Лабораторная работа № 7
Тема: Визуализация физического явления или процесса. Анимация.
Краткие теоретические сведения

Анимация (лат. Animare - оживить) - вид искусства, произведения которого создаются путём покадровой съёмки отдельных рисунков или сцен. Помимо термина «анимация» широко употребляется также и термин «мультипликация» (лат. multiplicatio — умножение, размножение).

Кадры - это рисованные или сфотографированные изображения последовательных фаз движения объектов или их частей.

При просмотре последовательности кадров возникает иллюзия оживления изображенных на них статичных персонажей.

Для создания эффекта плавного изменения их положения и формы, исходя из особенностей человеческого восприятия, частота смены кадров должна быть не менее 12-16 кадров в секунду.

В кино используется частота 24, в телевидении 25 или 30 кадров в секунду.

Физическая анимация - это область интереса в компьютерной графике, связанная с моделированием физически правдоподобного поведения в интерактивном режиме. Достижения в области анимации, основанной на физике, часто мотивируются необходимостью включения сложных, вдохновляемых физическими упражнениями моделей поведения в видеоигры, интерактивные симуляции и фильмы.

Хотя существуют методы автономного моделирования для решения большинства проблем, изучаемых в физической анимации, эти методы предназначены для приложений, требующих физической точности и медленных, детальных вычислений.

В отличие от методов, распространенных в автономном моделировании, методы физически обоснованной анимации ориентированы на физическую достоверность, числовую стабильность и визуальную привлекательность, а не на физическую точность.

Физическая анимация часто ограничивается приближением к физическому поведению из-за жестких временных ограничений, накладываемых интерактивными приложениями.

Целевая частота кадров для интерактивных приложений, таких как игры и симуляторы, часто составляет 25-60 Гц , при этом для физического моделирования остается лишь небольшая часть времени, отведенного на отдельный кадр. Упрощенные модели физического поведения обычно предпочтительнее, если они более эффективны, легче ускоряются (с помощью предварительных вычислений, умных структур данных или SIMD / GPGPU) или удовлетворяют желаемым математическим свойствам (таким как безусловная стабильность или сохранение объема, когда мягкое тело подвергается деформации).

Физическая анимация является обычным явлением в играх и симуляторах, где пользователи ожидают взаимодействия с окружающей средой. Физические движки, такие как Havok , PhysX и Bullet, существуют как отдельно разработанные продукты, которые подлежат лицензированию и включаются в игры.

В таких играх, как Angry Birds или World of Goo , физическая анимация сама по себе является основной игровой механикой, и ожидается, что игроки будут

взаимодействовать с физически смоделированными системами или создавать их для достижения целей.

Аспекты игр-головоломок с физикой существуют во многих играх, принадлежащих к другим жанрам, но имеющих физическое моделирование. Разрешение физического взаимодействия с окружающей средой с помощью физически обоснованной анимации способствует нелинейным решениям головоломок игроками и иногда может приводить к решениям проблем, представленных в играх, которые не были намеренно включены разработчиками уровней.

В симуляторах, не связанных с развлечениями, например, в военных симуляторах, также используется физическая анимация для изображения реалистичных ситуаций и сохранения погружения пользователей. Многие методы в физической анимации разработаны с учетом реализаций GPGPU или могут быть расширены, чтобы получить выгоду от графического оборудования, которое можно использовать для создания достаточно быстрых физических симуляций для игр.

Моделирование жесткого тела

Упрощенная физика твердого тела относительно дешева и проста в реализации, поэтому она появилась в интерактивных играх и симуляторах раньше, чем большинство других методов.

Предполагается, что твердые тела не подвергаются деформации во время моделирования, поэтому движение твердого тела между временными шагами можно описать как перемещение и вращение, традиционно с использованием аффинных преобразований, хранящихся в виде матриц 4x4.

В качестве альтернативы, кватернионы могут использоваться для хранения поворотов, а векторы могут использоваться для хранения объектов, смещенных от начала координат.

Наиболее затратными в вычислительном отношении аспектами динамики твердого тела являются обнаружение столкновений , корректировка взаимопроникновения тел и окружающей среды и управление контактом в состоянии покоя.

Жесткие тела обычно моделируются итеративно с обратным отслеживанием для исправления ошибки с использованием меньших временных шагов. Постоянный контакт между несколькими твердыми телами (как в случае, когда твердые тела падают в сваи или штабелируются) может быть особенно трудным для эффективного управления и может потребовать сложных графиков контакта и распространения ударной волны для разрешения с использованием импульсных методов.

При моделировании большого количества твердых тел упрощенная геометрия или выпуклые оболочки часто используются для представления их границ с целью обнаружения столкновений и реагирования (поскольку это обычно является узким местом при моделировании).

Для моделирования физического процесса нам понадобится:

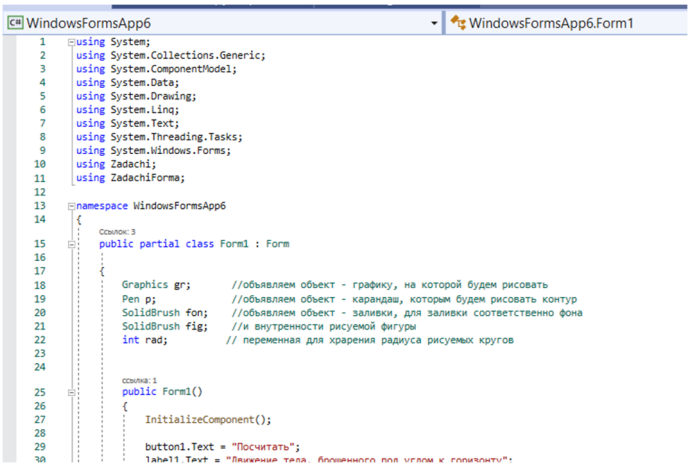
5. Графическая модель тела – в нашем случае – круг (окружность)
6. Физический закон движения модели тела – функция (x(t))

Ход работы:

В данном уроке используется среда программирования Microsoft Visual Studio 2019. Алгоритм работы аналогичен и для других сред программирования.

Перед тем, как приступить к данному уроку, следует выполнить Лабораторную работу № 5 и № 6

1. **Открываем проект с лабораторной работой № 5**
2. **Добавим к глобальным переменным необходимые нам новые переменные:**




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Zadachi;
using ZadachiForma;

namespace WindowsFormsApp6
{
    Ссылок: 3
    public partial class Form1 : Form
    {
        Graphics gr;       //объявляем объект - графику, на которой будем рисовать
        Pen p;             //объявляем объект - карандаш, которым будем рисовать контур
        SolidBrush fon;    //объявляем объект - заливки, для заливки соответственно фона
        SolidBrush fig;    //и внутренности рисуемой фигуры
        int rad;          // переменная для хранения радиуса рисуемых кругов


        ссылка: 1
        public Form1()
        {
            InitializeComponent();

            button1.Text = "Посчитать";
            label11.Text = "Движение тела, брошенного под углом к горизонту";
```

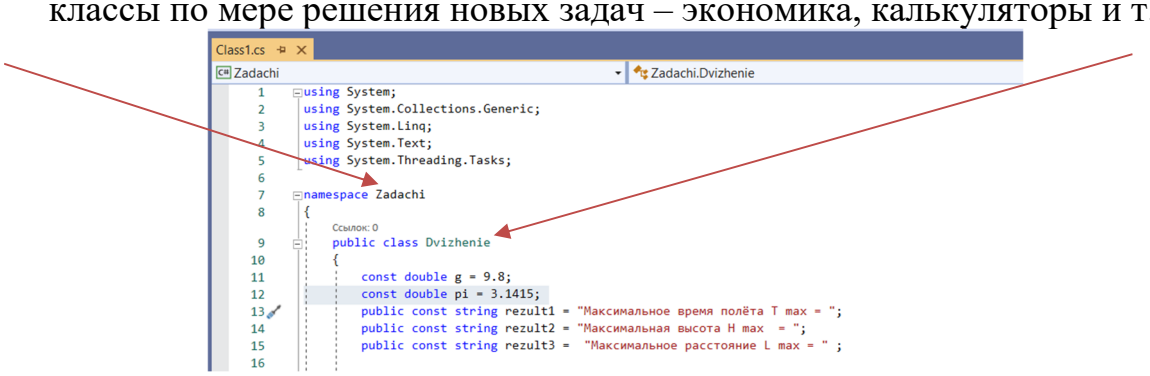
3. Перенесем метод рисования фигуры (простого примитива окружность) в наш проект из ЛР №6:

```
Ссылок: 3
public void DrawFigura(int n)
{
    Graphics gr = pictureBox1.CreateGraphics();  //инициализируем объект типа графики
                                                 // привязав  к PictureBox


    Pen p = new Pen(Color.Red);           // задали цвет для карандаша
    SolidBrush fon = new SolidBrush(Color.Beige); //    и для заливки
    int x = 50;
    int y = 50;

    switch (n)
    {
```

4. Модернизируем этот метод для рисования фигуры с использованием внешних координат:

```
ссылка: 1
void DrawCircle(int x, int y)
{
    int xc, yc;
    xc = x - rad;
    yc = y - rad;

    gr.FillEllipse(fig, xc, yc, rad, rad);

    gr.DrawEllipse(p, xc, yc, rad, rad);

}
```

5. Создаем библиотеку классов для решения нескольких видов задач (если у вас уже готовая библиотека, просто подключайте ее в проект), создаем класс задачи физического моделирования (в дальнейшем будем добавлять новые классы по мере решения новых задач – экономика, калькуляторы и т.д):



6. Теперь нам необходимо создать несколько методов для переноса физических формул в методы класса Dvizhenie:

- **вычисление максимального времени полета тела:**

```
Ссылок: 0
public static double vremyaT(double v, double a, double dt)
{

    double alpha = a * pi / 180;

    double T = Math.Round(2 * v * Math.Sin(alpha) / g, 2);


    return T;

}
```

- **вычисление максимальной высоты полета тела:**

```
Ссылок: 0
public static double vysotaH(double v, double a, double dt)
{
    double alpha = a * pi / 180;
    double H = Math.Round(v * v * Math.Sin(alpha) * Math.Sin(alpha) / (2 * g), 2);


    return H;

}
```

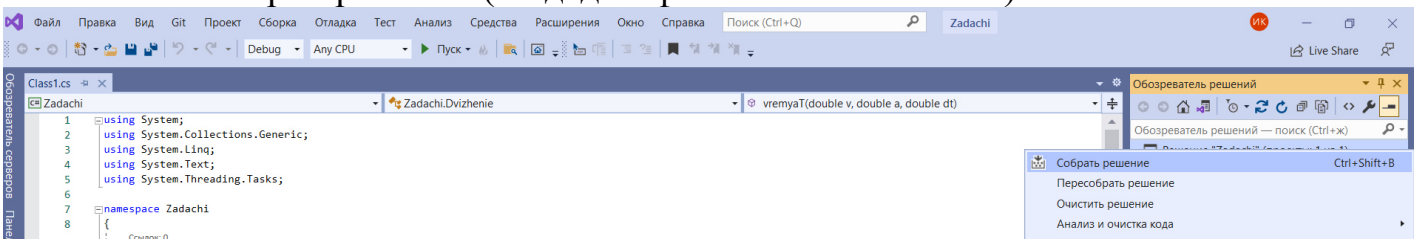
- **вычисление максимальной длины полета тела:**

```
Ссылок: 0
public static double dlinaL(double v, double a, double dt)
{
    double alpha = a * pi / 180;


    double L = Math.Round(v * v * Math.Sin(2 * alpha) / g, 2);


    return L;

}
```

- **вычисление координат X и Y :**

```
Ссылок: 0
public static double KoordX (double v, double a, double i)
{

    double X = v * i * Math.Cos(a * pi / 180);

    return X;
}
Ссылок: 0
public static double KoordY(double v, double a, double i)
{

        double Y = v * i * Math.Sin(a * pi / 180) - g * i * i / 2;

    return Y;
}
```

7. Соберем решение (создадим файл библиотеки *.dll):

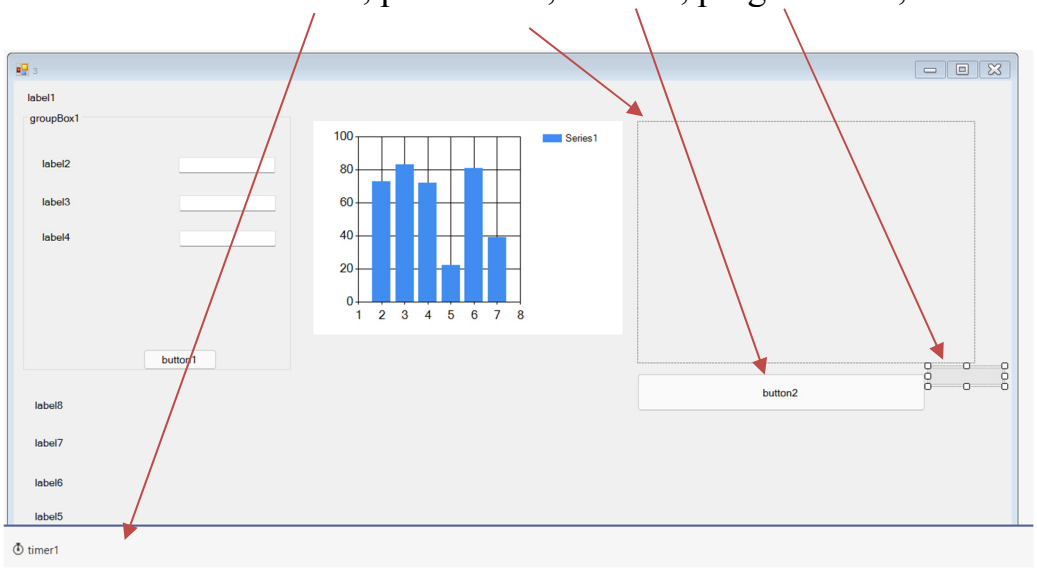


8. Подключим эту библиотеку в наш проект и заменим прямые формулы на вызовы методов нашей новой библиотеки:

```
private void button1_Click(object sender, EventArgs e)
{

    double v = Convert.ToInt32(textBox1.Text);
    double dt = Convert.ToInt32(textBox3.Text);
    double a = Convert.ToInt32(textBox2.Text);

    double T = Dvizhenie.vremyaT(v, a, dt);

    double L = Dvizhenie.dlinaL(v, a, dt);
    double H =   Dvizhenie.vysotaH (v, a, dt);

    label5.Text = Dvizhenie.rezult1 + Convert.ToString(T);
    label6.Text = Dvizhenie.rezult2 + Convert.ToString(H);

    label7.Text = Dvizhenie.rezult3 + Convert.ToString(L);

    chart1.Series[0].ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Spline;
    this.chart1.Series[0].Points.Clear();

    double i = 0;
    while (i <= T + dt)
    {
        double x = Zadachi.Dvizhenie.KoordX(v, a, i);
        double y = Zadachi.Dvizhenie.KoordY(v, a, i);

        this.chart1.Series[0].Points.AddXY(x, y);

        i = i + dt;
    }

}
```

9. Теперь попробуем создать анимацию, при помощи PictureBox и Timer, которая запустится после нажатия на кнопку (button2).

Добавим элементы timer1, pictureBox, button2, progressBar1, и на нашу форму:



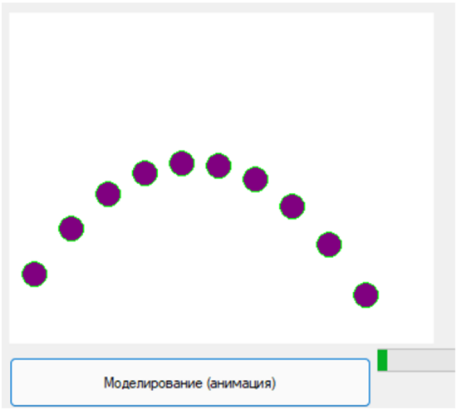
Следовательно для этого будем обрабатывать событие **нажатия на кнопку**

```
ссылка: 1
private void button2_Click(object sender, EventArgs e)
{
    progressBar1.Value = 0;
    double v = Convert.ToInt32(textBox1.Text);
    double a = Convert.ToInt32(textBox2.Text);
    double dt = Convert.ToInt32(textBox3.Text);

    double T = Dvizhenie.vremyaT(v, a, dt);


    gr = pictureBox1.CreateGraphics();  //инициализируем объект типа графики
                                        // привязав  к PictureBox
    gr.Clear(Color.White);
    p = new Pen(Color.Lime);           // задали цвет для карандаша
    fon = new SolidBrush(Color.AntiqueWhite  ); // и для заливки
    fig = new SolidBrush(Color.Purple);


     timer1.Enabled = true;  //включим в работу наш таймер,
                             // то есть теперь будет происходить событие Tick и его будет обрабатывать функция On_Tick (по умолчанию)

}
```

и событие "срабатывания" таймера:

```
ссылка: 1
private void timer1_Tick(object sender, EventArgs e)
{
    gr.Clear(Color.AntiqueWhite);
    progressBar1.Value++;

    double v = Convert.ToInt32(textBox1.Text);
    double a = Convert.ToInt32(textBox2.Text);
    double dt = Convert.ToInt32(textBox3.Text);
    double T = Dvizhenie.vremyaT(v, a, dt);

    if (progressBar1.Value >= T)
    {
        timer1.Stop();
    }


        double t = progressBar1.Value;

       double x = Zadachi.Dvizhenie.KoordX(v, a, t);
       double y = pictureBox1.Height - Zadachi.Dvizhenie.KoordY(v, a, t);


        DrawCircle(Convert.ToInt32(x), Convert.ToInt32(y));


    }
}
```

10. **После этого, можно запустить программу** и после проведенных расчетов по кнопке button1(«Рассчитать») и нажатия на кнопкуbutton2 («Моделирование») увидите простую анимацию – перемещение брошенного тела по траектории, рассчитанной в математической модели:



Траекторию полета тела вы сможете увидеть, если отключите обновление экрана на каждом кадре :

```
private void timer1_Tick(object sender, EventArgs e)
{
    // gr.Clear(Color.AntiqueWhite);
```

11. **Для того, чтобы анимация соответствовала требованиям иногда необходимо менять так называемый тик таймера, т.е. промежуток выполнения очередного шага анимации.**

Это выполняется в Инспекторе объектов.

Нужно выбрать элемент Timer, нажать на кнопку Свойства и там выбрать и изменить параметр Interval (выражается в миллисекундах). В данном примере Interval равен 300 мс.

Заключение:

- Переименуйте форму по названию проекта.
- Сделайте дизайн формы удобный для пользователя.
- Сформируйте отчет о создании приложения «**Анимация в моделировании**» в виде скриншотов в пустом текстовом документе.