

Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Казанский (Приволжский) федеральный университет»

ИНСТИТУТ МАТЕМАТИКИ И МЕХАНИКИ ИМ. Н.И.ЛОБАЧЕВСКОГО
КАФЕДРА ВЫСШЕЙ МАТЕМАТИКИ И МАТЕМАТИЧЕСКОГО
МОДЕЛИРОВАНИЯ

Направление: 050202.65. Информатика с дополнительной специальностью

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(Дипломная работа)

«Применение возможностей графической библиотеки OpenGL в среде MS
Visual Studio при визуализации физических задач»

Работа завершена:

“ ____ ” _____ 2015г. _____ (М.М. Атаев)

Работа допущена к защите:

Научный руководитель

к.ф.-м.н. доцент

“ ____ ” _____ 2015г _____ (О.А. Широкова)

Заведующий кафедрой

д.ф.-м.н., профессор

“ ____ ” _____ 2015г. _____ (Ю.Г. Игнатьев)

Казань – 2015

Содержание

Оглавление

Введение.....	3
Глава I. Основы работы с OpenGL и GLUT	5
1.1. Основы OpenGL	5
1.2. Основы работы с GLUT.....	12
1.3. Настройка GLUT на Visual Studio 2013	14
Глава II. Решение физических проблем методами компьютерного математического моделирования. Их визуализация.....	16
2.1. Моделирование движение тела брошенного под углом к горизонту.....	18
2.1.1. Математическая модель движения тела брошенного под углом к горизонту без учета сопротивления воздуха.	18
2.1.2. Математическая модель движения тела брошенного под углом к горизонту с учетом сопротивления воздуха	27
2.2. Математическая модель движения тела с переменной массой – взлета ракеты	40
2.3. Математическая модель движения космических тел.....	49
Заключение	65
Список литературы.....	66

Введение

Разработанные в выпускной квалификационной работе проекты решения физических задач написаны на языке C++ и отлажены в среде MS Visual Studio.

Microsoft Visual Studio 2013 – это интегрированная среда разработки (Integrated Development Environment – IDE) корпорации Майкрософт для создания, документирования, запуска и отладки программ, написанных на различных языках программирования.

Visual Studio – мощный программный инструмент, позволяющий разрабатывать сложные программные комплексы.

В данной работе рассматривается использование возможностей OpenGL, входящего в Visual Studio, и GLUT (OpenGL Utility Toolkit) при визуализации результатов моделирования физических задач.

Отметим, что для работы с GLUT необходимо его настроить в Visual Studio.

OpenGL (Open Graphics Library – открытая графическая библиотека, графический API) – спецификация, определяющая независимый от языка программирования платформ независимый программный интерфейс для написания приложений, использующих двухмерную и трехмерную компьютерную графику.

GLUT (OpenGL Utility Toolkit – графический инструмент пользователя) – кроссплатформенный графический интерфейс пользователя. Используется для управления и создания окнами, мышью и клавиатурой.

В работе в качестве предметной области для компьютерного моделирования выбраны физические задачи. Рассматриваются особенности технологии математического моделирования на компьютере. Рассмотрены вопросы постановки математических задач на базе фундаментальных физических законов и выбора масштаба физических величин.

В связи с вышесказанным **целью квалификационной работы** является разработка решения физических задач на языке C++ с применением возможностей графической библиотеки OpenGL в среде MS Visual Studio.

Для достижения цели необходимо решить следующие **задачи**:

- Изучить учебно – методическую и научную литературу по теме работы.
- Изучить основы работы с графическими библиотеками OpenGL и GLUT (OpenGL Utility Toolkit);
- Изучить математические модели физических задач;
- Разработать алгоритмы решения и их реализацию на языке C++ в Visual Studio;
- Изучить особенности визуализации результатов моделирования физических задач с использованием графических библиотек OpenGL и GLUT.

В работе рассмотрены задачи: моделирование движения тела, брошенного под углом к горизонту; моделирование движения космических тел; моделирование движения тела с переменной массой – взлета ракеты. Все эти задачи реализованы на языке C++ и отлажены в среде MS Visual Studio. Визуализация результатов моделирования разработана с применением графических библиотек OpenGL и GLUT.

Квалификационная работа состоит из введения, двух глав, заключения и списка литературы. Первая глава включает в себя основы работы с графическими библиотеками OpenGL и GLUT. Вторая глава включает в себя описание математических моделей физических задач, методов их решения, а также кодов программ, реализующих решения на C++. Также, во второй главе показаны результаты решения в виде траекторий и графиков построенных с использованием Visual Studio и GLUT.

Глава I. Основы работы с OpenGL и GLUT

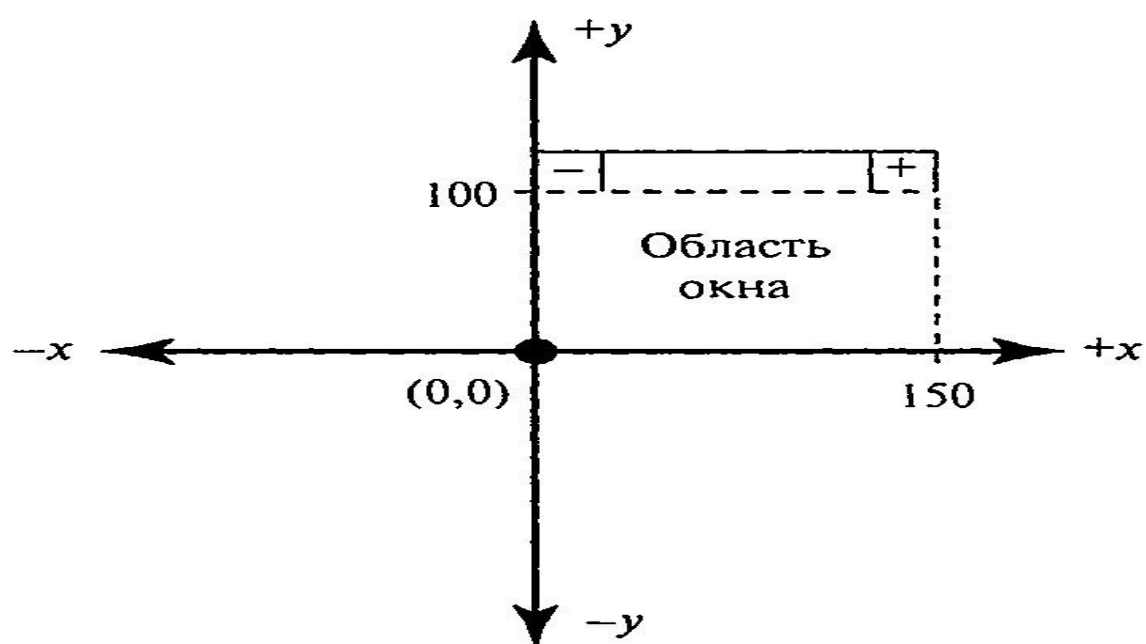
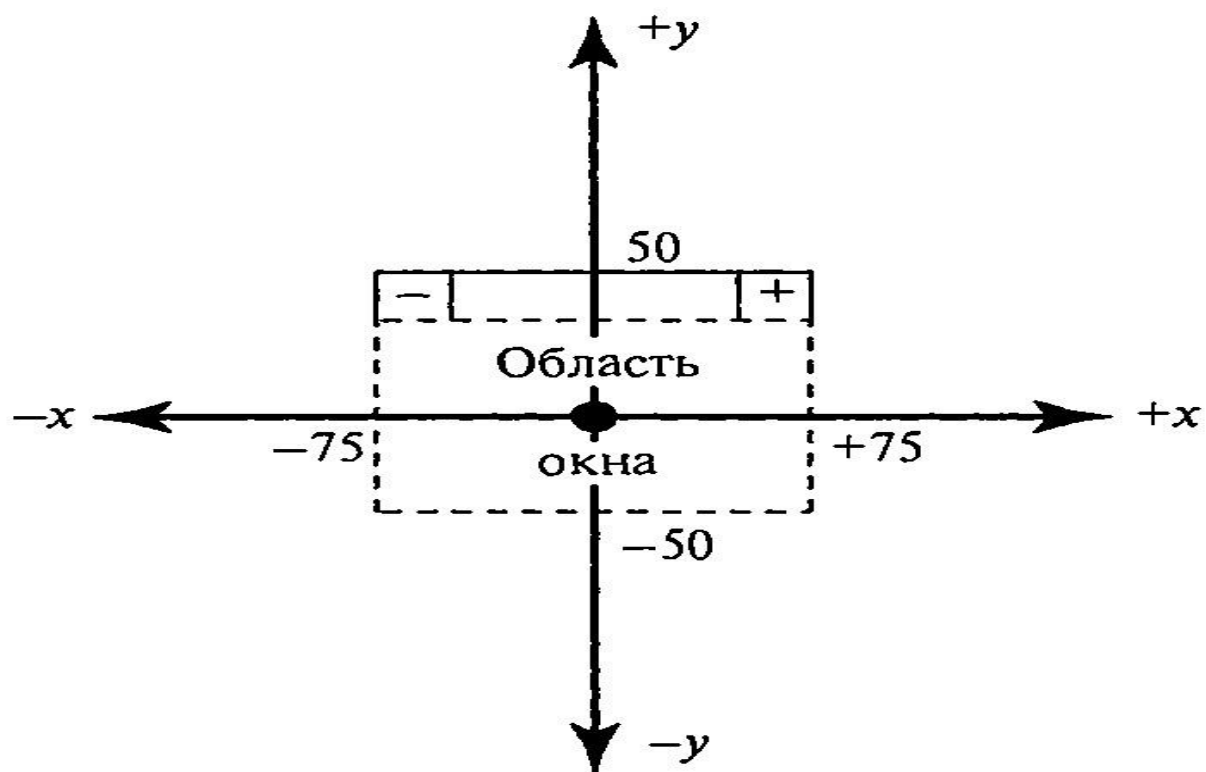
1.1. Основы OpenGL

OpenGL включает более 300 функций для рисования сложных трехмерных сцен из простых примитивов. Используется при создании компьютерных видеоигр, САПР, виртуальной реальности, визуализации в научных исследованиях.

OpenGL – это просто спецификация, то есть документ, описывающий набор функций и их точное поведение. Производители оборудования на основе этой спецификации создают реализации – библиотеку функций, соответствующих набору функций спецификации.

Визуализация — это превращение геометрического описания трехмерного объекта в изображение этого объекта на экране.

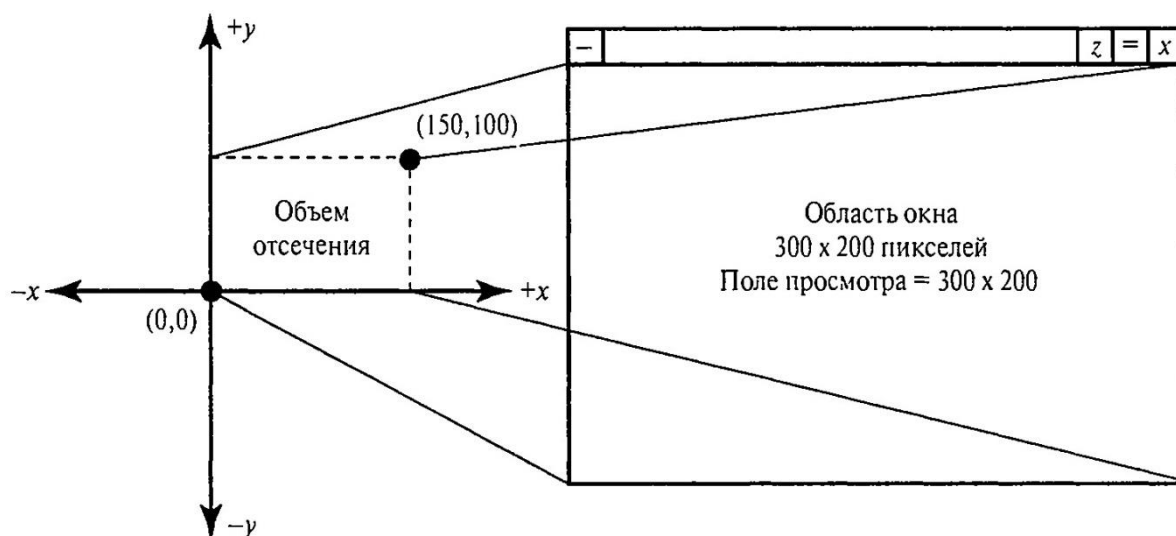
Физически окно измеряется в пикселях. Прежде чем вы сможете начать изображать в окне точки, линии и формы, вы должны сообщить OpenGL, как переводить указанные пары значений в экранные координаты. Для этого вы задаете область декартового пространства, которую занимает окно; она называется областью отсечения. В двухмерном пространстве область отсечения – это минимальные и максимальные значения x и y , которые принадлежат окну. Мы можем описать это и по-другому, задав положение начала системы координат относительно окна. Два распространенных выбора областей отсечения показаны на следующих рисунках:



В первом примере координаты x в окне меняются слева направо от 0 до +150, а координаты y — снизу вверх от 0 до +100. Точка в центре экрана будет представлена как (75, 50). На втором рисунке показана область отсечения, координата x которой меняется слева направо от -75 до $+75$, а координата y — снизу вверх от -50 до $+50$. В этом примере точка в центре экрана будет совпадать с началом координат — точкой (0,0).

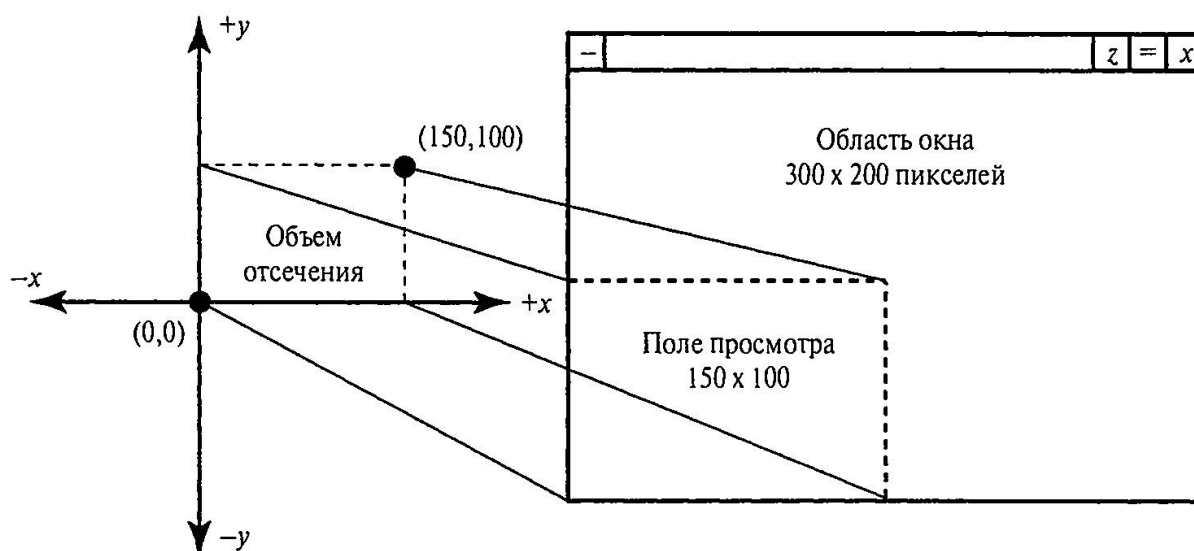
Очень редко ширина и высота области отсечения точно совпадает с шириной и высотой окна в пикселях. Следовательно, нужно преобразовать систему координат: отобразить логические декартовы координаты в физические координаты пикселей экрана. Такое отображение задается с помощью *поля просмотра*. Поле просмотра — это область внутри клиента окна, которая используется для рисования области отсечения. Поле просмотра просто отображает область отсечения в область окна. Обычно поле просмотра определяется как все окно, но это не является строго необходимым, например, вы можете указать, что рисовать разрешается только в нижней половине окна.

На следующем рисунке показано большое окно, насчитывающее 300 x 200 пикселей, с полем просмотра, определенным как вся область клиента. Если установить, что область отсечения этого окна простирается от 0 до 150 вдоль оси x и от 0 до 100 вдоль оси y , логические координаты будут отображены в большую систему экранных координат в окне наблюдения. Каждый элемент в логической системе координат будет удвоен в физической системе координат (пиксели) окна.



А на следующем рисунке показано поле просмотра, равное области отсечения. Окно наблюдения по-прежнему равно 300 х 200 пикселей, поэтому область наблюдения занимает левую нижнюю область окна.

С помощью поля просмотра можно сжимать или растягивать изображения внутри окна, а также отображать только часть области отсечения (в таком случае поле просмотра задается большим областью клиента окна).



При рисовании двух- или трехмерного объекта вы составляете его из нескольких меньших форм, именуемых примитивами. Примитивы — это такие одно- или двумерные объекты или поверхности, как точки, линии и многоугольники (плоские формы с несколькими сторонами), собираемые в трехмерном пространстве для создания трехмерных объектов. Например, трехмерный куб состоит из шести двумерных квадратов, размещенных в разных плоскостях. Каждый угол квадрата (или любого примитива) называется вершиной. Этим вершинам сопоставляются координаты в трехмерном пространстве. Вершина — это всего лишь набор координат в двух- или трехмерном пространстве.

В OpenGL есть следующие типы данных:

Тип данных OpenGL	Внутреннее представление	Определение в форме типа C	Суффикс литералов C
GLbyte	8-битовое целое	signed char	b
GLshort	16-битовое целое	short	s
GLint, GLsizei	32-битовое целое	long	l
GLfloat, GLclampf	32-битовое с плавающей запятой	float	f
GLdouble, GLclampd	64-битовое с плавающей запятой	double	d
GLubyte, GLboolean	8-битовое целое без знака	unsigned char	ub
GLushort	16-битовое целое без знака	unsigned short	us
GLuint, GLenum, GLbitfield	32-битовое целое без знака	unsigned long	ui

Для большинства функций OpenGL используются правила именования, позволяющие сообщить, из какой библиотеки пришла функция, сколько она принимает аргументов и какого типа. Все функции имеют корень, представляющий соответствующую функции команду OpenGL. Например, *glColor3f* имеет корень *Color*. Префикс *gl* представляет библиотеку *gl*, а суффикс *3f* означает, что функция принимает три аргумента с плавающими запятыми. Все функции OpenGL имеют следующий формат.

<Префикс библиотеки><Команда основной библиотеки><Необязательный счетчик аргументов><Необязательный тип аргументов>

На следующем рисунке приведен пример:



Все графические объекты в OpenGL представляют собой набор точек, линий и многоугольников.

Существует 10 различных примитивов, при помощи которых строятся все объекты.

Как двухмерные, так и трехмерные. Все примитивы в свою очередь задаются точками — вершинами.

- **GL_POINTS** — каждая вершина задает точку
- **GL_LINES** — каждая отдельная пара вершин задает линию

- **GL_LINE_STRIP** — каждая пара вершин задает линию (то есть конец предыдущей линии является началом следующей)
- **GL_LINE_LOOP** — аналогично предыдущему за исключением того, что последняя вершина соединяется с первой и получается замкнутая фигура
- **GL_TRIANGLES** — каждая отдельная тройка вершин задает треугольник
- **GL_TRIANGLE_STRIP** — каждая следующая вершина задает треугольник вместе с двумя предыдущими (получается лента из треугольников)
- **GL_TRIANGLE_FAN** — каждый треугольник задается первой вершиной и последующими парами (то есть треугольники строятся вокруг первой вершины, образуя нечто похожее на диафрагму)
- **GL_QUADS** — каждые четыре вершины образуют четырехугольник
- **GL_QUAD_STRIP** — каждая следующая пара вершин образует четырехугольник вместе с парой предыдущих
- **GL_POLYGON** — задает многоугольник с количеством углов равным количеству заданных вершин

Для задания примитива используется конструкция `glBegin (ТИП_ПРИМИТИВА)...glEnd ()`. Вершины задаются `glVertex*`.

Вершины задаются против часовой стрелки. Координаты задаются от верхнего левого угла окна. Цвет вершины задается командой `glColor*`.

Цвет задается в виде RGB или RGBA. Команда `glColor*` действует на все вершины, что идут после до тех пор, пока не встретится другая команда `glColor*` или же на все, если других команд `glColor*` нет. Вот код рисующий квадрат с разноцветными вершинами и очищает текущим цветом:

```
glBegin(GL_QUADS);
glClear(GL_COLOR_BUFFER_BIT);
```

```
glColor3f(1.0f, 1.0f, 1.0f);  
glVertex2f(250f, 450f);  
glColor3f(0.0f, 1.0f, 1.0f);  
glVertex2f(250.0f, 150.0f);  
glColor3f(1.0f, 1.0f, 0.0f);  
glVertex2f(550.0f, 150.0f);  
glColor3f(1.0f, 0.0f, 0.0f);  
glVertex2f(550.0f, 450.0f);  
glEnd();
```

1.2. Основы работы с GLUT

Для платформонезависимой работы с окнами используем библиотеку GLUT. GLUT упрощает работу с OpenGL.

Для инициализации GLUT в начале программы надо вызвать **glutInit (&argc, argv)**.

Для задания режима дисплея вызывается **glutInitDisplayMode (режим)**, где режим может принимать следующие значения: Для инициализации GLUT в начале программы надо вызвать **glutInit (&argc, argv)**. Для задания режима дисплея вызывается **glutInitDisplayMode (режим)**, где режим может принимать следующие значения:

GLUT_RGBA — включает четырехкомпонентный цвет (используется по умолчанию)

GLUT_RGB — то же, что и **GLUT_RGBA**

GLUT_INDEX — включает индексированный цвет

GLUT_DOUBLE — включает двойной экранный буфер

GLUT_SINGLE — включает одиночный экранный буфер (по умолчанию)

GLUT_DEPTH — включает Z-буфер (буфер глубины)
GLUT_STENCIL — включает трафаретный буфер
GLUT_ACCUM — включает буфер накопления
GLUT_ALPHA — включает альфа-смешивание (прозрачность)
GLUT_MULTISAMPLE — включает мультисемплинг (сглаживание)
GLUT_STEREO — включает стерео-изображение

Для выбора нескольких режимов одновременно нужно использовать побитовое ИЛИ '|'. Например: **glutInitDisplayMode (GLUT_DOUBLE|GLUT_RGBA|GLUT_DEPTH)** включает двойную буферизацию, Z-буфер и четырехкомпонентный цвет.

Размеры окна задаются **glutInitWindowSize (ширина, высота)**.

Его позиция — **glutInitWindowPosition (x, y)**.

Создается окно функцией **glutCreateWindow (заголовок_окна)**.

GLUT реализует событийно-управляемый механизм. Т.е. есть главный цикл, который запускается после инициализации, и в нем уже обрабатываются все объявленные события. Например, нажатие клавиши на клавиатуре или движение курсора мыши и т.д. Зарегистрировать функции-обработчики событий можно при помощи следующих команд:

void glutDisplayFunc (void (*func) (void)) — задает функцию рисования изображения

void glutReshapeFunc (void (*func) (int width, int height)) — задает функцию обработки изменения размеров окна

void glutVisibilityFunc (void (*func)(int state)) — задает функцию обработки изменения состояния видимости окна

void glutKeyboardFunc (void (*func)(unsigned char key, int x, int y)) — задает функцию обработки нажатия клавиш клавиатуры (только тех, что генерируют ascii-символы)

void glutSpecialFunc (void (*func)(int key, int x, int y)) — задает функцию обработки нажатия клавиш клавиатуры (тех, что не генерируют ascii-символы)

void glutIdleFunc (void (*func) (void)) — задает функцию, вызываемую при отсутствии других событий

void glutMouseFunc (void (*func) (int button, int state, int x, int y)) — задает функцию, обрабатывающую команды мыши

void glutMotionFunc (void (*func)(int x, int y)) — задает функцию, обрабатывающую движение курсора мыши, когда зажата какая-либо кнопка мыши

void glutPassiveMotionFunc (void (*func)(int x, int y)) — задает функцию, обрабатывающую движение курсора мыши, когда не зажато ни одной кнопки мыши

void glutEntryFunc (void (*func)(int state)) — задает функцию, обрабатывающую движение курсора за пределы окна и его возвращение

void glutTimerFunc (unsigned int msecs, void (*func)(int value), value) — задает функцию, вызываемую по таймеру.

Затем можно запускать главный цикл **glutMainLoop ()**.

1.3. Настройка GLUT на Visual Studio 2013

Для того чтобы программы запускались на Visual Studio надо сообщить систему что используется GLUT. Настройка будет выглядеть следующим образом:

1. Открыть любой браузер и ввести следующий ссылку на адресной строке: <https://www.opengl.org/resources/libraries/glut/>

2. Распаковываем то, что скачали в общую папку. Например, на рабочий стол в папку “GL”
3. Из этой папки копируем **glut32.dll** файл на системную папку (Если у Вас x86 архитектура, то в C:\Windows\System32. А если у Вас x64 архитектура, то в C:\Windows\SysWOW64).
4. **Glut32.lib** копируем на lib папку, куда был установлен Visual Studio.(
C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\lib)
5. **glut.h** копируем на Include папку (C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\include)

Глава II. Решение физических проблем методами компьютерного математического моделирования. Их визуализация

Математические модели можно разделить на следующие группы:

- детерминированные
- стохастические
- эмпирико-статистические

Детерминированные – основаны на внутреннем описании системы (модель Мальтуса).

Стохастические – модели, которые используют вероятности.

Отметим, что, говоря о математических моделях, мы имеем в виду сугубо прикладной аспект. В современной математике есть достаточно формализованный подход к понятию «математическая модель». Внутри него вполне допустимо игнорировать вопрос о связи математики с физической реальностью. В этом подходе моделями являются, например, система целых чисел, система действительных чисел, евклидова геометрия, алгебраическая группа, топологическое пространство и т.д. К исследованию таких формальных моделей вполне можно подключить компьютеры, но все равно это останется «чистой» математикой.

В физике математическое моделирование является чрезвычайно важным методом исследования. Наряду с традиционным делением физики на экспериментальную и теоретическую сегодня уверенно выделяется третий фундаментальный раздел – вычислительная физика. Причину этого в целом можно сформулировать так: при максимальном проникновении в физику математических методов, порой доходящем до фактического сращивания этих наук, реальные возможности решения возникающих математических задач традиционными методами очень ограничены. Из многих конкретных причин выделим две наиболее часто встречающихся:

- нелинейность многих физических процессов и отсюда нелинейность описывающих их математических моделей
- необходимость исследования совместного движения многих тел, для которого приходится решать системы большого числа уравнений.

Численное моделирование в физике называют вычислительным экспериментом, поскольку оно имеет много общего с лабораторным экспериментом (Таблица 1).

Таблица 1. Аналогии между лабораторным и вычислительным экспериментами

лабораторный эксперимент	вычислительный эксперимент
Образец	Модель
Физический прибор	Программа для компьютера
Калибровка прибора	Тестирование программы
Измерение	Расчёт
Анализ данных	Анализ данных

Численное моделирование (как и лабораторные эксперименты) чаще всего является инструментом познания качественных закономерностей природы. Важнейшим его этапом является анализ результатов, представление их в максимально наглядной и удобной для восприятия форме. Получение распечатки чисел еще не означает окончания моделирования (даже если эти числа верны). Важно представить результаты в виде графиков, диаграмм, траекторий движения динамических объектов для получения качественной информации. Здесь необходима помощь компьютера – возможность визуализации абстракций.

2.1. Моделирование движение тела брошенного под углом к горизонту

2.1.1. Математическая модель движения тела брошенного под углом к горизонту без учета сопротивления воздуха.

Будучи брошенным, под углом α к горизонту с начальной скоростью v_0 , тело летит без учета сопротивления воздуха по параболе и через некоторое время падает на землю.

Решим задачу о брошенном под углом теле без учета сопротивления воздуха.

Разложим скорость v_0 на горизонтальную и вертикальную составляющие:

$$v_x^{(0)} = v_0 \cos \alpha, \quad v_y^{(0)} = v_0 \sin \alpha. \quad (1)$$

Движение по вертикали не равномерно. Оно является равнозамедленным до достижения верхней точки на траектории и равноускоренным после неё. Движение по горизонтали является равномерным. Для вертикальной составляющей $v_y = v_y^{(0)} - gt$; Вычислим время достижения верхней точки траектории. Имеем в верхней точке $v_y = 0$, тогда в верхней точке $\{v_y = v_y^{(0)} - gt\}$, $v_y^{(0)} = gt$. Отсюда время достижения верхней точки:

$$\tilde{t} = \frac{v_y^{(0)}}{g} = \frac{v_0 \sin \alpha}{g} \quad (2)$$

Высота этой точки h равна:

$$h = v_y^{(0)} \tilde{t} - \frac{g \tilde{t}^2}{2} = \frac{v_0^2 \sin^2 \alpha}{g} - \frac{g v_0^2 \sin^2 \alpha}{2g^2} = \frac{v_0^2 \sin^2 \alpha}{2g} \quad (3)$$

Полное время до падения на землю $2\tilde{t}$; за это время, двигаясь равномерно вдоль оси x со скоростью $v_x^{(0)}$, тело пройдет путь:

$$l = v_x^{(0)} \cdot 2\tilde{t} = v_0 \cos \alpha \cdot 2 \frac{v_0 \sin \alpha}{g} = \frac{v_0^2 \sin 2\alpha}{g} \quad (4)$$

Для нахождения траектории достаточно из текущих значений x и y исключить t :

$$x = v_x^{(0)} t, \quad y = v_y^{(0)} t - \frac{gt^2}{2}; \quad (5)$$

следовательно:

$$y = v_y^{(0)} \frac{x}{v_x^{(0)}} - \frac{g}{2} \cdot \frac{x^2}{v_x^{(0)2}} = \operatorname{tg} \alpha \cdot x - \frac{g}{2v_0^2 \cos^2 \alpha} x^2 \quad (6)$$

Уравнение (6) – уравнение параболы.

Программа построение траектории движения брошенного под углом к горизонту.

Код программы:

```
#include<glut.h>
#include<cmath>

const double PI = 3.1415;
float V = 20.0f;
float alpha = 30;
double cx, cy;
int m;
const double g = 9.8;

void render_scene(){
    glClear(GL_COLOR_BUFFER_BIT);
```

```

glColor3f(1.0, 0.0, 0.0);
glBegin(GL_LINES);
glVertex2f(.0, .0);
glVertex2f(cx, .0);
glEnd();

glBegin(GL_LINES);
glVertex2f(.0, .0);
glVertex2f(.0, cy);
glEnd();

/*
Изменение цвета рисовки
*/
glColor3f(0.5, 0.8, 0.4);
double x = 0.0, y;
alpha *= PI / 180.0;
double t = .0;

glBegin(GL_POINTS);
while (t <= (V*sin(alpha) / g))
{
    x = V*cos(alpha)*t;
    y = V*sin(alpha)*t - g*t*t / 2;
    glVertex2d(x, y);
    t += 0.01;
}
double h = y;
double l = x;
t = 0.0;

```

```

while (t <= (V*sin(alpha) / g))
{
    x = l + V*cos(alpha)*t;
    y = h - g*t*t / 2;
    glVertex2d(x, y);
    t += 0.01;
}
glEnd();

glutSwapBuffers();
}

void change_size(GLsizei w, GLsizei h){
    if (h == 0)
        h = 1;

    cx = V*V*sin(2 * (alpha*PI / 180.0)) / g + 15;
    cy = V*V*sin(alpha*PI / 180.0)*sin(alpha*PI / 180.0) /
(2 * g) + 15;
    glViewport(5, 5, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, cx, 0.0, cy );

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void Initialise(){

```

```

        glClearColor(0, 0, 0, 0);
    }

    void check_menu(int v){
        if (v == 0)
            glutSetWindowTitle("Hi");
    }

    void checkS2menu(int v){
        switch (v)
        {
            case 6:
                alpha = 30;
                glutPostRedisplay();
                break;
            case 7:
                alpha = 45;
                glutPostRedisplay();
                break;
            case 8:
                alpha = 60;
                glutPostRedisplay();
                break;
        }
    }

    void initialMenu(){

        int SM2 = glutCreateMenu(checkS2menu);
    }

```

```

    glutSetMenu(SM2);
    glutAddMenuEntry("30 градусов", 6);
    glutAddMenuEntry("45 градусов", 7);
    glutAddMenuEntry("60 градусов", 8);

    int M = glutCreateMenu(check_menu);
    glutSetMenu(M);
    glutAddSubMenu("Угол равен", SM2);

    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

int main(int argc, char **argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(480, 360);
    glutInitWindowPosition(40, 60);
    glutCreateWindow("Бросание под углом тела к горизонту
без учета сопротивления воздуха");
    glutDisplayFunc(render_scene);
    glutReshapeFunc(change_size);
    Initialise();
    initialMenu();

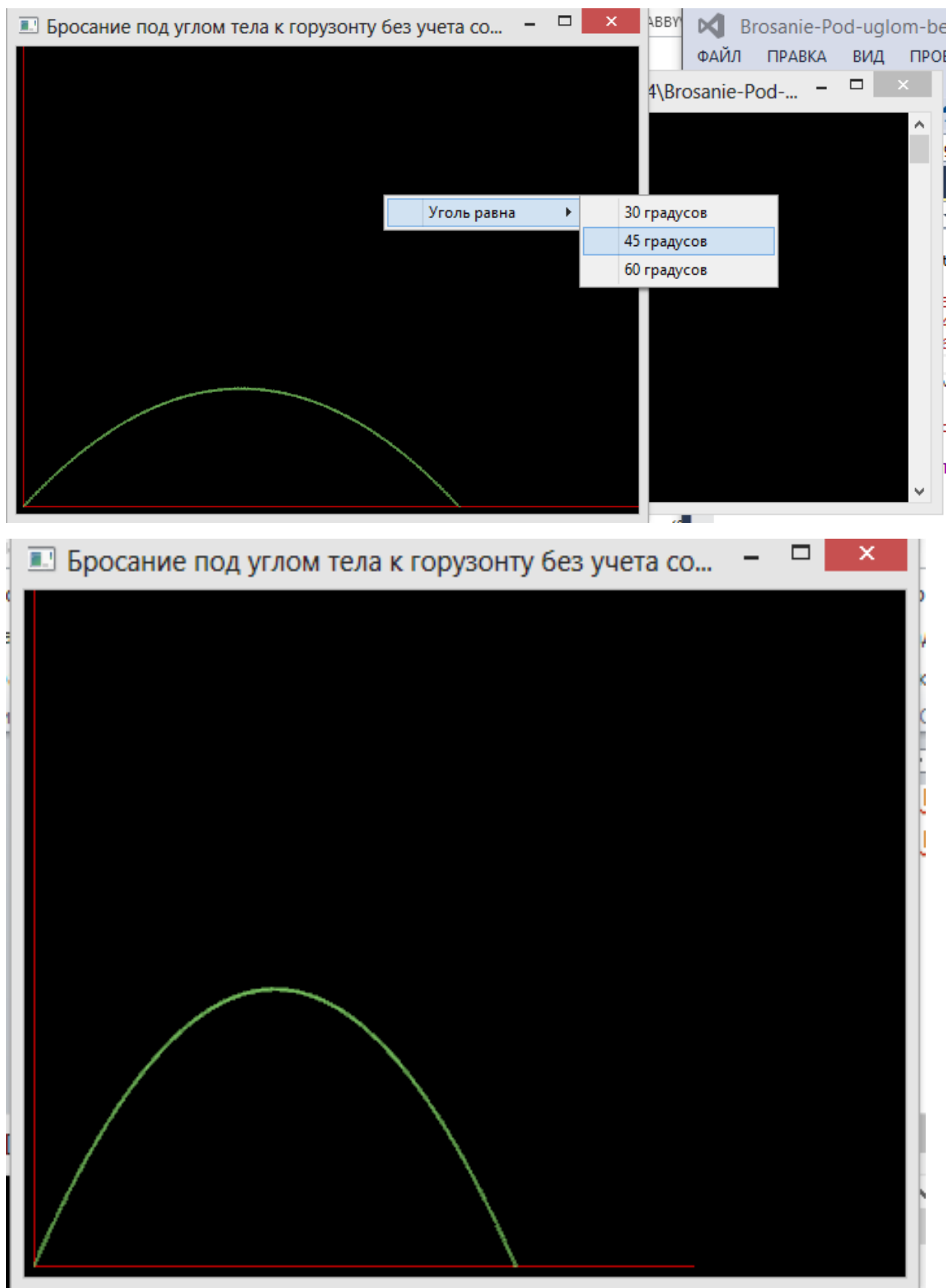
    glutMainLoop();

    return 0;
}

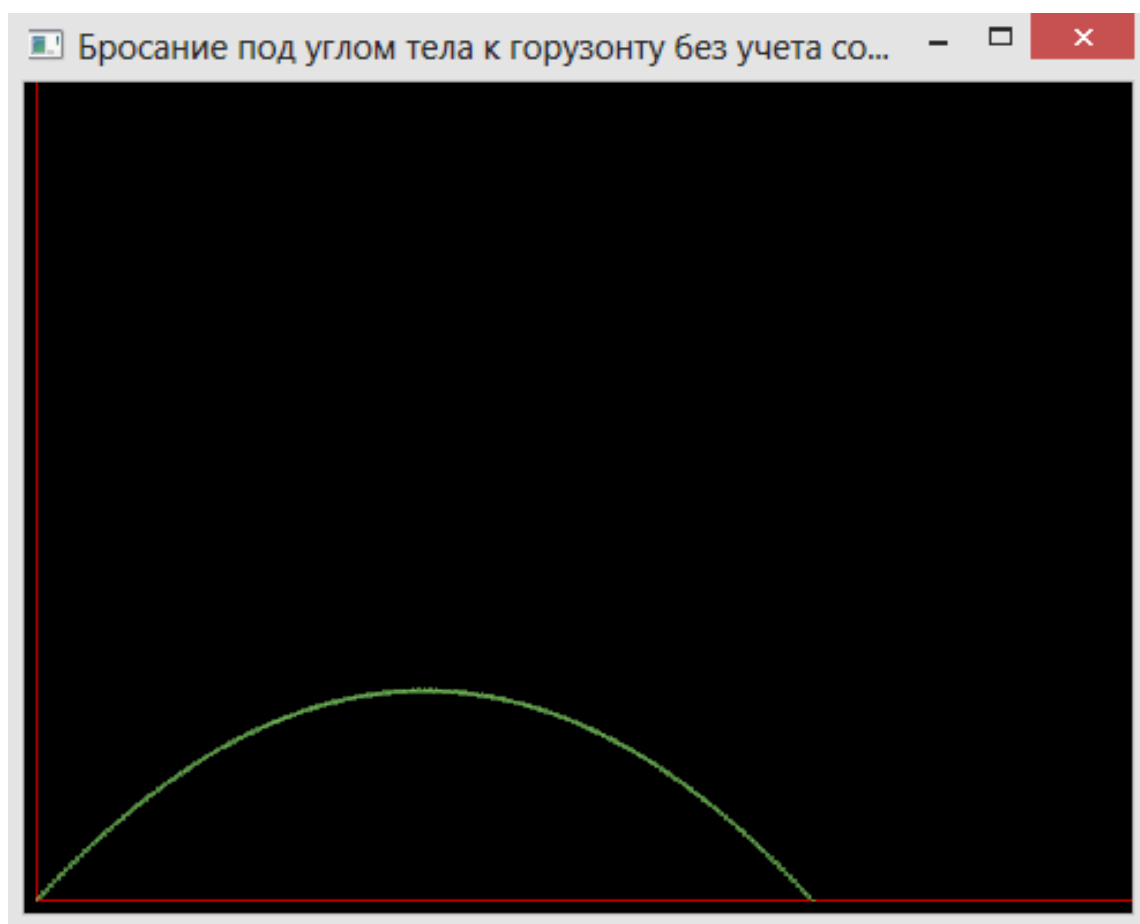
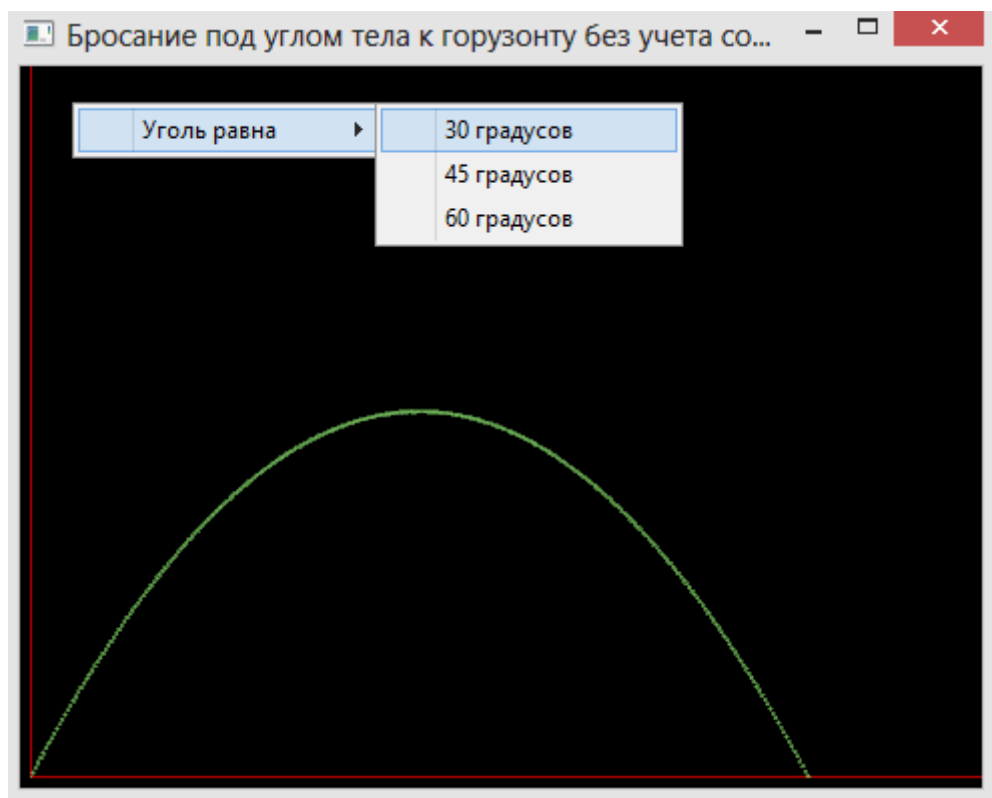
```

Результаты работы:

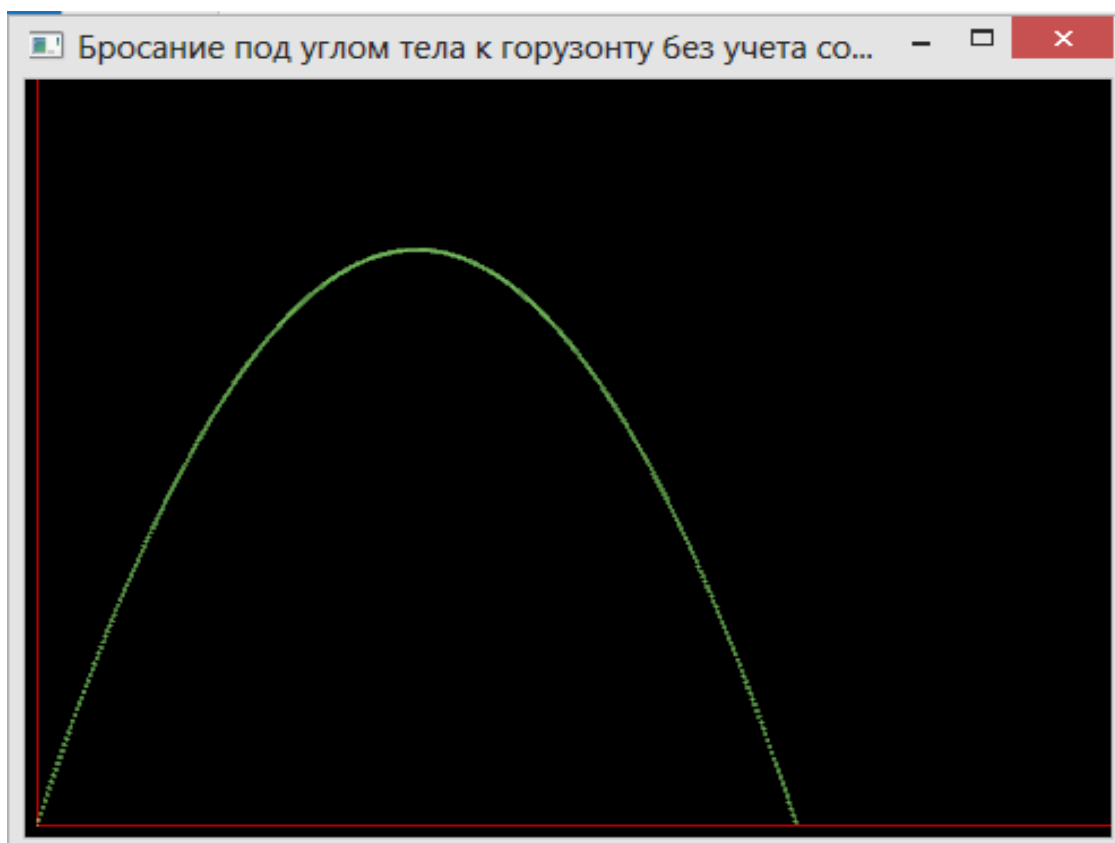
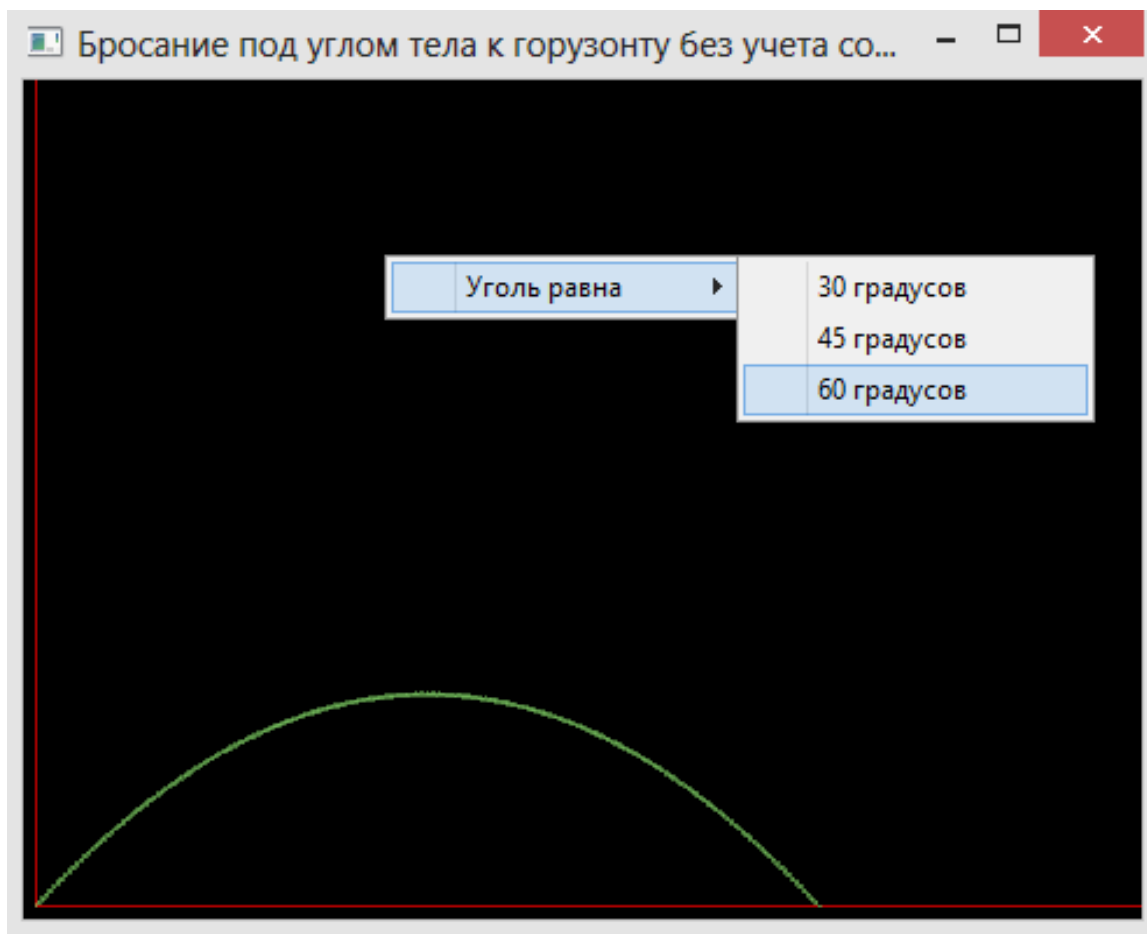
А) $v_0 = 20 \text{ м/с}$, $\alpha = 45$



Б) $v_0 = 20$, $\alpha = 30$



В) $v_0 = 20$, $\alpha = 60$



2.1.2. Математическая модель движения тела брошенного под углом к горизонту с учетом сопротивления воздуха

Рассмотрим эту задачу с учетом сопротивления воздуха. При большой начальной скорости полета тела, сопротивление воздуха может значительно изменить движение. Приступим к ранжированию и выясним, какой из составляющих силы сопротивления (линейной или квадратичной) можно пренебречь.

Оценку проведем для шарика. (Она не зависит от формы тела). Шарик радиусом $r \approx 0,1$ м, движущийся со скоростью 1 м/с, испытывает в воздухе линейную (по закону Стокса) силу сопротивления

$$F_1 = 6\pi\mu r v \approx 3 \cdot 10^{-2} \text{ Н}$$

и квадратичную силу сопротивления

$$F_2 = \frac{cS\rho v^2}{2} \approx 8 \cdot 10^{-3} \text{ Н}$$

Мы видим, что F_1 и F_2 различаются менее, чем в пять раз. Они одного порядка. Поэтому, если исследуется движение брошенного мяча, учитывается обе составляющие. Однако если моделируется полет снаряда, где скорость - сотни метров в секунду, то линейной составляющей можно

пренебречь. Запишем проекции уравнения $\frac{d\vec{v}}{dt} = \frac{\vec{F}}{m}$ (3. Ньютона) на оси x и y , получим:

$$\frac{dv_x}{dt} = \frac{F_{\text{comp}}^x}{m}, \quad \frac{dv_y}{dt} = \frac{-mg + F_{\text{comp}}^y}{m} \quad (1)$$

Поскольку в каждой точке траектории сила сопротивления направлено по касательной к траектории в сторону, противоположную движению, то

$$F_{conp}^x = -F_{conp} \cos \theta = -F_{conp} \frac{v_x}{v} = -(k_1 v + k_2 v^2) \frac{v_x}{v} \quad (2)$$

$$F_{conp}^y = -F_{conp} \sin \theta = -F_{conp} \frac{v_y}{v} = -(k_1 v + k_2 v^2) \frac{v_y}{v}$$

где θ – угол между текущим направлением скорости и осью x .
Подставляя (2) в уравнения (1) и учитывая, что $v = \sqrt{(v_x^2 + v_y^2)}$, получаем уравнения движения в переменных v_x, v_y :

$$\begin{cases} \frac{dv_x}{dt} = -\frac{k_1 + k_2 \sqrt{(v_x^2 + v_y^2)}}{m} v_x \\ \frac{dv_y}{dt} = -g - \frac{k_1 + k_2 \sqrt{(v_x^2 + v_y^2)}}{m} v_y \end{cases} \quad (3)$$

Поскольку представляет несомненный интерес и траектория движения, дополним систему (3) еще двумя уравнениями

$$\frac{dx}{dt} = v_x, \frac{dy}{dt} = v_y, \quad (4)$$

Решая систему (3) и (4) получим четыре функции: $v_x(t), v_y(t), x(t), y(t)$. (3) и (4) описывают движение с учетом сопротивления среды.

На примере рассмотренной выше задачи, используем очень важный и полезный прием, популярный в физическом моделировании, называемый обезразмериванием. При решении задач мы пользуемся системой единиц (СИ), в которой далеко не все числовые значения находятся в удобном

диапазоне. Кроме того, абсолютные значения величин дают мало информации для качественного понимания. Скорость 15 м/с –это много или мало? Все дело - по сравнению с чем.

Идея обезмеривания заключается в переходе от абсолютных значений (расстояний, V , t), скоростей, времен и т.д. к относительным, причем отношения строятся к величинам типичным для данной ситуации. В рассматриваемой ситуации это хорошо просматривается. Сопротивление воздуха изменит характер движения, и если мы введем в качестве переменных величины

$$X = \frac{x}{l}, \quad Y = \frac{y}{h}, \quad \tau = \frac{t}{\tilde{t}}$$

X , Y , τ - безразмерные расстояния по осям и времени - тогда при отсутствии сопротивления воздуха они будут изменяться в диапазоне от 0 до 1, а в задаче с учетом сопротивления отличия их максимальных значений от единицы ясно характеризуют влияние этого сопротивления. Для скоростей естественно ввести безразмерные переменные, соотнося проекции скорости на оси x и y с начальной скоростью v_0 :

$$V_x = \frac{v_x}{v_0}, \quad V_y = \frac{v_y}{v_0}.$$

Покажем, как перейти к безразмерным переменным в уравнениях системы (3) на примере второго уравнения этой системы.

Имеем:

$$\frac{dv_y}{dt} = \frac{d(v_0 V_y)}{d(\tilde{t} \cdot \tau)} = \frac{v_0}{\tilde{t}} \cdot \frac{dV_y}{d\tau},$$

так как постоянный множитель можно вынести за знак производной.

Подставляя это в уравнение (3), получаем:

$$\frac{v_0}{\tilde{t}} \cdot \frac{dV_y}{d\tau} = -g - \frac{k_1 + k_2 \sqrt{(v_0^2 V_x^2 + v_0^2 V_y^2)}}{m} v_0 V_y$$

или

$$\frac{dV_y}{d\tau} = -\frac{\tilde{t} g}{v_0} - \frac{\tilde{t}}{v_0} \cdot \frac{k_1 + k_2 \sqrt{(v_0^2 V_x^2 + v_0^2 V_y^2)}}{m} v_0 V_y$$

Подставляя $\tilde{t} = \frac{v_0 \sin \alpha}{g}$, получаем:

$$\frac{dV_y}{d\tau} = -\sin \alpha - \underline{a} \cdot \sin \alpha \cdot V_y - \underline{b} \cdot \sin \alpha \cdot \sqrt{(V_x^2 + V_y^2)} \cdot V_y,$$

где \underline{a} , \underline{b} -- безразмерные комбинации параметров, входящих в исходные уравнения,

$$\underline{a} = \frac{k_1 v_0}{mg}, \quad \underline{b} = \frac{k_2 v_0^2}{mg}.$$

Выполним обезразмеривание во всех уравнениях систем (3), (4):

$$\frac{dV_x}{d\tau} = -\underline{a} \cdot \sin \alpha \cdot V_x - \underline{b} \cdot \sin \alpha \cdot \sqrt{(V_x^2 + V_y^2)} \cdot V_x,$$

$$\frac{dX}{d\tau} = \frac{V_x}{2 \cos \alpha}, \tag{5}$$

$$\frac{dV_y}{d\tau} = -\sin \alpha - \underline{a} \cdot \sin \alpha \cdot V_y - \underline{b} \cdot \sin \alpha \cdot \sqrt{(V_x^2 + V_y^2)} \cdot V_y,$$

$$\frac{dY}{d\tau} = \frac{2V_y}{\sin \alpha}.$$

Начальные условия для безразмерных переменных:

$$V_x(0) = \cos \alpha, \quad V_y(0) = \sin \alpha, \quad X(0) = 0, \quad Y(0) = 0$$

Важнейшая роль обезразмеривания – установление законов подобия. У изучаемого движения есть множество вариантов, определяемых наборами значений параметров, входящих в систему уравнений (3) - (4) или являющихся для них начальными условиями: $k_1, k_2, m, g, v_0, \alpha$. После обезразмеривания переменных появляются безразмерные комбинации параметров. В данном случае a, b, α – фактически определяют характер движения. Если мы изучаем два разных движения с разными размерными параметрами, но такие, что a, b, α – одинаковые, тогда движения будут качественно одинаковыми. Число таких комбинаций обычно меньше числа размерных параметров (в данном случае вдвое), что также создает удобства при полном численном исследовании задачи. Наконец, как уже отмечалось, величины V_x, V_y, X, Y, τ физически легче интерпретировать, чем их размерные аналоги, так как они измеряются относительно величин, смысл которых очевиден. Для решения дифференциальных уравнений системы (5) используем метод Эйлера-Коши.

Программа построение траектории движения тела брошенного под углом к горизонту с учетом сопротивления воздуха.

Программа выдает в графическом режиме семейство траекторий, отличающихся значениями одного из трех безразмерных параметров (a – характеризует линейную часть силы сопротивления, b – квадратичную часть силы сопротивления, α – угол бросания).

Код программы:

```
#include<glut.h>
```

```
#include<cmath>
```

```
double x;
```

```
const double pi = 3.1415;
```

```

double A;
double B;
double a = 30;
double Y[4], y[4];
double h = 0.00001, hpr = 0.0001;

double Ff(int i, double *x, double *y){
    switch (i)
    {
        case 0:
            return (-A*sin(a)*y[0] - B*sin(a)*sqrt(y[0] * y[0]
+ y[1] * y[1]))*y[0]);
            break;
        case 1:
            return (-sin(a) - A*sin(a)*y[0] -
B*sin(a)*sqrt(y[0] * y[0] + y[1] * y[1]))*y[1]);
            break;
        case 2:
            return (y[0] / (2 * cos(a)));
            break;
        case 3:
            return (2 * y[1] / sin(a));
            break;
    }
}

```

//Вычисление правой части дифференциального уравнения

```

void RightDiff(double *x, double *y, double *g){
    int i;

```



```

    for (i = 0; i < 4; i++)
        g[i] = Ff(i, x, y);
}

//Реализация алгоритма Рунге-Кутте
void runge_kutte(double *x, double *y0, double *y, double
h){
    int i;
    double z[4], k1[4], k2[4], k3[4], k4[4];

    RightDiff(x, y0, k1); *x = *x + h/2;

    for (i = 0; i < 4; i++)
        z[i] = y0[i] + h*k1[i] / 2;
    RightDiff(x, z, k2);

    for (i = 0; i < 4; i++)
        z[i] = y0[i] + h*k2[i] / 2;
    RightDiff(x, z, k3); *x = *x + h / 2;

    for (i = 0; i < 4; i++)
        z[i] = y0[i] + h*k3[i] / 2;
    RightDiff(x, z, k4);

    for (i = 0; i < 4; i++)
        y[i] = y0[i] + h*(k1[i] + 2*k2[i] + 2*k3[i] +
k4[i])/ 6;
    RightDiff(x, z, k2);

```

```
}
```

```
//Рисует траекторию тела брошенного под  
//углом к горизонту с учетом сопротивления среды
```

```
void render_scene(){  
    //Очищает текущим цветом  
    glClear(GL_COLOR_BUFFER_BIT);
```

```
    //Рисует координатную систему  
    //Задаёт цвет рисовки красным  
    glColor3d(1.0, 0.0, 0.0);
```

```
    glBegin(GL_LINES);  
    glVertex2d(0.0, 0.0);  
    glVertex2d(2.0, 0.0);  
    glEnd();  
    glBegin(GL_LINES);  
    glVertex2d(0.0, 0.0);  
    glVertex2d(0.0, 2.0);  
    glEnd();
```

```
A = 0.5;  
B = 0.04;  
double x0 = 0;  
x = x0;  
y[3] = Y[3];  
int i;
```

```

//Рисует траекторию
int j;
for (j = 1; j <= 4; j++){
    A *= j;
    B *= j;
    //Переводит угол из градуса в радианы
    a *= pi / 180.0;
    //Начальная скорость по оси Ox
    Y[0] = cos(a);
    //Начальная скорость по оси Oy
    Y[1] = sin(a);
    //Начальная координата x
    Y[2] = 0;
    //Начальная координата y
    Y[3] = 0;
    //Задаёт цвет рисовки
    glColor3f(1 - 0.4 / j, 0.4 / j, 0.4 + 0.4 / j);
    x0 = 0;
    x = x0;
    y[3] = Y[3];

    glBegin(GL_POINTS);
    while (y[3] >= 0.0){
        runge_kutte(&x, Y, y, h);
        for (i = 0; i < 4; i++)
            Y[i] = y[i];
        //Рисовать в двумерном пространстве точку
        //в указанной координате
        glVertex2d(Y[2], Y[3]);
    }
}

```

```

        }
        glEnd();
    }

    glutSwapBuffers();
}

void change_size(GLsizei w, GLsizei h){
    if (h == 0)
        h = 1;

    float ratio = w*1.0 / h;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    glViewport(5, 5, w, h);
    gluOrtho2D(0, 2.0, 0, 2.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void Initialise(){
    //Цвет очистки черный
    glClearColor(0.0, 0.0, 0.0, 0.0);
}

int main(int argc, char **argv){
    //Создание окна
    glutInit(&argc, argv);

```

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowSize(640, 480);
glutInitWindowPosition(20, 20);
glutCreateWindow("Promo");

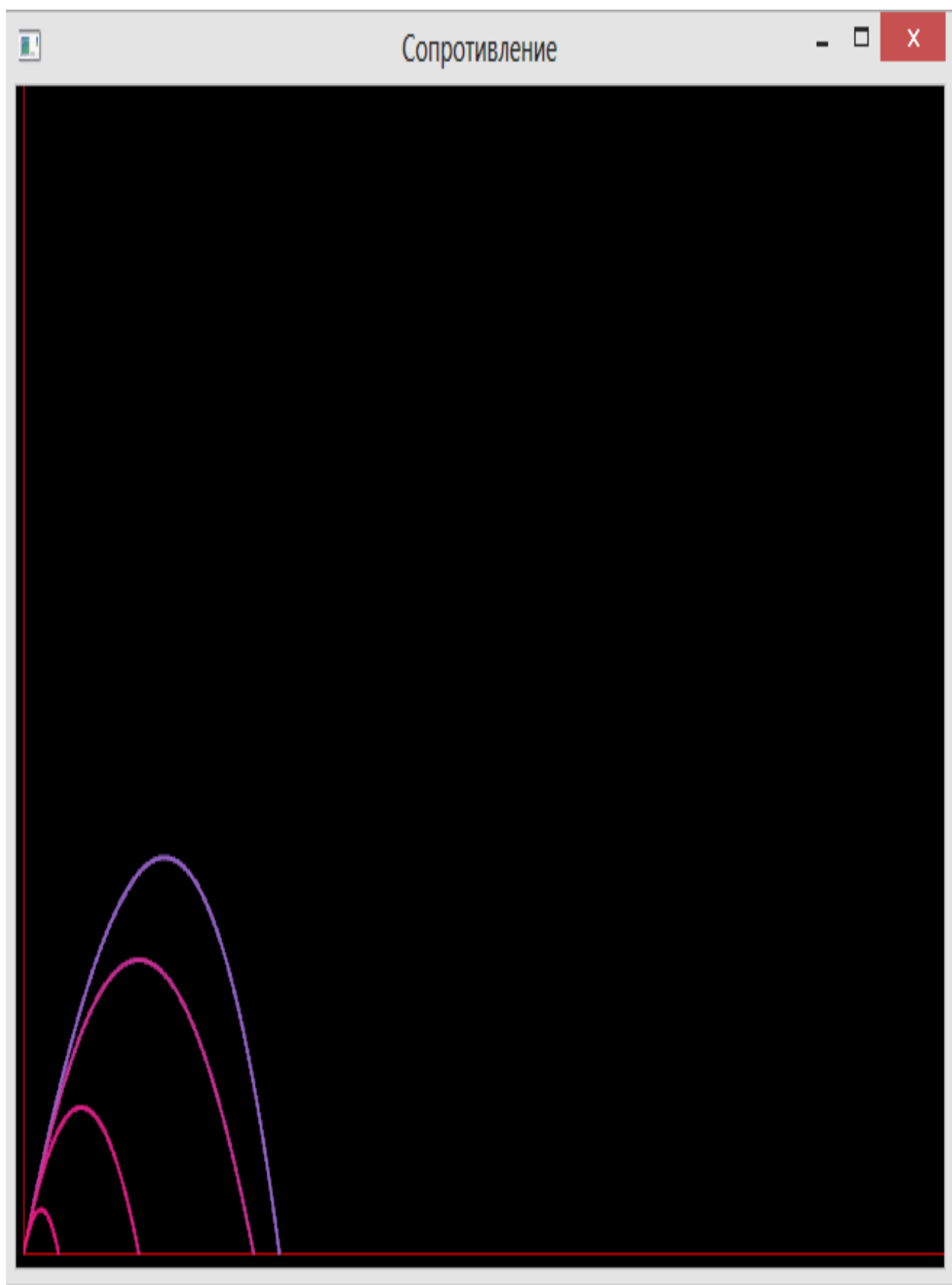
//Вызывание обратных функций
glutDisplayFunc(render_scene);
glutReshapeFunc(change_size);
Initialise();
//initMenu();

//Запустить оболочку GLUT
glutMainLoop();

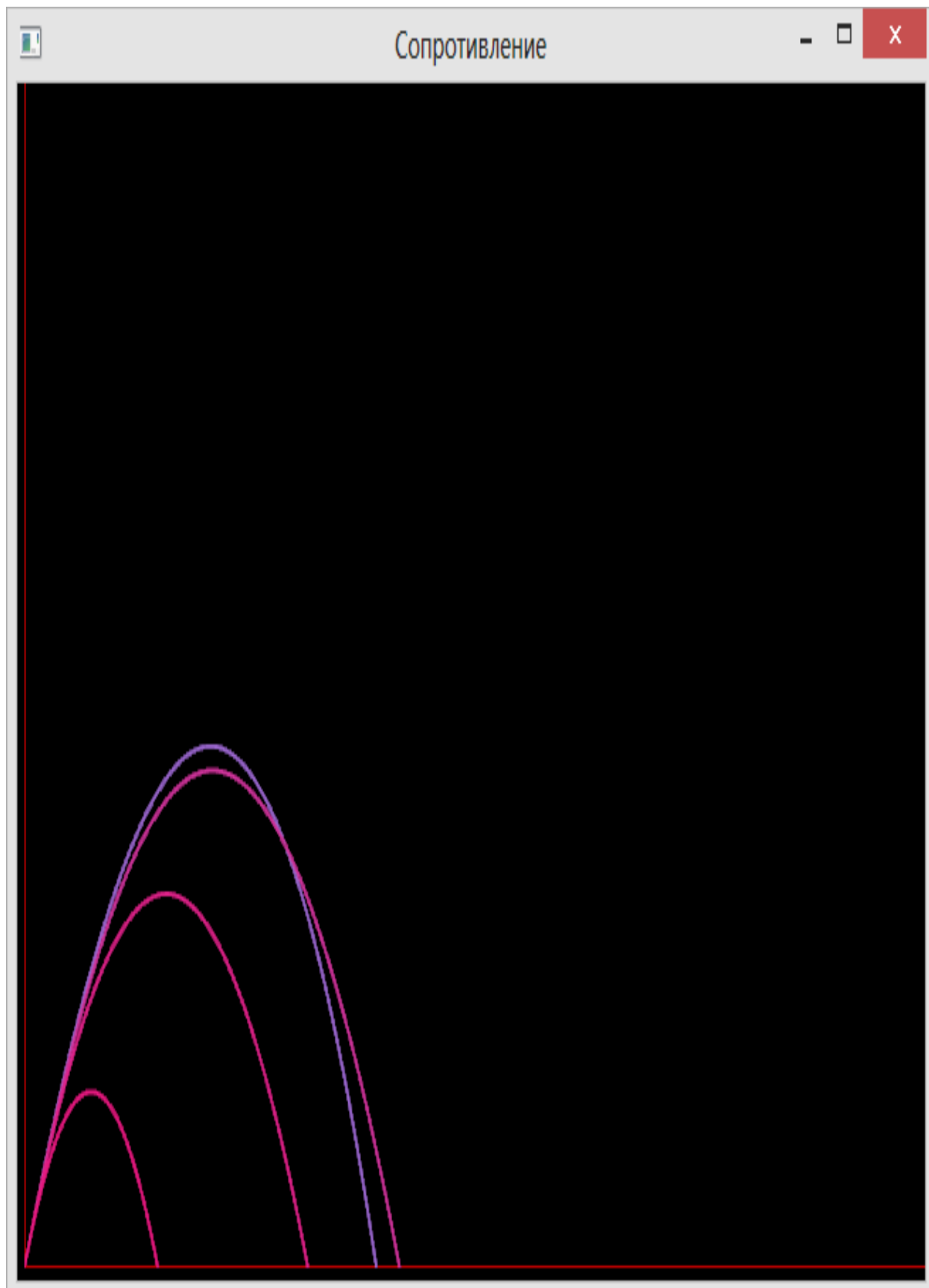
return 0;
}
```

Результаты работы программы

A) $\alpha = 30^\circ$, $\{A = 0.3, 0.6, 0.9, 1.2\}$, $\{B = 0.5, 1.0, 1.5, 2.0\}$



Б) А) $\alpha = 45^\circ$, $\{A = 0.1, 0.2, 0.3, 0.4\}$, $\{B = 0.2, 0.4, 0.6, 0.8\}$



2.2. Математическая модель движения тела с переменной массой – взлета ракеты

Построим простейшую модель вертикального взлета ракеты. Приняв гипотезу, что ее масса уменьшается во время взлета по линейному закону (масса меняется со временем):

$$m(t) = m_0 - \alpha \cdot t, \quad \text{причем } m(t) \geq m_{\text{кон}}$$

где m_0 - начальная масса ракеты, $m_{\text{кон}}$ - конечная масса (т.е. масса полезного груза, выводимого на орбиту), α – расход топлива.

Это допущение согласуется с допущением о постоянной силе тяги.

В данной задаче нельзя пренебречь убыванием массы ракеты в процессе взлета – оно огромно и составляет большую часть исходной массы.

Поиск математического описания проблемы не составляет труда – в его основе все тот же второй закон Ньютона. Поскольку ракета очень быстро набирает высокую скорость, то линейной составляющей $F_{\text{соп}}$ можно пренебречь. Тогда $F_{\text{соп}} = k_2 v^2$, и уравнение для скорости в проекции на вертикальную ось принимает вид

$$\frac{dv}{dt} = \frac{F_{\text{тяги}} - m(t)g - k_2 v^2}{m(t)} \quad (1)$$

В коэффициент k_2 входит величина ρ – плотность окружающей среды, которая на «космических» высотах во много раз меньше, чем на поверхности Земли.

Заглянем в справочник: на высоте 5,5 км плотность воздуха вдвое меньше, чем у поверхности, на высоте 11 км - вчетверо и т.д. Математически зависимость плотности атмосферы от высоты хорошо передается формулой

$$\rho = \rho_0 \exp(-\beta h)$$

где $\beta = 1,29 \cdot 10^{-4}$ (h измеряется в метрах, ρ_0 - плотность вблизи поверхности Земли). Поскольку величина h меняется в ходе полета, уравнение для

изменения $h(t)$ следует добавить к уравнению (1) и записать следующую систему дифференциальных уравнений:

$$\begin{cases} \frac{dv}{dt} = \frac{F_{тяги} - m(t)g \cdot \frac{1}{2} c \rho_0 \cdot \exp(-\beta h) \cdot Sv^2(t)}{m(t)}, \\ \frac{dh}{dt} = v(t). \end{cases} \quad (2)$$

Перед решением уравнений удобно обезразмерить переменные. Естественной характерной скоростью в данной задаче является первая космическая скорость $v^* \approx 7,8$ км/с, при которой возможен вывод на орбиту полезного груза; характерное время - момент полной выработки горючего

$$t^* = \frac{(m_0 - m_{кон})}{\alpha}$$

где $m_{кон}$ - масса груза. Реально t^* - две-три минуты. За характерную высоту можно взять, например, h^* - ту, на которой плотность атмосферы уменьшается в 10 раз (примерно 17 км). Последняя величина может показаться несколько произвольной (впрочем, она таковой и является), но все равно удобнее измерять расстояния в данной задаче относительно величины, равной нескольким километрам, чем в метрах в системе СИ. Итак, введя безразмерные переменные

$$V = \frac{v}{v^*}, \quad \tau = \frac{t}{t^*}, \quad H = \frac{h}{h^*},$$

после несложных преобразований получим уравнения

$$\begin{cases} \frac{dV}{d\tau} = \frac{1}{f(\tau)} \cdot [a - b \cdot f(\tau) - p \cdot \exp(-2,3026H) \cdot V^2] \\ \frac{dH}{d\tau} = eV \end{cases} \quad (3)$$

где $f(\tau)$ - известная функция:

$$f(\tau) = \begin{cases} 1 - (1 - k)\tau, & \text{если } \tau \leq 1, \\ k, & \text{если } \tau > 1, \end{cases}$$

а безразмерные параметры a, b, p, e, k выражаются через исходные так:

$$a = \frac{F_{тяги} t^*}{m_0 v^*}, \quad b = \frac{t^* g}{v^*}, \quad p = \frac{0,5 c \rho_0 S v^* t^*}{m_0}, \quad e = \frac{v^* t^*}{h^*}, \quad k = \frac{m_{кон}}{m_0}.$$

$F_{тяги}$ - Сила тяги двигателя(принять постоянной)

α - расход топлива

ρ – плотность окружающей среды

m_0 - начальная масса ракеты

$m_{кон}$ - конечная масса(масса полезного груза, выводимого на орбиту)

То, что $f(\tau)$ определяется двумя формулами, связано с наличием двух этапов полета: до и после выработки топлива. Безразмерное время, разделяющее эти этапы - $\tau = 1$; если к этому моменту безразмерная скорость $V \geq 1$, то первая космическая скорость достигнута, в противном случае - нет. Параметр, a управляет режимом полета; если при достижении величиной V значения, равного единице, топливо еще не все выработано (т.е. $\tau < 1$), можно с этого момента либо положить, $a = 0$ («выключить двигатель»), либо продолжать разгон. Для решения дифференциальных уравнений системы (3) используется метод Эйлера-Коши.

Программа построение зависимости скорости и высоты от времени

Код программы:

```
#include<glut.h>
#include<cmath>
#include<iostream>
using namespace std;

double ro = 1.29;
double g = 9.8;
double dt = 0.1;
double c = 0.45;
```

```

double b = 0.00129;
double F ;
double s = 20;
double alpha;
double m0 ;
double mk ;
double m, a, v, h, t;

//Рисует траекторию тела брошенного под
//углом к горизонту с учетом сопротивления среды
void render_scene(){
    //Очищает текущим цветом
    glClear(GL_COLOR_BUFFER_BIT);

    //Рисует координатную систему
    //Задаёт цвет рисовки красным
    glLineWidth(4.0);
    glColor3d(1.0, 1.0, 1.0);

    glBegin(GL_LINES);
    glVertex2d(0.0, 0.0);
    glVertex2d(1.4, 0.0);
    glEnd();
    glBegin(GL_LINES);
    glVertex2d(0.0, 0.0);
    glVertex2d(0.0, 1.4);
    glEnd();

    glLineWidth(1.0);

```

```

glColor3f(1.0, 0.0, 0.0);
float i;
for (i = 0; i <= 1.3; i = i + 0.25){
    glBegin(GL_LINES);
    glVertex2f(1.0, i);
    glVertex2f(1.0, i + 0.1);
    glEnd();
}

for (i = 0; i <= 1.3; i = i + 0.25){
    glBegin(GL_LINES);
    glVertex2f(i,1.0);
    glVertex2f(i+0.1,1.0);
    glEnd();
}

cout << "F = "; cin >> F;

cout << "m0: "; cin >> m0;
cout << "mk: "; cin >> mk;
cout << "alpha: "; cin >> alpha;
cout << "k = "; cin >> k;
for (i = 0; i <= k; i = i + 1){
    F = F + F*0.5;
    v = 0;
    h = 0;
    t = dt;
    double tk = ((m0 - mk) / alpha);

```

```

        glColor3d(i/10.0, 1.0 - i/10.0, 1.0);
        glBegin(GL_POINTS);
        while ((t <= tk) && (v <= 1.0)){
            m = m0 - alpha*t;
            a = (F - m*g - (1 / 2)*c*ro*exp(-b*h)*s*v*v) /
m;

            v = v + a*dt / 7800;
            h = h + v*dt;
            glVertex2d(t / tk, v);
            t += dt;
        }
        glEnd();

        glutSwapBuffers();
    }
}

```

```

void change_size(GLsizei w, GLsizei h){
    if (h == 0)
        h = 1;

    float ratio = w*1.0 / h;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    glViewport(5, 5, w, h);
    gluOrtho2D(0, 1.5, 0, 1.5);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

```

```

}

void Initialise(){
    //Цвет очистки черный
    glClearColor(0.0, 0.0, 0.0, 0.0);
}

int main(int argc, char **argv){
    //Создание окна
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(20, 20);
    glutCreateWindow("Полет ракеты");

    //Вызывание обратных функций
    glutDisplayFunc(render_scene);
    glutReshapeFunc(change_size);
    Initialise();
    //initMenu();

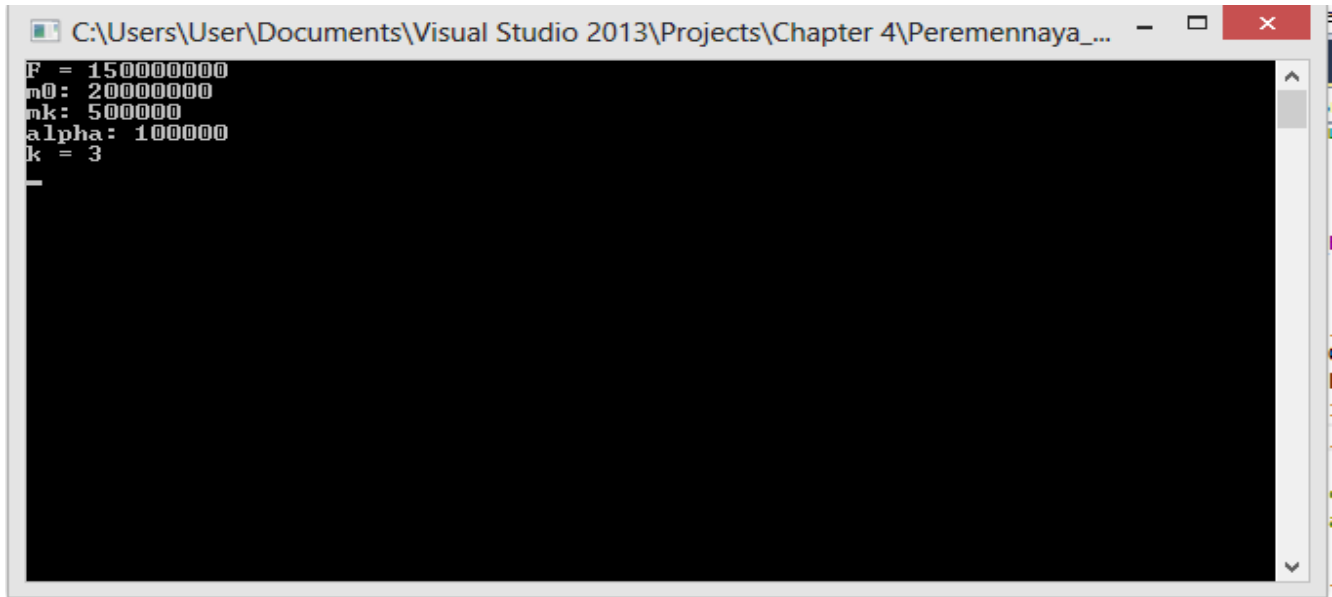
    //Запустить оболочку GLUT
    glutMainLoop();

    return 0;
}

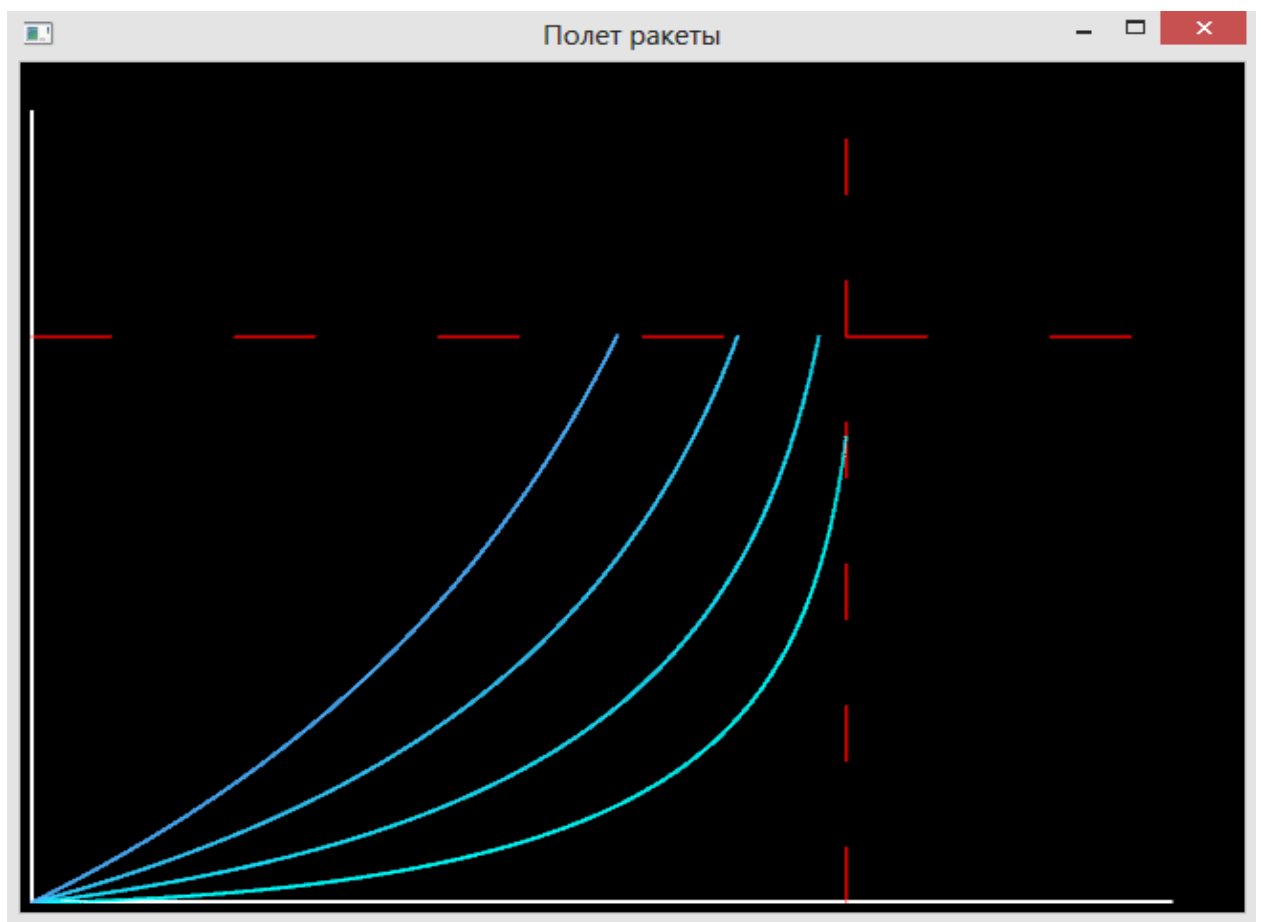
```

Результаты работы программы

A) $m_0 = 200000000 \text{ кг}$, $m_k = 500000 \text{ кг}$, $\alpha = 100000 \text{ кг/с}$, $k = 3$,
 $F = \{1500000000 \text{ Н}, 2250000000 \text{ Н}, 3375000000 \text{ Н}\}$



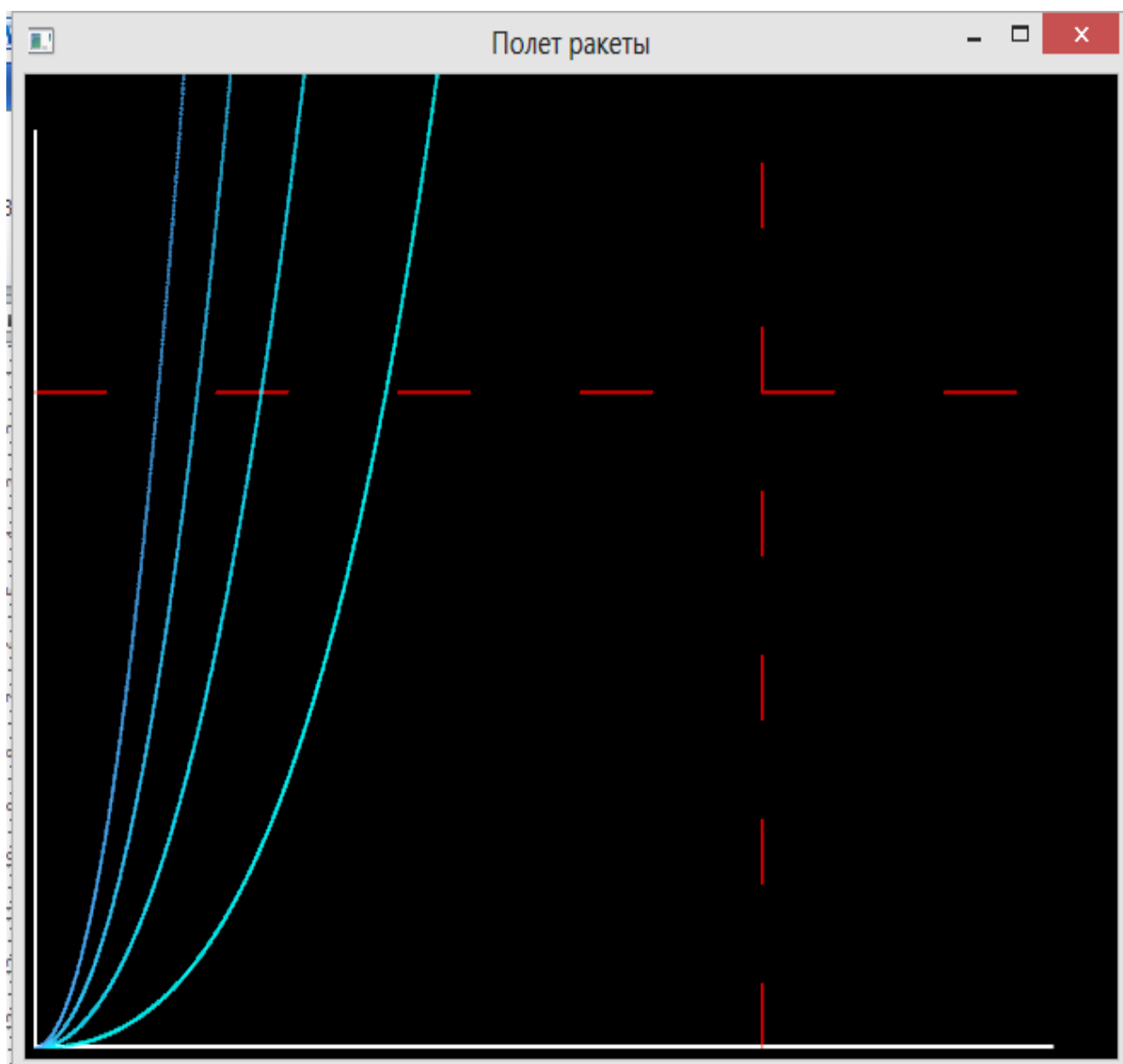
```
C:\Users\User\Documents\Visual Studio 2013\Projects\Chapter 4\Peremennaya_...  
F = 1500000000  
m0: 200000000  
mk: 500000  
alpha: 100000  
k = 3
```



(Зависимость $v(t)$ от времени t . Чем ближе график к оси Oy , тем скорость v высока)

Зависимость $H(t)$ от времени t

```
C:\Users\User\Documents\Visual Studio 2013\Projects\Chapter 4\Peremennaya_...  
F = 1500000000  
m0: 200000000  
mk: 5000000  
alpha: 100000  
k = 3
```



2.3. Математическая модель движения космических тел

Как движется Земля и другие планеты в пространстве? Что ждет комету, залетевшую из глубин космоса в Солнечную систему? Многовековая история поисков ответов на эти вопросы о движении небесных тел хорошо известна; для многих людей, внесших большой вклад в науку, именно интерес к астрономии, устройству большого мира, был первым толчком к познанию.

По закону всемирного тяготения сила притяжения, действующая между двумя телами, пропорциональна их массам и обратно пропорциональна квадрату расстояния между ними. Если поместить начало системы координат на одном из тел (размерами тел по сравнению с расстоянием между ними будем пренебрегать), математическая запись силы, действующей на второе тело, имеет вид

$$\vec{F} = -G \frac{Mm}{r^3} \vec{r} \quad (1)$$

Здесь $G = 6,67 \cdot 10^{-11} \text{ м}^3 / (\text{кг} \cdot \text{с}^2)$ - гравитационная постоянная.

Знак «минус» в формуле связан с тем, что гравитационная сила является силой притяжения, т.е. стремится уменьшить расстояние r между телами.

Далее мы ограничимся лишь изучением взаимного движения двух тел. При этом возникает вопрос: с какой позиции (в какой системе координат) изучать это движение? Ограничимся лишь простейшей ситуацией: рассмотрим движение одного из тел с точки зрения наблюдателя, находящегося на втором, например, движения планеты или кометы относительно Солнца. Разумеется, мы тем самым произвели ранжирование факторов, и наши последующие действия имеют отношение к реальности лишь в меру соблюдения определенных условий.

Уравнение, описывающее движение тела массы m в указанной системе координат, имеет вид

$$m \frac{d^2 \vec{r}}{dt^2} = -G \frac{Mm}{r^3} \vec{r} ,$$

или в проекциях на оси x, y

$$\frac{d^2 x}{dt^2} = -GM \frac{x}{\sqrt{(x^2 + y^2)^3}}, \quad \frac{d^2 y}{dt^2} = -GM \frac{y}{\sqrt{(x^2 + y^2)^3}}, \quad (2)$$

Интересующая нас орбита сильно зависит от «начальной скорости» тела m и «начального расстояния». Мы взяли эти слова в кавычки, так как при изучении движения космических тел нет столь отчетливо выделенного «начального момента», как в ранее рассмотренных ситуациях. При моделировании нам придется принять некоторое положение условно за начало, а затем изучать движение дальше. Очень часто космические тела движутся практически с постоянной скоростью по орбитам, близким к круговым. Для таких орбит легко найти элементарное соотношение между скоростью и радиусом. В этом случае сила тяготения выступает в роли центростремительной, а центростремительная сила при постоянной скорости выражается известной из начального курса физики формулой, mv^2/r . Таким образом, имеем

$$\frac{mv^2}{r} = G \frac{Mm}{r^2}$$

или

$$v = \sqrt{\frac{MG}{r}} \quad (3)$$

- искомое соотношение связи скорости и радиуса для круговой орбиты.

Период движения по такой орбите

$$T = \frac{2\pi r}{v} = 2\pi \sqrt{\frac{r^3}{MG}} .$$

Заметим, что отсюда вытекает один из законов Кеплера, приведший Ньютона к открытию закона всемирного тяготения: отношение кубов радиусов орбит любых двух планет Солнечной системы равно отношению квадратов

периодов их обращения вокруг Солнца, т.е. $\left(\frac{r_1}{r_2}\right)^3 = \left(\frac{T_1}{T_2}\right)^2$. Более точная формулировка дана ниже (так как реально орбиты планет не вполне круговые).

Если соотношение (3) нарушено, то орбита не будет круговой. Выяснить, какой она будет, можно в ходе численного моделирования. Сведем (2) к системе четырех дифференциальных уравнений первого порядка:

$$\begin{cases} \frac{dx}{dt} = v_x \\ \frac{dv_x}{dt} = -GM \frac{x}{\sqrt{(x^2 + y^2)^3}} \\ \frac{dy}{dt} = v_y \\ \frac{dv_y}{dt} = -GM \frac{y}{\sqrt{(x^2 + y^2)^3}} \end{cases} \quad (4)$$

В этой задаче особенно неудобно работать с размерными величинами, измеряемыми миллиардами километров, секунд и так далее. В качестве характерных величин обезразмеривания удобно принять

1. характерное расстояние от Земли до Солнца $\rho = 1.496 \cdot 10^{11}$ м,
2. период круговой орбиты $T = 2\pi \sqrt{\frac{\rho^3}{MG}}$, соответствующий этому расстоянию,
3. скорость движения по орбите $\bar{v} = \sqrt{\frac{MG}{\rho}}$,

то есть принять новые безразмерные переменные:

$$X = \frac{x}{\rho}, \quad Y = \frac{y}{\rho}, \quad V_x = \frac{v_x}{\bar{v}}, \quad V_y = \frac{v_y}{\bar{v}}, \quad \tau = \frac{t}{T}.$$

После обезразмеривания получаем

$$\left\{ \begin{array}{l} \frac{dX}{d\tau} = 2\pi V_x \\ \frac{dV_x}{d\tau} = -2\pi \frac{X}{\sqrt{(X^2 + Y^2)^3}} \\ \frac{dY}{d\tau} = 2\pi V_y \\ \frac{dV_y}{d\tau} = -2\pi \frac{Y}{\sqrt{(X^2 + Y^2)^3}} \end{array} \right. \quad (5)$$

Отметим, что в безразмерных переменных уравнения не содержат параметров. Отличают разные режимы движения друг от друга – начальные условия.

Можно доказать, что возможные траектории движения, описываемые уравнениями (5) – эллипс, парабола и гипербола.

Напомним **законы Кеплера**:

1. Всякая планета движется по эллиптической орбите с общим фокусом, в котором находится Солнце.
2. Каждая планета движется так, что ее радиус-вектор за одинаковые промежутки времени описывает равные площади. Это означает, что чем ближе планета к Солнцу, тем у нее больше скорость движения по орбите.
3. Отношение кубов больших полуосей орбит двух любых планет равно отношению квадратов периодов их обращения вокруг Солнца.

Уравнения (5) описывают движение не только планет, но и любых тел, попадающих в поле тяготения большой массы.

Все эти утверждения можно проверить и детально исследовать с помощью уравнений (5). При этом полезно использовать свойство консервативности данной системы – сохранение полной энергии движущегося тела. Полная энергия движущегося небесного тела m в системе двух тел имеет значение

$$E = \frac{1}{2}mv^2 - G \frac{mM}{r}.$$

Первое слагаемое – кинетическая, второе – потенциальная энергия.

В безразмерных переменных

$$\varepsilon = \frac{E}{\left(\frac{mMG}{\rho}\right)} = \frac{1}{2} \sqrt{(V_x^2 + V_y^2)} - \frac{1}{\sqrt{X^2 + Y^2}}.$$

Наличие неизменного параметра ε в ситуации, когда изменяются V_x , V_y , X , Y , позволяет контролировать процесс решения системы дифференциальных уравнений, проверять устойчивость метода, подбирать шаг интегрирования.

Рассмотрим модель инерционного движения космического тела (спутника) под действием силы всемирного тяготения в гравитационном поле, создаваемом телом с многократно большей массой (Землей). Будем интересоваться тем, какие траектории спутника возможны, какой должна быть его минимальная скорость вблизи поверхности Земли, чтобы он, двигаясь по круговой траектории, не упал на Землю (первая космическая скорость), какой должна быть минимальная начальная скорость спутника, чтобы получилась незамкнутая траектория и спутник «ушел» от Земли (вторая космическая скорость).

В основу модели мы положим закон всемирного тяготения. Будем считать, что на тело, движение которого рассматривается, действует только сила тяготения и уравнение Ньютона имеет вид:

$$m\vec{a} = -G \frac{Mm}{r^3} \vec{r} \quad (1.1)$$

Здесь m и M — масса спутника и масса притягивающего центра, G — гравитационная постоянная, r — радиус-вектор, задающий положение спутника относительно притягивающего центра, a — ускорение спутника. Заметим, что хотя закон всемирного тяготения в форме (1.1) записан для материальных точек, он имеет такую же форму для сферически-симметричных тел.

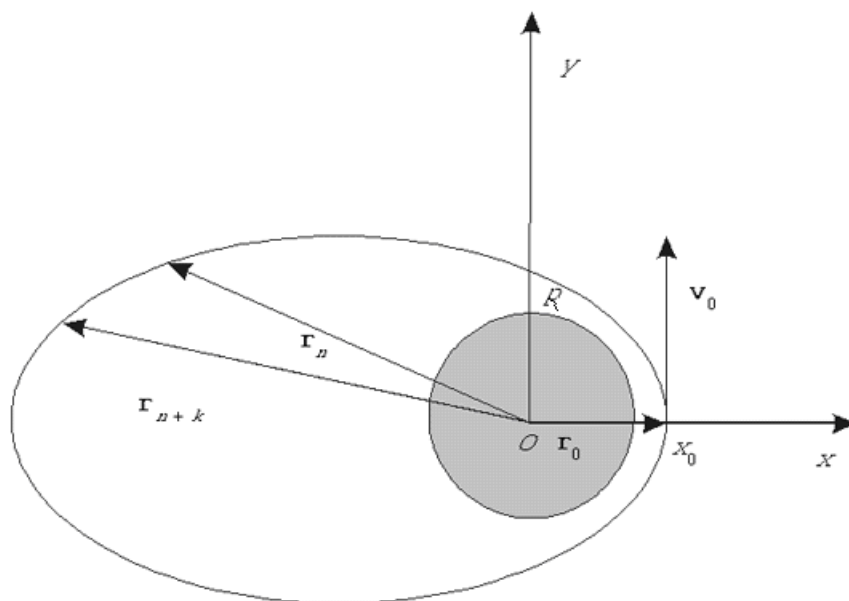


Рисунок 1

Система координат и возможная траектория спутника

Движение тела под влиянием центральной силы происходит в одной плоскости, положение которой определяется векторами r_0 и v_0 , задающими начальное положение тела и его начальную скорость. Декартову систему координат с началом в центре тяготения и начало отсчета времени выберем так, чтобы движение происходило в плоскости Oxy и в начальный момент скорость тела была перпендикулярна оси x (рис. 20).

Тогда начальные условия можно записать в виде:

$$t = 0: x = x_0, y = 0, v_x = 0, v_y = v_0 \quad (2)$$

Уравнение (1) вместе с условиями (2) полностью определяют траекторию спутника и все ее свойства.

Осуществим обезразмеривание задачи. Численный анализ задачи удобно проводить, используя в качестве единиц измерения характерные масштабы задачи. В качестве единицы длины удобно взять x_0 . Если разговор идет о спутнике Земли, то эта величина имеет порядок радиуса Земли R и равняется $R + h$, где h — высота спутника над поверхностью Земли. Всякое расстояние теперь будет задаваться числом, которое показывает, сколько раз в нем укладывается x_0 . Безразмерное x будет равняться x , измеренному в

метрах, деленному на x_0 , также измеренному в метрах. Единицу времени удобно построить, используя гравитационную постоянную и характеристики притягивающего центра. Из уравнения (1) легко видеть, что множитель GM/r^2 имеет размерность ускорения (m/c^2). Вместо расстояния r возьмем x_0 и сформируем выражение с размерностью времени (c): $(GM/x_0^3)^{-1/2}$. Его и выберем в качестве единицы времени. В качестве единицы скорости тогда естественно взять $x_0/(GM/x_0^3)^{-1/2}$, т.е. $(GM/x_0)^{1/2}$.

Измеренные в этих единицах проекции ускорения определяются следующими уравнениями (здесь и далее для безразмерных физических величин использованы те же обозначения, какие использовались для соответствующих размерных величин):

$$a_x = -\frac{x}{\sqrt{(x^2 + y^2)^3}}, \quad a_y = -\frac{y}{\sqrt{(x^2 + y^2)^3}}, \quad (3)$$

а начальные условия принимают вид:

$$t = 0: \quad x = 1, \quad y = 0, \quad v_x = 0, \quad v_y = V_0 \quad (4)$$

где $V_0 = v_0 \cdot (x_0/GM)^{1/2}$.

Отметим замечательное обстоятельство: в безразмерных переменных уравнения вообще не содержат параметров! Единственное, что отличает разные режимы движения друг от друга – начальные условия. Все физические величины измеряются теперь в относительных единицах и будут одинаковыми для всех систем “спутник — притягивающий центр”. Уменьшилось также число параметров задачи. Единственный безразмерный параметр V_0 , который остался в задаче, показывает, как соотносятся между собой кинетическая и потенциальная энергии спутника в начальный момент. Для нахождения в различные моменты времени проекций скорости спутника и его координат на временной оси выберем дискретные точки t_n , отстоящие друг от друга на малые интервалы Δt . Тогда проекции скорости в момент времени t_{n+1} будут приближенно (считаем, что ускорение на этом интервале времени не изменилось) представляться выражениями

$$V_x^{(n+1)} = V_x^{(n)} + \Delta t \cdot a_x^{(n)} \quad (5)$$

$$V_y^{(n+1)} = V_y^{(n)} + \Delta t \cdot a_y^{(n)} \quad (6)$$

а координаты в этот момент будем вычислять, как при равномерном движении (опять считая, что интервал времени Δt мал, и скорость в течение него такая, как в конце интервала):

$$x^{(n+1)} = x^{(n)} + \Delta t \cdot V_x^{(n+1)} \quad (7)$$

$$y^{(n+1)} = y^{(n)} + \Delta t \cdot V_y^{(n+1)} \quad (8)$$

В начальный момент времени проекции скорости и координаты спутника известны:

$$x^{(0)} = 1, \quad y^{(0)} = 0, \quad V_x^{(0)} = 0, \quad V_y^{(0)} = V_0$$

Система (3),(5)–(8) позволяет шаг за шагом, при малом Δt , достаточно точно вычислить траекторию спутника и все ее характеристики.

Программа построение траекторию движения космических тел вокруг земли

Код программы:

```
//Подключаем библиотеку glut.h и cmath
#include<glut.h>
#include<cmath>

//Задаем начальные значение
//x - Расстояние между спитником и центром притяжения
float x = 1.2;
//y - координата спутник
//Она поначалу равна 0
float y = 0;
//vx - Составляющие скорости по Ох
//Эта скорость в начальный момент времени равна 0
double vx = 0;
```



```

//vy - составляющее скорости по Oy
//Она в начальный момент времени равна 1.2
double vy = 1.2;
//dt - шаг
double dt = 0.00005;
//t - Общее время
double t = dt;
//PI - число Пи
const double PI = 3.1415926535;
//m,k - Промежуточные переменные
double m, k;

double velo(double v, double z, double l, double ta){
    m = z*z + l*l;
    k = sqrt(m);
    return (v - (z / (k*k*k)*ta));
}

//Вызывается при изменении размеров окна
void change_size(GLsizei w, GLsizei h){
    GLdouble aspect_ratio;
    //Предотвращает деление на 0
    if (h == 0)
        h = 1;

    //Устанавливает поле просмотра с размерами окна
    glViewport(0, 0, w, h);
    //Обновляет систему координат
    glMatrixMode(GL_PROJECTION);

```

```

glLoadIdentity();

//С помощью плоскостей отсечения (левая, правая, нижняя,
//верхняя, ближняя, дальняя) устанавливает объем
отсечения
aspect_ratio = (GLdouble)w / (GLdouble)h;
if (w <= h)
    glOrtho(-40.0, 6.0, -23.0 / aspect_ratio, 23.0 /
aspect_ratio, -1.0, 1.0);
else
    glOrtho(-40.0*aspect_ratio, 6.0*aspect_ratio, -
23.0, 23.0, -1.0, 1.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}

//Задаёт состояние визуализации
void initialise(){
    //Задаёт цвет очистки окна
    //Цвет очистки - черный
    glClearColor(0.0, 0.0, 0.0, 0.0);
}

void render_scene(){
    //Очищает окно, используя текущий цвет очистки
    glClear(GL_COLOR_BUFFER_BIT);

    //В качестве текущего цвета рисования задает красный
    //R G B

```

```

glColor3f(1.0, 0.0, 0.0);
//Рисует координатную систему:
//Рисует ось Ox
glBegin(GL_LINES);
glVertex2f(-39.0, 0.0);
glVertex2f(5.0, 0.0);
glEnd();
//Рисует ось Oy
glBegin(GL_LINES);
glVertex2f(0.0, -22.0);
glVertex2f(0.0, 22.0);
glEnd();

//Рисуем центр притяжения
//Радиус центра равна 1
//Цвет рисовки - зеленый
glColor3f(0.0, 1.0, 0.0);
double radian = 0.0;
double xx, yy;
glBegin(GL_POINTS);
while (radian <= 2 * PI){
    xx = cos(radian);
    yy = sin(radian);
    glVertex2d(xx, yy);
    radian += 0.05;
}
glEnd();

//Задаёт в качестве цвета рисовки синий

```

```

glColor3f(0.0, 0.0, 1.0);
//Рисуем траекторию первого спутника
glBegin(GL_POINTS);
while (t <= 70){
    glVertex2d(x, y);
    vx = velo(vx, x, y, dt);
    vy = velo(vy, y, x, dt);
    x += vx*dt;
    y += vy*dt;
    t += dt;
}
glEnd();

//Задает начальные данные
x = 1.25;
y = 0;
vx = 0;
vy = 1.2;
t = 0;

//Рисуем траекторию второго спутника
glColor3f(0.0, 1.0, 1.0);
glBegin(GL_POINTS);
while (t <= 200){
    glVertex2d(x, y);
    vx = velo(vx, x, y, dt);
    vy = velo(vy, y, x, dt);
    x += vx*dt;
    y += vy*dt;
}
glEnd();

```

```

        t += dt;
    }
    glEnd();

    //Рисуем траекторию 3-го спутника
    x = 1.3;
    y = 0;
    vx = 0;
    vy = 1.2;
    t = 0;
    glColor3f(.5, .6, 1.0);
    glBegin(GL_POINTS);
    while (t <= 220){
        glVertex2d(x, y);
        vx = velo(vx, x, y, dt);
        vy = velo(vy, y, x, dt);
        x += vx*dt;
        y += vy*dt;
        t += dt;
    }
    glEnd();

    //Очищает очередь текущих команд
    glFlush();
}

//Точка входа основной программы
int main(int argc, char **argv){

```

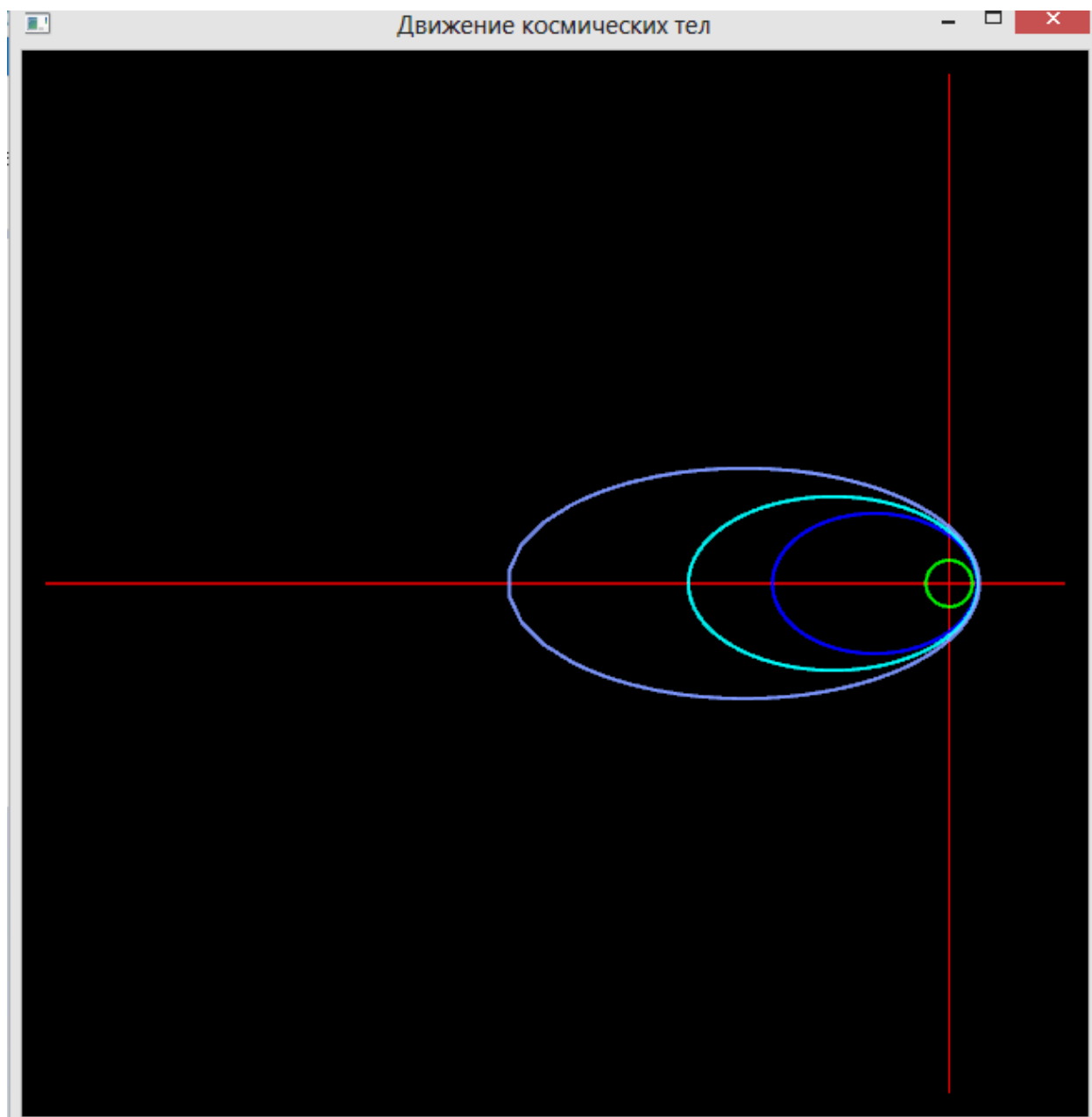
```
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
//Задает размер окна
glutInitWindowSize(640, 640);
//Определяет позиции окна на рабочем столе
glutInitWindowPosition(20, 20);
//Создает окно с названием в кавычках
glutCreateWindow("Движение небесных тел");
//Дисплейная функция
glutDisplayFunc(render_scene);
//Функция перерисовки
glutReshapeFunc(change_size);
initialise();

//Запускает оболочку GLUT
glutMainLoop();

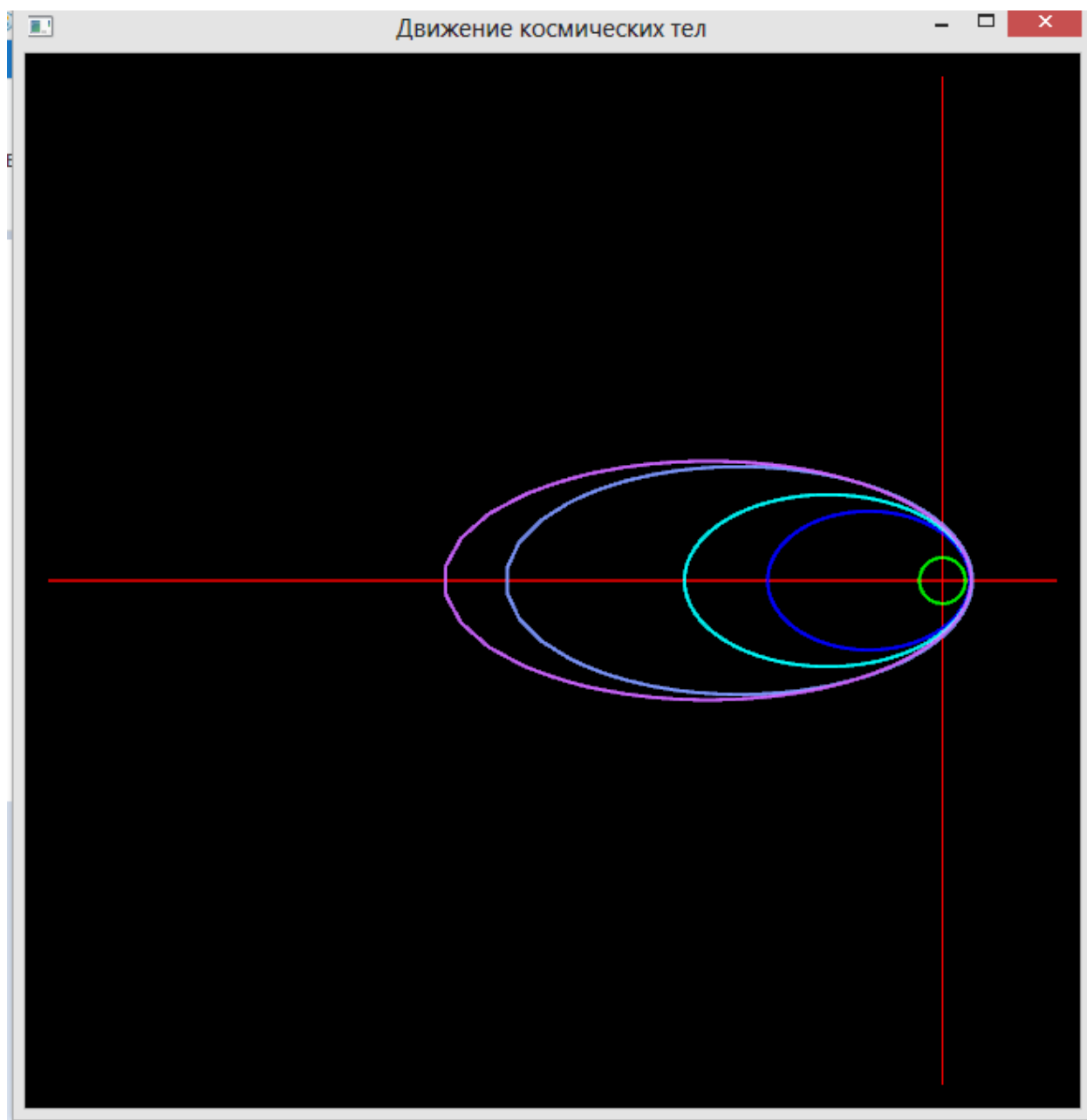
return 0;
}
```

Результаты работы программы:

А) Траектория движения трех спутников вокруг Солнца



Б) Траектория движения четырех спутников вокруг Солнца



Заключение

В работе рассмотрены задачи: моделирование движения тела, брошенного под углом к горизонту; моделирование движения космических тел; моделирование движения тела с переменной массой – взлета ракеты. Все эти задачи реализованы на языке C++ и отлажены в среде MS Visual Studio. Визуализация результатов моделирования разработана с применением графических библиотек OpenGL и GLUT.

Создан мультимедийный ролик с описанием настройки GLUT (OpenGL Utility Toolkit) – графического инструмента пользователя.

Все поставленные задачи решены.

Список литературы

1. Могилев А.В., Пак Н.И., Хённер Е.К. Информатика: Учеб. пособие для студ. высш. учеб. заведений - М.: Издательский центр «Академия», 2001.–816с.
2. Могилев А.В., Пак Н.И., Хённер Е.К. Практикум по информатике: Учеб. пособие для студ. высш. учеб. заведений - М.: Издательский центр «Академия», 2001.–608с.
3. Горстко А. Б. Познакомьтесь с математическим моделированием. - М.: Знание, 1991–160с.
4. Тимофеева Л.А. Численные методы решения задач на ЭВМ// Информатика и образование. – 2003. – №12 –с.45-50
5. Мясникова О.К. Моделирование и формализация в курсе информатики// Информатика и образование. – 2003. – № 9, 10, 11, 12.
6. Тарасевич Ю.Ю Математическое и компьютерное моделирование. Вводный курс: Учебное пособие. – М.: Едиториал УРСС, 2003. –144с.
7. Бордовский Г.А. Физические основы математического моделирования. М: Издательский центр «Академия», 2005 – 321с
8. OpenGL. Суперкнига, 3-е издание. : Пер. с англ – М. : Издательский дом “Вильямс”, 2006 – 1039с
9. Тихомиров Ю. Программирование трехмерной графики – СПб BHV – Санкт-Петербург, 1998 – 256с
10. OpenGL superBible: comprehensive tutorial and reference. – Sixth edition / Graham Sellers, Richard S. Wright, Jr., Nicholas Haemel - Addison Wesley, 2013 – 853pp
11. OpenGL programming guide: the official guide to learning OpenGL / Dave Shreiner, Graham Sellers, John, Kessenich, Bill Licea-Kane; the Kronos OpenGL ARB Working Group. – Eighth edition, 2013 – 986pp

12. Professional Visual Studio 2013 / Bruce Johnson - John Wiley & Sons, Inc, 2014 – 1102pp
13. Информационные технологии в науке и образовании / Е. Л. Федотова, А. А. Федотов: учеб. пособие. – М. : ИД «ФОРУМ» : ИНФРА – М, 2011. – 336с.: ил. – (Высшее образование)
14. Компьютерное моделирование. Лабораторный практикум / А. Л. Королёв. – М.: БИНОМ. Лаборатория знаний, 2012. – 296с.: ил. – (Педагогическое образование).

Интернет ресурсы

1. <http://www.cse.chalmers.se/~uffe/glut-3.spec.pdf>
2. <https://msdn.microsoft.com/ru-ru/library/60k1461a.aspx>
3. <http://www.cplusplus.com/>
4. <https://www.opengl.org/>
5. <http://en.wikipedia.org/wiki/>
6. <http://www.opengl.org.ru/>