плавное движение на каждом участке траектории, а наблюдаемый скачок является следствием дискретного представления кривой, а не ошибкой.

4.2 Работа программы при вводе некорректных данных

Для проверки устойчивости программы к ошибкам пользователя было выполнено тестирование с вводом некорректных значений параметров. Программа должна корректно обрабатывать такие ситуации, выводя сообщения об ошибках и не допуская сбоев или зависаний.

При вводе в любое из полей параметров буквенного значения (например, «abc» вместо числа) программа должна определить, что введённые данные не являются числом, и вывести соответствующее сообщение «Все параметры должны быть числами!», которое можно увидеть на рисунке 12. Это сообщение появляется при попытке построить график с любыми некорректными данными.

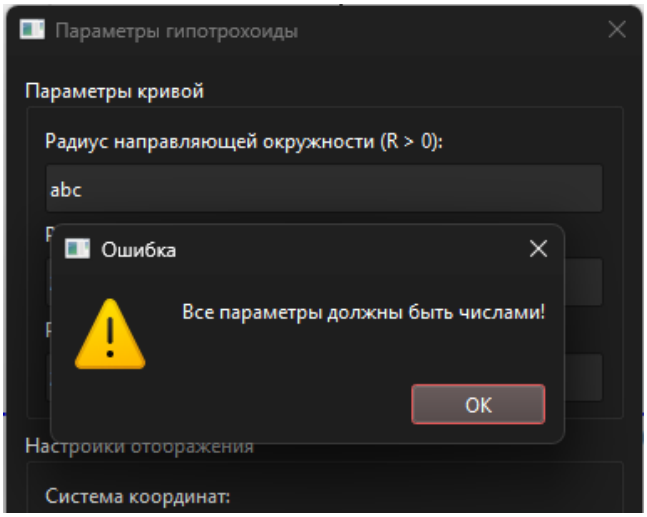

Рисунок 12 – Сообщение об ошибке при вводе нечислового значения параметра

Все параметры кривой (R, r, d) должны быть положительными. При вводе нуля или отрицательного значения программа правильно его определяет и выводит сообщение «Все параметры должны быть положительными числами!», показанное на рисунке 13.

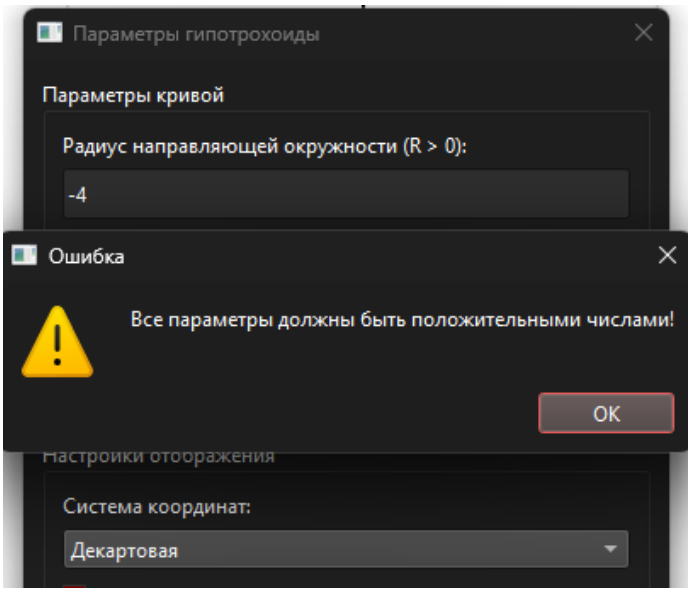

Рисунок 13 – Сообщение об ошибке при вводе неположительного значения параметра

Согласно математическому описанию гипотрохоиды, радиус катящейся окружности r должен быть меньше радиуса направляющей окружности R. При нарушении этого условия (например, R = 3.0, r = 5.0) программа выводит сообщение «Радиус катящейся окружности (r) должен быть меньше радиуса направляющей (R)!», которое можно увидеть на рисунке 14.

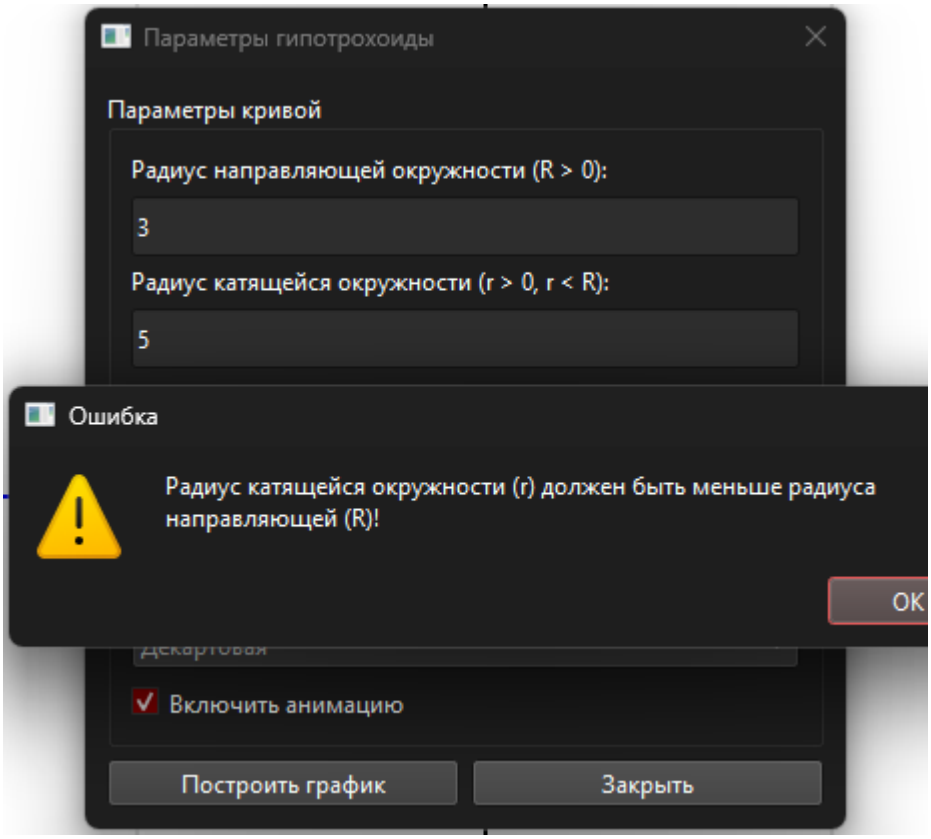

Рисунок 14 – Сообщение об ошибке при вводе параметров, где r > R

Во всех рассмотренных случаях программа не допускает построения графика с некорректными параметрами, выдаёт понятные сообщения об ошибках и остаётся в рабочем состоянии, позволяя пользователю исправить ввод, что подтверждает корректную реализацию механизма валидации входных данных.

Заключение

В ходе прохождения учебной ознакомительной практики было разработано интерактивное кроссплатформенное приложение для построения гипотрохоиды с возможностью изменения параметров кривой, переключения между декартовой и полярной системами координат и анимацией движения примитива по траектории кривой. Для реализации приложения была использована объектно-ориентированная парадигма программирования на языке Python с применением графической библиотеки PySide6.

Это значит, что в результате выполнения работы поставленная цель была достигнута, а также были решены все поставленные задачи:

− изучены теоретические основы гипотрохоиды, включая её историю открытия и области практического применения; геометрический способ её построения, включая параметрические уравнения и влияние параметров R, r, d на форму кривой;

− проведён анализ ТЗ и разработана блок-схема алгоритма работы программы;

− осуществлён и обоснован выбор языка программирования Python с графической библиотекой PySide6 и среды разработки Visual Studio Code в качестве средств разработки;

− разработана программа, реализующая отрисовку гипотрохоиды с автоматическим масштабированием при изменении размеров окна и поддержкой отображения в декартовой и полярной системах координат;

− реализована анимация движения примитива, представляющего собой закрашенный квадрат синего цвета, по траектории построенной кривой;

− проведено тестирование приложения на корректность работы при вводе допустимых и недопустимых значений параметров;

− обобщены материалы практики и оформлен отчёт с необходимыми документами в соответствии с установленными требованиями.

Выполнение работы позволило углубить понимание параметрических кривых и способов их практического применения, закрепить знания в области алгоритмизации и объектно-ориентированного программирования, а также получить практические навыки, которые окажутся полезными в ходе последующего изучения учебных дисциплин.

Список использованных источников

1 Гипотрохоида // Wikipedia: свободная энциклопедия. – URL: https://goo.su/MyUsik (дата обращения: 15.06.2026).

2 Пара Туси // Wikipedia: свободная энциклопедия. – URL: https://goo.su/efUCM (дата обращения: 15.06.2026).

3 Тимофеев Г.А. Теория механизмов и машин: учебник и практикум для вузов / Г.А. Тимофеев. – 4-е изд., перераб. и доп. – Москва: Юрайт, 2026. – 432 с. – URL: https://urait.ru/book/teoriya-mehanizmov-i-mashin-582512 (дата обращения: 16.06.2026).

4 Hypotrochoids, Kinematic Model by Martin Schilling // National Museum of American History. – URL: https://americanhistory.si.edu/collections/object/nmah_1213881 (дата обращения: 17.06.2026).

5 Спирограф (детская игрушка) // Wikipedia: свободная энциклопедия. – URL: https://goo.su/TZy0sTg (дата обращения: 17.06.2026).

6 Смирнова М.В. Теоретические основы теплотехники: учебник для вузов / М.В. Смирнова – 2-е изд. – Москва: Юрайт, 2026. – 237 с. – URL: https://urait.ru/book/teoreticheskie-osnovy-teplotehniki-587867 (дата обращения: 18.06.2026).

7 Блок-схемы: виды, элементы и примеры // Tutortop: интернет-блог. – URL: https://blog.tutortop.ru/blok-shemy-polnoe-rukovodstvo-po-vizualizaczii-proczessov-i-algoritmov/ (дата обращения: 21.06.2026).

8 Черпаков И.В. Алгоритмизация и программирование на Python: учебник для вузов / И.В. Черпаков – Москва: Юрайт, 2026. – 159 с. – URL: https://urait.ru/book/algoritmizaciya-i-programmirovanie-na-python-582412 (дата обращения: 23.06.2026).

9 Лаврищева Е.М. Программная инженерия. Парадигмы, технологии и CASE-средства: учебник для вузов / Е.М. Лаврищева – 2-е изд., испр. – Москва: Юрайт, 2026. – 280 с. – URL: https://goo.su/6XzBUo (дата обращения: 24.06.2026).

10 tkinter // Python documentation. – URL: https://docs.python.org/3/library/tkinter.html (дата обращения: 24.06.2026).

11 Qt for Python // Qt Documentation. – URL: https://doc.qt.io/qtforpython-6/ (дата обращения: 25.06.2026).

12 Что такое Visual Studio Code и для чего он нужен? // Информатек Диджитал. – URL: https://goo.su/GZYvj (дата обращения: 26.06.2026).

13 IntelliJ IDEA: что это за среда разработки // Gitverse: интернет-блог. – URL: https://goo.su/qeNXqA (дата обращения: 26.06.2026).

14 Код программы с гипотрохоидой // Яндекс Диск. – URL: https://disk.yandex.ru/d/i-KzkbzjRUz_HA (дата обращения: 09.07.2026).

Приложение А

Блок-схема, описывающая работу программы

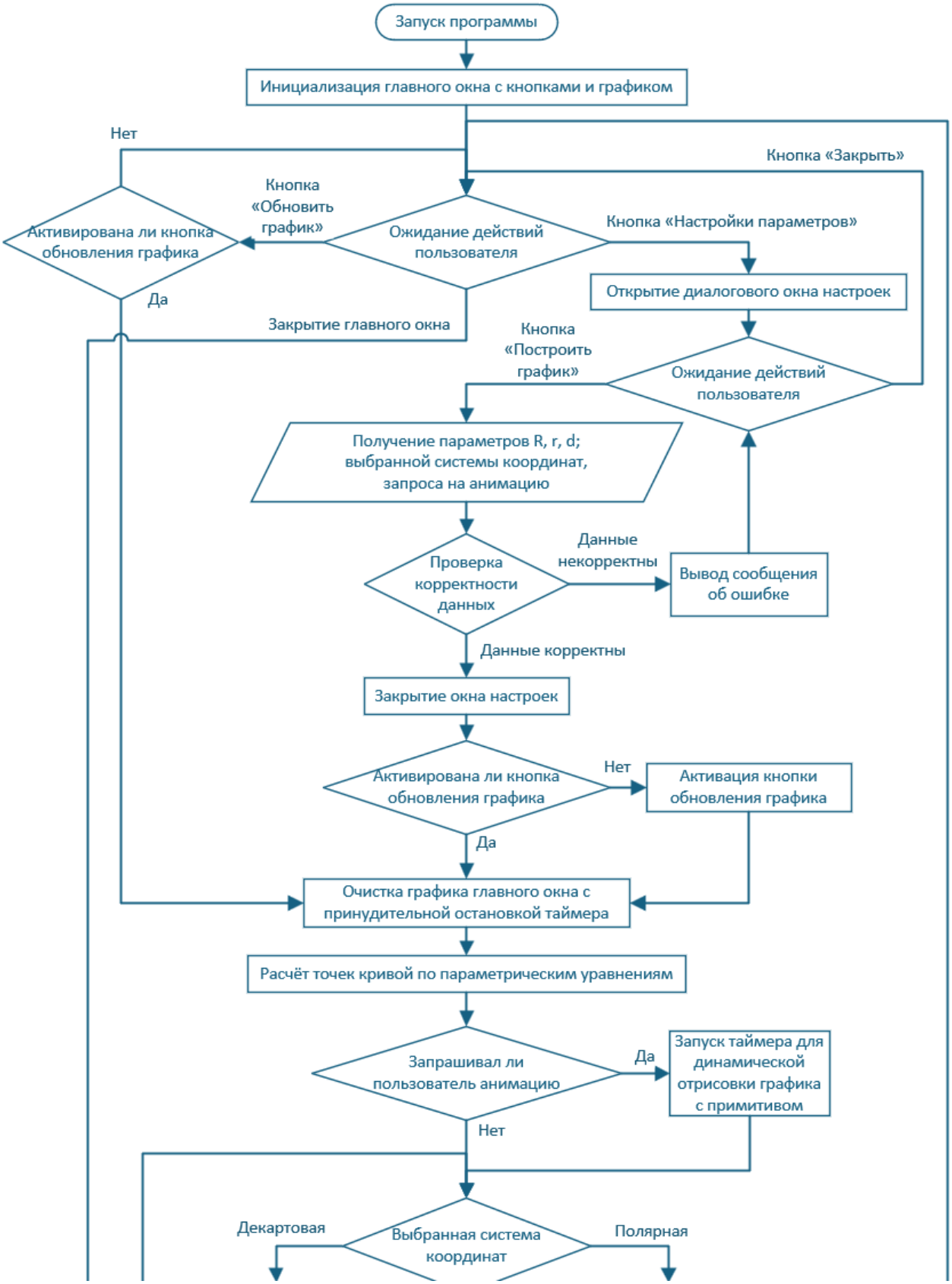

Рисунок А.1 – Блок-схема с алгоритмом работы приложения (часть 1)

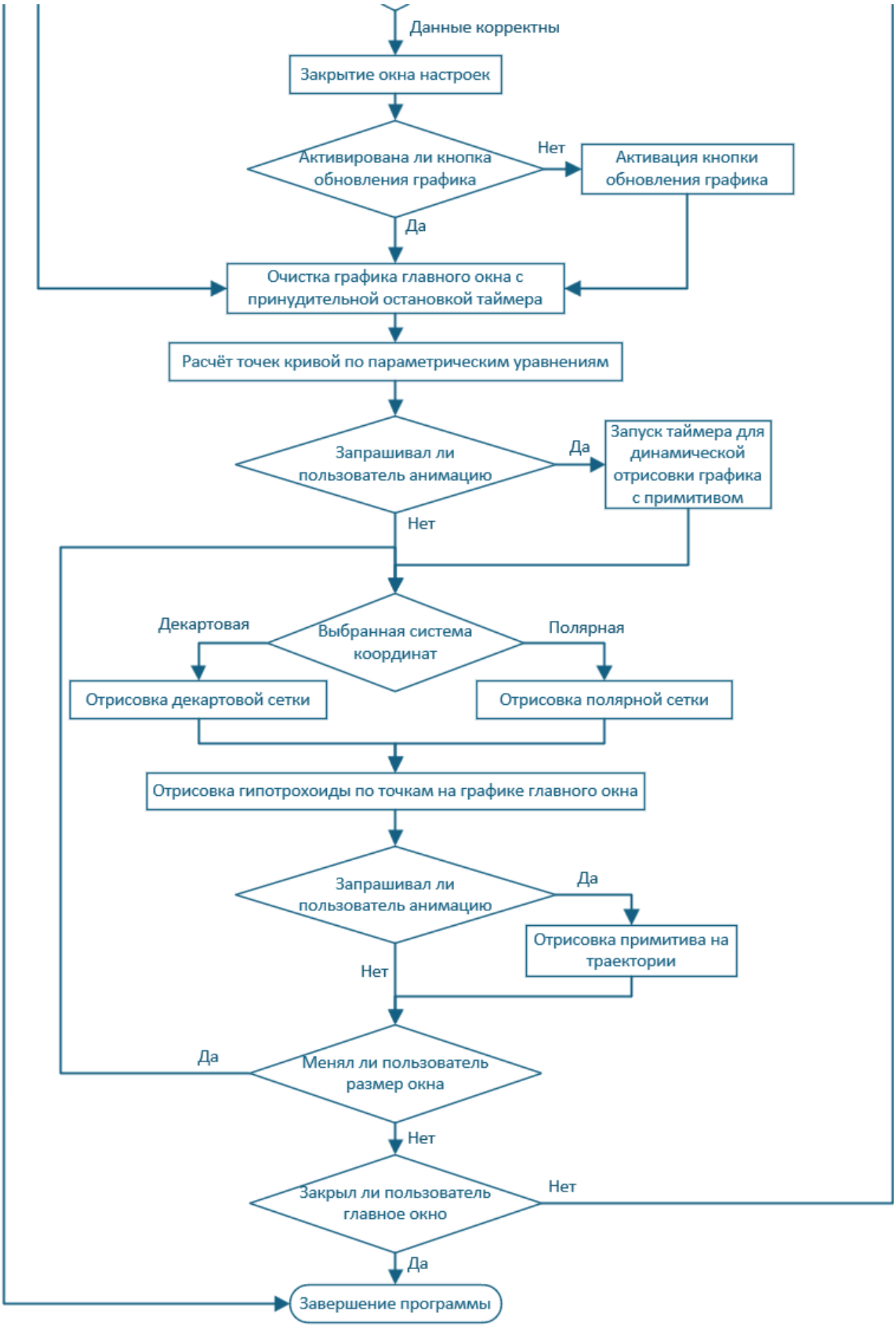

Рисунок А.2 – Блок-схема с алгоритмом работы приложения (часть 2)

КАЛЕНДАРНЫЙ ПЛАН-ГРАФИК (ДНЕВНИК)

прохождения учебной практики по получению первичных профессиональных умений и навыков студента ВВГУ

Студент Шевцов Никита Михайлович направляется на учебную практику по получению первичных профессиональных умений и навыков

с 15 июня 2026 г. по 18 июля 2026 г.

№ п/п	Содержание выполняемых работ по программе	Сроки выполнения		Заключение и оценка руководителя или консультанта	Подпись руководителя или консультанта
		Начало	Окончание		
1	Изучение теоретических основ гипотрохоиды	15.06.2026	18.06.2026	ОТЛ	
2	Разработка логики работы программы, создание блок-схемы	19.06.2026	22.06.2026	ОТЛ	
3	Проведение сравнительного анализа средств разработки	23.06.2026	26.06.2026	ОТЛ	
4	Создание программы	27.06.2026	05.06.2026	ОТЛ	
5	Тестирование программы	06.06.2026	07.07.2026	ОТЛ	
6	Оформление отчёта и необходимых документов	08.04.2026	11.07.2026	ОТЛ	

Согласовано:

Студент-практикант _____ Н.М. Шевцов

подпись

дата

Руководитель от кафедры _____ О.О. Соколов

подпись

дата