Основы Web-технологий

КУРСОВАЯ РАБОТА

ВИЗУАЛИЗАЦИЯ ДВИЖЕНИЯ ТЕЛА
В ГРАВИТАЦИОННОМ ПОЛЕ НЕСКОЛЬКИХ
НЕПОДВИЖНЫХ ТЕЛ

Выполнила:

Шишкова С.С. 204 гр.

Преподаватель:

Алексеев А.А.

Москва, 2016

СОДЕРЖАНИЕ

Введение

1.1. Цель работы

Очень важным фактором, способным положительно повлиять на процесс обучения, является наглядность. Именно её обычно так не хватает при изучении технических и естественных наук.

Целью данной работы является визуализация движения точечного или сферически симметричного тела в гравитационном поле таких же неподвижных тел.

В работе были поставлены две задачи, а именно:

1) Расчёт траектории движения тела (backend)
2) Создание интерфейса (frontend)

1.2. Элемент Canvas

Canvas — элемент HTML5, предназначенный для создания растрового двухмерного изображения при помощи программ на языке JavaScript. Он представляет собой ключевой компонент для ряда графически продвинутых приложений, таких, как игры, динамические графики.

При использовании Canvas с сервера загружается не картинка, а набор точек (или алгоритм прорисовки), по которым браузер прорисовывает картинку. Благодаря этому свойству элемент Canvas позволяет иллюстрировать траекторию движения тела без обновления Web-страницы.

Решение задачи

2.1. Вычислительная часть

Моделирование происходит на основе Ньютоновских уравнений движения:

$$\frac{d\vec{v}}{dt} = \vec{a} \quad (1)$$

$$\frac{d\vec{r}}{dt} = \vec{v} \quad (2)$$

где a и v – ускорение и скорость соответственно. Ускорение считается ка

$$\vec{a}_n = \sum_{k=0}^{N} m_k \frac{(\overrightarrow{r_n} - \overrightarrow{r_k})}{|\vec{r}_{n-}\overrightarrow{r_k}|^3}, \quad \text{n и k – номера тел, N – кол-во тел.} \quad (3)$$

Воспользуемся вторым законом Ньютона, который связывает ускорение движущегося тела и силы, действующие на него:

$\vec{F} = m\vec{a}$ (4)

Здесь $\vec{F}$ - сумма всех сил, действующих на тело, $\vec{a}$ - полное ускорение. Это же уравнение можно записать в проекциях на оси x, y:

$F_x = ma_x$ – сила, действующая в горизонтальном направлении. (5)

$F_y = ma_y$ – сила, действующая в вертикальном направлении. (6)

```
vector f(vector u, float t)  //function describing the motion of the flying body
{
        vector res;                    //derivate
        float r1, r2, r3, r4, r5;
        res.x = u.vx;                          //how to get vx from x
        res.y = u.vy;                          //how to get vy from y
        r1 = sqrt((u.x-x1)*(u.x-x1)+(u.y-y_1)*(u.y-y_1));  // distanse to body number 1
        r2 = sqrt((u.x-x2)*(u.x-x2)+(u.y-y2)*(u.y-y2));    // ... 2
        r3 = sqrt((u.x-x3)*(u.x-x3)+(u.y-y3)*(u.y-y3));    // ... 3
        r4 = sqrt((u.x-x4)*(u.x-x4)+(u.y-y4)*(u.y-y4));    // ... 4
        r5 = sqrt((u.x-x5)*(u.x-x5)+(u.y-y5)*(u.y-y5));    // ... 5

        //how to get ax
        res.vx = -G * (m1/pow(r1,3+eps)*(u.x-x1) + m2/pow(r2,3+eps)*(u.x-x2) + m3/pow(r3,3+eps)
        *(u.x-x3) + m4/pow(r4,3+eps)*(u.x-x4) + m5/pow(r5,3+eps)*(u.x- x5));
        //and ay
        res.vy = -G * (m1/pow(r1,3+eps)*(u.y-y 1) + m2/pow(r2,3+eps)*(u.y-y2) + m3/pow(r3,3+eps)
        *(u.y-y3) + m4/pow(r4,3+eps)*(u.y-y4) + m5/pow(r5,3+eps)*(u.y- y5));
        return(res);
}
```

Рис. 1 – Уравнения движения тела (программа на C++)

Получим систему уравнений, описывающую движение нашего подвижного объекта.

Для того, чтобы решение этой системы существовало и было единственным, необходимо и достаточно, чтобы были заданы значения неизвестных в начальный момент времени t=0. Их вводит на Web-странице сам пользователь.

При решении уравнений движения в данной работе я использовала метод Рунге-Кутты 4 порядка – явный одношаговый численный метод, имеющий четвёртый порядок точности.

В одношаговом методе каждое последующее значение зависит только от предыдущего; в многошаговом – от нескольких предыдущих значений. Преимущество одношаговых методов перед многошаговыми состоит в том, что при нахождении каждого последующего значения не требуется решать систему уравнений, учитывающую зависимость от каждого предыдущего значения.

Введём четырехмерный вектор U(x, y, v_x, v_y). Координатами этого вектора являются координаты тела и проекции его скорости. Нашу систему уравнений можно записать в виде:

$U' = F(U, t)$, где $F(U,t) = (v_x, v_y, a_x, a_y)$ (7)

Суть метода состоит в том, чтобы разбить интервал времени, в течение которого нам надо узнать координаты и скорости, на некоторое большое количество отрезков времени dt.

Пусть мы знаем значение вектора U в некоторый момент времени t и хотим узнать его в момент времени t+dt.

Методом Эйлера называется метод, в котором

$$U(t+dt) = U(t) + F(U,t)dt\ , \tag{8}$$

т.е.

$$x(t+dt) = x(t) + v_x dt \tag{9}$$
$$y(t+dt) = y(t) + v_y dt \tag{10}$$
$$v_x(t+dt) = v_x(t) + a_x dt \tag{11}$$
$$v_y(t+dt) = v_y(t) + a_y dt \tag{12}$$

Методом Рунге-Кутта 4-го порядка называется метод, в котором

$$U(t+dt) = U(t) + (k_1 + 2k_2 + 2k_3 + k_4)/6\ , \tag{13}$$

где

$$k_1 = F(U,t)dt \tag{14}$$
$$k_2 = F(U+k_1/2,\ t+dt/2)dt \tag{15}$$
$$k_3 = F(U+k_2/2,\ t+dt/2)dt \tag{16}$$
$$k_4 = F(U+k_3,\ t+dt)dt \tag{17}$$

```
class vector                                        //just a class
{
        public:
        float x, y, vx, vy;
        vector()
        {
                x = 0;
                y = 0;
                vx = 0;
                vy = 0;
        }
        vector(float ax, float ay, float avx, float avy)
        {
                x = ax;
                y = ay;
                vx = avx;
                vy = avy;
        }
        void out()
        {
                printf("x=%f\t", x);
                printf("y=%f\t", y);
                printf("vx=%f\t", vx);
                printf("vy=%f\t", vy);
        }
};
```

Рис. 2 – Фрагмент кода на C++

В данном участке кода я ввожу класс под названием vector. У него есть 4 компоненты: x, y, vx, vy. Далее я ввела операции сложения и вычитания таких векторов (покомпонентное сложение или вычитание) а также операции

умножения вектора на число и числа на вектор (которые совпадают с точностью до перестановки множителей).

```
vector operator +(vector a, vector b)           //overwritten operator
{
	vector c;
	c.x = a.x + b.x;
	c.y = a.y + b.y;
	c.vx = a.vx + b.vx;
	c.vy = a.vy + b.vy;
	return(c);
}
vector operator -(vector a, vector b)           //overwritten operator
{
	vector c;
	c.x = a.x - b.x;
	c.y = a.y - b.y;
	c.vx = a.vx - b.vx;
	c.vy = a.vy - b.vy;
	return(c);
}
vector operator *(float a, vector b)            //overwritten operator
{
	vector c;
	c.x = a * b.x;
	c.y = a * b.y;
	c.vx = a * b.vx;
	c.vy = a * b.vy;
	return(c);
}
vector operator *(vector b, float a)            //overwritten operator
{
	vector c;
	c.x = a * b.x;
	c.y = a * b.y;
	c.vx = a * b.vx;
	c.vy = a * b.vy;
	return(c);
}
```

Рис. 3 – Программа на C++. Ввод операторов

Напишем функцию F(U, t) для нашей задачи (см. рис. 1) и главную часть программы:

```
// MAIN :

int main ( int argc, char *argv[] )
{
        if( argc == 21 )
        {
                x_0 = strtof (argv[1], NULL);
                y_0 = strtof (argv[2], NULL);
                vx0 = strtof (argv[3], NULL);
                vy0 = strtof (argv[4], NULL);
                m = strtof (argv[5], NULL);
                x1 = strtof (argv[6], NULL);
                y_1 = strtof (argv[7], NULL);
                m1 = strtof (argv[8], NULL);
                x2 = strtof (argv[9], NULL);
                y2 = strtof (argv[10], NULL);
                m2 = strtof (argv[11], NULL);
                x3 = strtof (argv[12], NULL);
                y3 = strtof (argv[13], NULL);
                m3 = strtof (argv[14], NULL);
                x4 = strtof (argv[15], NULL);
                y4 = strtof (argv[16], NULL);
                m4 = strtof (argv[17], NULL);
                x5 = strtof (argv[18], NULL);
                y5 = strtof (argv[19], NULL);
                m5 = strtof (argv[20], NULL);
        }
        else {return 0;}

        vector k1, k2, k3, k4;
        float dt = 0.02, t=0;
        float r1, r2, r3, r4, r5;
        vector u(x_0, y_0, vx0, vy0);
        vector k1, k2, k3, k4;
        float dt = 0.02, t=0;
        float r1, r2, r3, r4, r5;
        vector u(x_0, y_0, vx0, vy0);

        id = fopen("array.txt", "w");
        if( id != NULL ) {
                char line[255];
                sprintf(line, "%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f",
         m, m1, m2, m3, m4, m5, x_0, y_0, x1, y_1, x2, y2, x3, y3, x4, y4, x5, y5);
                fputs(line, id);
                while (t <= 100)
                {
                        k1 = f(u, t)*dt;
                        k2 = f(u + 0.5*k1, t+dt/2)*dt;
                        k3 = f(u + 0.5*k2, t+dt/2)*dt;
                        k4 = f(u + k3, t+dt)*dt;
                        u = u + 1.0/6.0 * (k1 + 2*k2 + 2*k3 + k4);
                        t += dt;

                        sprintf(line,"%f %f %f %f %f ", t, u.x, u.y, u.vx, u.vy);
                        fputs(line, id);

                        printf ("t=%f x=%f y=%f vx=%f vy=%f\n", t, u.x, u.y, u.vx, u.vy);
                }
                fclose(id);
        }
        else {
                printf( "Can't create file!\n" );
        }

}
```

Рис. 4 – Главная часть программы на C++

2.2. Интерфейсная часть

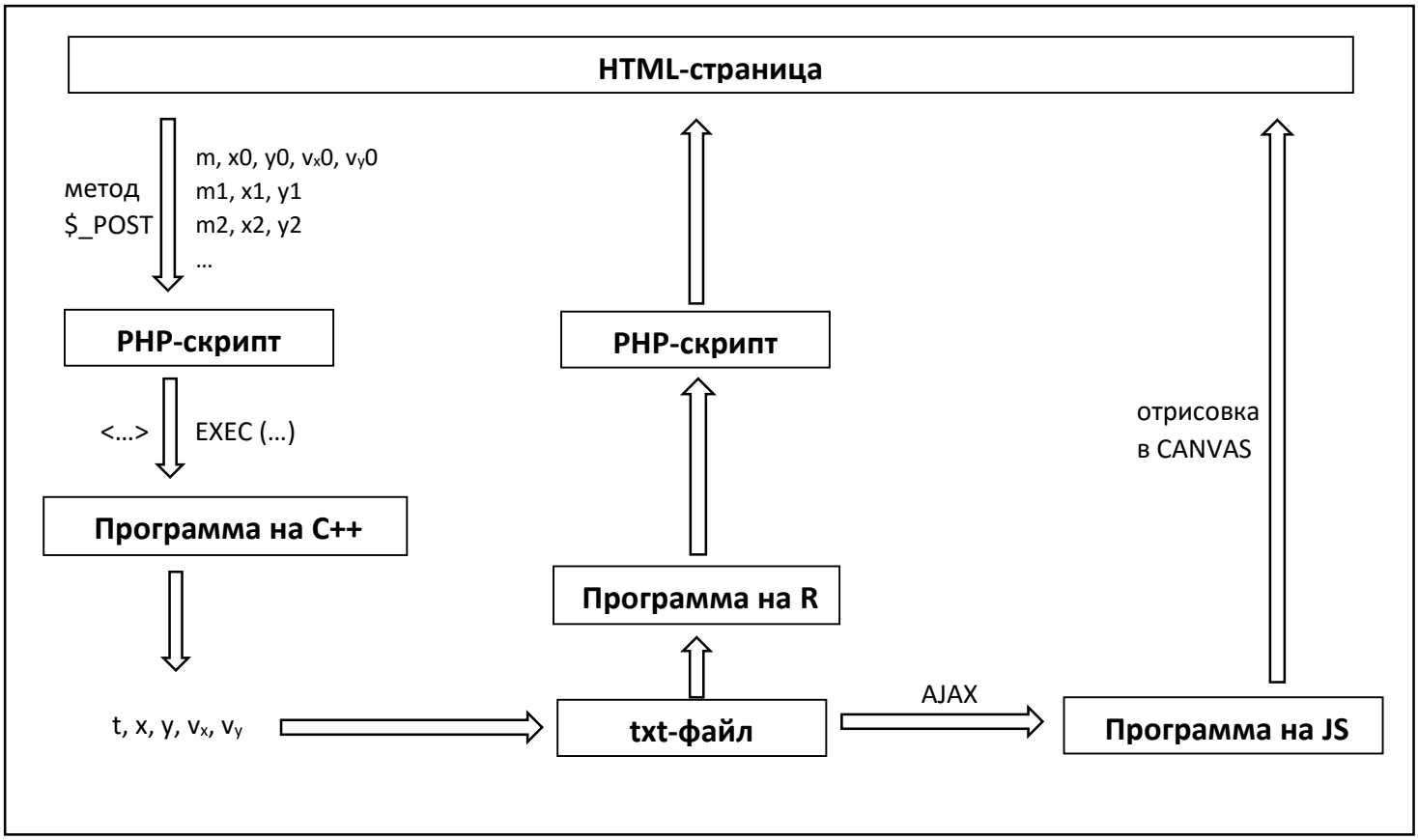

Рис. 5 – Блок-схема передачи данных по программам

При запуске программы *stranica.php* открывается Web-страница с формой для задания параметров системы. Для каждого стационарного тела задаётся его положение и масса, для движущегося объекта – координаты, масса и скорость.

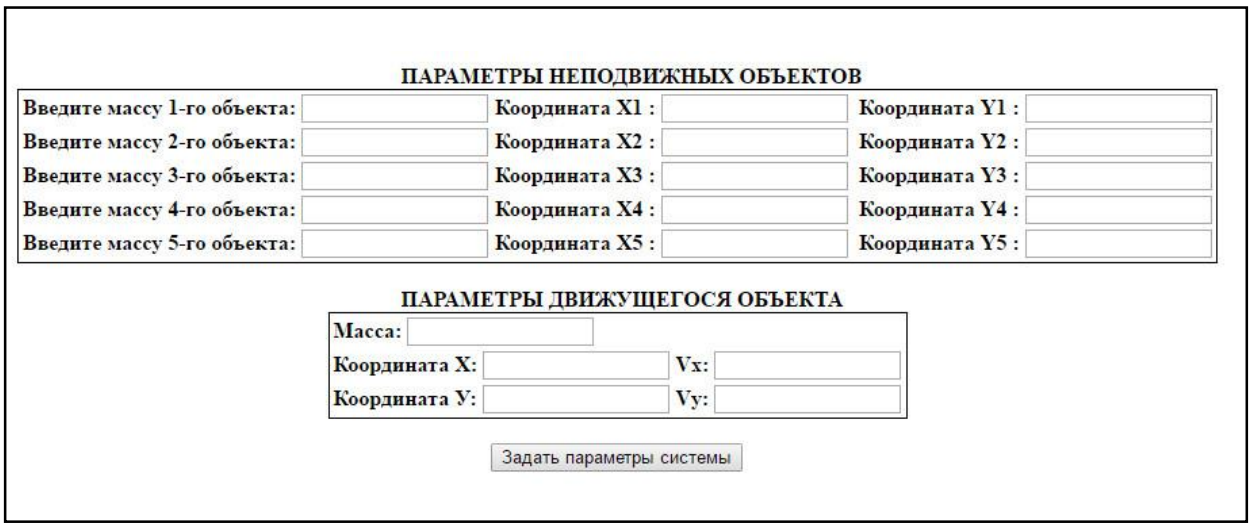

Рис. 6 – Фрагмент Web-страницы с формой для ввода данных

Используя метод POST, я передаю данные в программу на php, из которой функцией exec (argc, argv) запускается программа на C++.

```
$x1=$_POST["x1"];
$y_1=$_POST["y1"];
$m1=$_POST["m1"];

$x2=$_POST["x2"];
$y2=$_POST["y2"];
$m2=$_POST["m2"];

$x3=$_POST["x3"];
$y3=$_POST["y3"];
$m3=$_POST["m3"];

$x4=$_POST["x4"];
$y4=$_POST["y4"];
$m4=$_POST["m4"];

$x5=$_POST["x5"];
$y5=$_POST["y5"];
$m5=$_POST["m5"];

exec ("../BE/a.out ".$x_0." ".$y_0." ".$vx0." ".$vy0." ".$m." ".$x1." ".$y_1." ".$m1."
".$x2." ".$y2." ".$m2." ".$x3." ".$y3." ".$m3." ".$x4." ".$y4." ".$m4." ".$x5." ".$y5."
".$m5);
```

Рис. 7 – Передача данных методом POST и функцией exec (…)

Траектория движения тела рассчитывается численным методом Рунге-Кутты. Полученные координаты движущегося объекта записываются программой *RUNGE.cpp* в текстовый файл *array.txt*. При запуске программы на php данные передаются в код методом AJAX в виде массива, разбиваются на элементы и отрисовываются на странице с помощью элемента Canvas.

2.3. Результат

В ходе проведённой работы был реализован метод Рунге-Кутты решения уравнений движения тела в гравитационном поле других тел. С помощью этого метода определялась траектория движения и скорость тела в каждый момент времени.

Траектория полёта объекта отображалась на экране при помощи графического элемента Canvas.

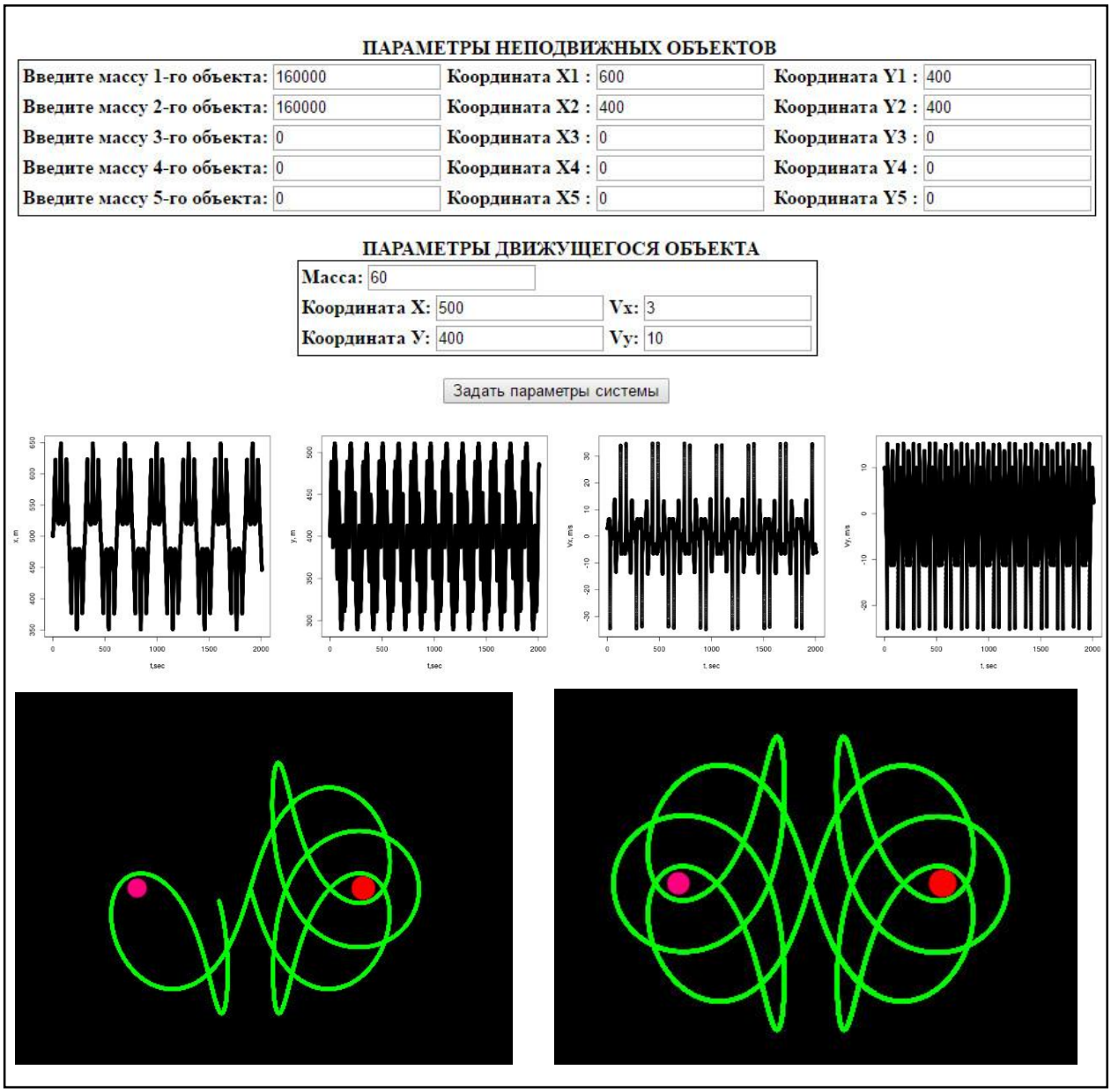

Рис. 8 – Результат работы программы

2.4. Перспективы

Перспективы этой работы могут быть самыми различными. Эволюцией данного проекта может быть создание модели Солнечной системы. Если заменить гравитационное взаимодействие на, например, кулоновское, можно продемонстрировать взаимодействие точечных заряженных объектов.

В моей работе тело движется в поле неподвижных объектов. Если учесть, что все объекты могут быть подвижными, то можно провести визуализацию гравитационной задачи N тел.