Министерство образования и науки РФ
ФГБОУ ВО
ДАГЕСТАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

**А. М. Магомедов**

# Упражнения по компьютерной графике

**МАХАЧКАЛА – 2016**

УДК 004.92

Магомедов А. М.

Упражнения по компьютерной графике. Пособие рассчитано на студентов бакалавриата направлений 010302 и 020302 и содержит материал для практикума по учебной дисциплине «Компьютерная графика».

Рецензенты:

Рамазанов А.-Р.К. – проф., д-р физ.-мат. наук

Лугуев Т.С. – доцент, канд. физ.-мат. наук

Учебное пособие печатается в соответствии с решением учебно-методического совета факультета математики и компьютерных наук Дагестанского государственного университета

# Оглавление

# Введение

Актуальность работы. С расширением направлений обучения на факультете математики и компьютерных наук ДГУ изменилась структура учебных дисциплин. В частности, в учебные планы включена дисциплина «Компьютерная графика», которой отводится значительное количество часов. Методическая поддержка лекционных занятий по данной дисциплине и разработка программного обеспечения для сопровождения лабораторных занятий особенно востребованы на данном этапе преподавания в связи с изменением языковой основы преподавания (с Delphi на C#).

Использованные в данном пособии технологии Windows Forms и Windows Presentation Foundation вполне соответствуют современным стандартам создания настольных графических и мультимедийных приложений.

Структура работы. Работа состоит из четырех частей.

В первой части после кратких сведений об окнах Windows и действиях с макетом окон Visual Studio проведен краткий экскурс в интерфейс графических устройств. Сначала рассмотрены приложения, для создания которых достаточно овладение сокращенным графическим инструментарием, но уже в последнем пункте первой части приложения созданы с применением основных методов класса Graphics. Среди этих приложений: программное создание массива оконных элементов; рисование графа, допускающего интерактивное управление размещением вершин; отображение двудольного графа по заданным в файле спискам связности; управляемая кривая Безье; стираемая рукопись; модель «солнечной системы» и др.

Вторая часть посвящена развитым графическим средствам. Рассмотрены вопросы двойной буферизации и создания масштабируемых приложений, реализации аффинных преобразований (поворот, перенос, масштабирование) средствами класса Matrix из пространства имен System.Drawing.Drawing2D, а

также средства GraphicPath, задачи создания регионов и форм нестандартной конфигурации.

В третьей части рассмотрены вопросы рисования графиков функций одной и двух переменных, на примере сферы изложена схема построения трехмерных фигур, построена интерактивно управляемая модель рельефа.

В четвертой части обсуждается воспроизведение мультимедийных файлов и обработка последовательных кадров видеопотока. Приведены начальные сведения о библиотеке компьютерного зрения OpenCV. В частности, создано приложение, которое взаимодействует с веб-камерой и эффективно отслеживает движение выбранной точки в околокомпьютерном пространстве.

Основной акцент в работе сделан на составление графических проектов, выполнено около 30 упражнений по графике и мультимедиа. В тексте приведены их подробные решения, в электронном приложении – соответствующие проекты Visual C#. Там, где это необходимо, даны разъяснения с целью адаптации работы к проведению лабораторного практикума по компьютерной графике.

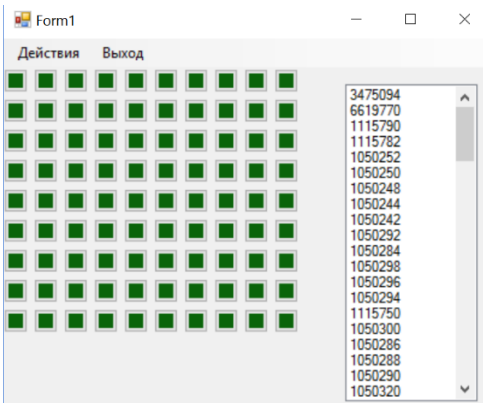
## Часть I. Основные методы класса Graphics

### 1.1. Интерфейс графических устройств (Graphics Device Interface, GDI)

Окнам присваиваются дескрипторы – целые неотрицательные числовые идентификаторы. Кроме того, видимые окна обладают рядом свойств: Parent, Name, Left, Top, Width, Height и др.

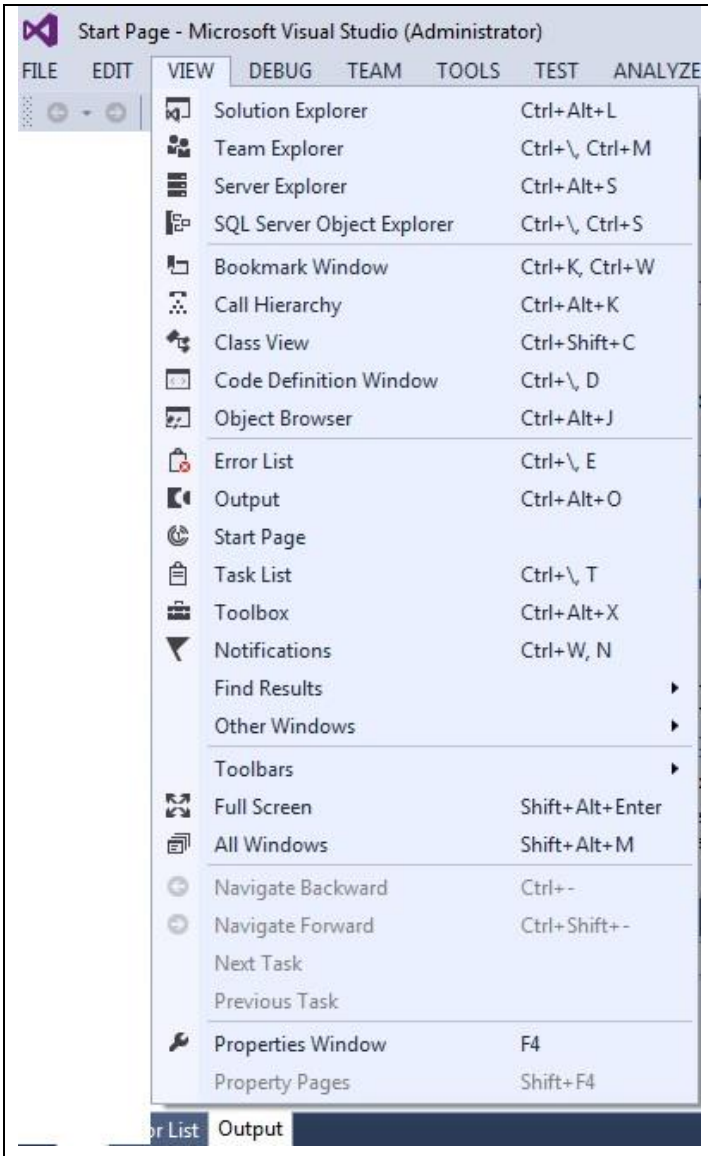
Упражнение (генерация кнопок). Создать меню из двух пунктов: Действия и Выход. При нажатии на первый пункт сгенерировать n=100 кнопок зеленого цвета размерами 20*20, расположенные по 10 кнопок в каждом ряду. Их дескрипторы вывести в listBox1.

```
private void выходToolStripMenuItem_Click(object sender, EventArgs e)
    {
       Application.Exit();
    }

    private void
```




```
генерацияПанелейToolStripMenuItem_Click(object sender, EventArgs e)
    {
       const int n = 100;
       int r = 10;
       Button [] p = new Button [n];
       for (int i=0; i<n; i++)
       {
          p[i] = new Button ();
          p[i].Parent = this;
          p[i].Name = i.ToString();
          p[i].Top = i / r * 25; p[i].Left = i % r * 25;
          p[i].Width = 20; p[i].Height = 20;
          p[i].BackColor = Color.FromArgb(10, 100, 10);
          listBox1.Items.Add(p[i].Handle.ToString());
          p[i].Show();
          //Application.DoEvents();
       }
    }
```

Отступление. Приведем сведения о настройке макетов окон в Visual Studio.




**Обозреватель решений Ctrl+Alt+L**
Командный обозреватель
Обозреватель серверов
Обозреватель объектов SQL Server

**Окно закладок Ctrl+K, Ctrl+W**
Иерархия вызовов
Классы
Окно определения кода
Обозреватель объектов Ctrl+Alt+J

Список ошибок
**Вывод Ctrl+Alt+O**
Начальная страница
Список задач
**Панель элементов Ctrl+Alt+X**
Уведомления
Результаты поиска
Другие окна

Панели инструментов
Во весь экран Shift+Alt+Enter
Все окна Shift+Alt+M

Назад
Вперед
Следующая задача
Предыдущая задача

**Окно свойств F4 (Shift+F4)**
Обновить

Рис. 1. Меню «Вид». Список названий окон и их клавиатурных эквивалентов.

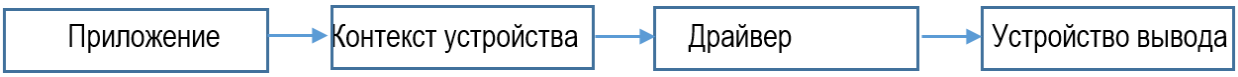
| Рис.1. Меню View | Перевод |
|---|---|

Примечание. Пункт «Сброс макета окон» меню **Окно** устанавливает макет по умолчанию.

Microsoft Windows предоставляет для приложений набор системных функций, реализующих интерфейс графических устройств (Graphics Device Interface, GDI). Для приложений Microsoft .NET Framework доступен также усовершенствованный интерфейс GDI+.

1. GDI предназначен для взаимодействия приложений с графическими устройствами вывода, такими как видеоадаптер, принтер или плоттер. Но создатели приложений, обращающихся к GDI для выполнения операции вывода графического изображения, не в состоянии учитывать характеристики реальных (физических) устройств, неизбежно полагаясь на абстрактные (логические) устройства с феноменальными характеристиками. Такой подход обеспечивает аппаратную независимость.
2. GDI состоит из контекста отображения, инструментов, предназначенных для рисования, и функций для работы с ними. Приложения Microsoft .NET Framework к тому же реализуют возможности интерфейса GDI+ с помощью классов и интерфейсов.
3. Инструменты для рисования — это перья, кисти и шрифты. Контекст отображения можно сравнить с листом бумаги, на котором приложение рисует то или иное графическое изображение, в частности, текст.

Приложение может выбирать в контекст отображения различные инструменты, например, перья различной толщины и цвета, с различными «наконечниками». Поэтому для рисования линии красного или зеленого цвет сначала следует выбрать в контекст отображения соответствующее перо.
4. Контекст устройства выступает в роли связующего звена между приложением и драйвером устройства и представляет собой структуру данных размером примерно 800 байт. Эта структура данных содержит информацию о том, как нужно выполнять операции вывода на данном устройстве (цвет и толщину линии, тип системы координат и т. д.).
5. Схема взаимодействия приложения с устройством вывода:




6. Если приложение получает контекст для отображения в одном из своих окон, такой контекст называется контекстом отображения, если же требуется выполнять операцию вывода для устройства (для принтера или для экрана дисплея) - контекстом устройства. Контексты устройства и отображения содержат описания одних и тех же характеристик и имеют одинаковую структуру.

7. Весь графический инструментарий расположен в пяти пространствах имен:

System.Drawing содержит классы Graphics, Pen, Brush, Bitmap, Font и др.,

System.Drawing.Drawing2D содержит дополнительные классы: AdjustableArrowCap (реализует наконечники для линий), Matrix (реализует аффинные преобразования изображений),

остальные три пространства имен называются: System.Drawing.Printing, System.Drawing.Text, System.Drawing.Imaging.

## 1.2. Класс Graphics. Сокращенный инструментарий рисования

1) Создание холста – объекта класса Graphics

Одно из отличий GDI+ (от «классического» графического интерфейса GDI) – использование класса Graphics, реализующего как свойства контекста отображения, так и инструменты, предназначенные для рисования в этом контексте. Для рисования в окне необходимо получить или создать для этого окна объект класса Graphics. Рассмотрим несколько способов создания такого объекта.

```
Graphics g = CreateGraphics();
Graphics g = pictureBox1.CreateGraphics();
Graphics g =Graphics.FromHwnd (panel1.Handle);

Bitmap b=new Bitmap (pictureBox1.Width, pictureBox1.Height);
g=Graphics.FromImage (b);
```

2) Сохранение рисунка в файл и загрузка рисунка из файла

2.1a. Следующее решение можно рассматривать и как приглашение к двойной буферизации: рисовать не непосредственно на pictureBox1 (присвоим для краткости имя pB1), а на холсте gb объекта b типа Bitmap, соразмерного pB1, в нужное время присваивая объект b свойству pB1.Image.

```
Bitmap b; … b= new Bitmap (pB1.Width, pB1.Height); … gb=Graphics.FromImage (b);
… pB1.Image=b; … b.Save (“1.bmp”);
```

2.1b. `Bitmap b = new Bitmap (r, r); b = (Bitmap) pB1.Image; b.Save(“1.bmp”);`

2.1c. `((Bitmap) (pB1.Image)).Save("000.bmp");`

2.2. Загрузка рисунка из файла или графического контейнера

```
if (File.Exists("1.bmp")) Bitmap b = new Bitmap ("1.bmp");
```

3) Обработка рисунка (масштабирование и прозрачность)

```
b = new Bitmap(b, new Size(w1, h1));
b.MakeTransparent (цвет);
```

4) Вывод рисунка на холст или в граф.контейнер

```
g.DrawImage(b, x, y); или g.DrawImage (img, x,y,w,h);

pictureBox1.Image = img;
```

5) Цвет точки: получение и изменение

```
Color c1 = b.GetPixel(x, y);
Color c2 = Color.FromArgb(c1.R, 100, 10); b.SetPixel(x, y, c2);
```

или

```
if (colorDialog1.ShowDialog() == DialogResult.OK)
{pictureBox1.BackColor = colorDialog1.Color;}
```

В методах класса Graphics используются цвет, перо, кисть, шрифт.

6) Color.Blue или Color.FromArgb(255,0,0)

7) Pen p= new Pen (Color.Blue, 1);

или Pen p= Pens.Blue; p.DashStyle=DashStyle.Dot;

Рисование отрезка прямой: g.DrawLine (p, x1, y1, x2, y2);

8) Пример задания цвета кисти: Brushes.Black.

9) Шрифт: Font f= new Font("Arial", 10);

10) Точка: Point P = new Point (x,y);

11) Вывод строки: DrawString (“Строка”, new Font (“Arial”, 15), Brushes.Black,

new Point (10, 20));

Упражнение (движение множества объектов на заданном фоне). Используя файлы с рисунками фона и объекта, организовать движение множества случайно расположенных копий объекта на заданном фоне. Предусмотреть прозрачность краев объекта, продвижение объектов привязать к тику таймера.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace myplane
{
   public partial class Form1 : Form
   {
      const int n = 200;
      Bitmap sky, plane;
      Graphics g;
      int [] dx = new int [n];
      Rectangle [] rct  = new Rectangle [n];
      Random rnd;
      public Form1()
      { InitializeComponent(); }
      private void button1_Click(object sender, EventArgs e)
      {
         sky = new Bitmap("sky.bmp");      // небо
```

```
    plane = new Bitmap("plane.bmp"); // самолет
    // загрузить и задать фоновый рисунок формы
    this.BackgroundImage =  new Bitmap("sky.bmp");
    // придать прозрачность краям объекта
    plane.MakeTransparent();
    // задать размер формы в соответствии
    // с размером фонового рисунка
    this.ClientSize = new
          Size( new Point(BackgroundImage.Width,  BackgroundImage.Height));
    this.MaximizeBox = false;
    g = Graphics.FromImage(BackgroundImage);
    // инициализация генератора случайных чисел
    rnd = new System.Random();
    // стартовое положение объектов
    for (int i = 0; i < n; i++)
    {
       rct[i].X = -500+rnd.Next(500);          rct[i].Y = 20 + rnd.Next(220);
       rct[i].Width = plane.Width+2;           rct[i].Height = plane.Height+2;
       dx[i] = rnd.Next(2) + 2;
    }
    timer1.Interval = 20;
    timer1.Enabled = true;
}
private void timer1_Tick(object sender, EventArgs e)
{
    // удалим изображение самолетов путем копирования
    // рисунка фона на холст
    g.DrawImage(sky, new Point(0, 0));
    // изменяем координаты самолета
    for (int i = 0; i < n; i++)
    {
       if (rct[i].X < this.ClientRectangle.Width)
       { rct[i].X += dx[i];}
       else
       {
          // если объект достиг правой границы,
          // заново зададим его исходное положение
          rct[i].X = -100 + rnd.Next(60);
          rct[i].Y = 20 +
             rnd.Next(this.ClientSize.Height - 40 - plane.Height);
          dx[i] = rnd.Next(5) + 2;
       }
       g.DrawImage(plane, rct[i].X, rct[i].Y);
    }
    this.Refresh();
```

```
    }
  }
}
```

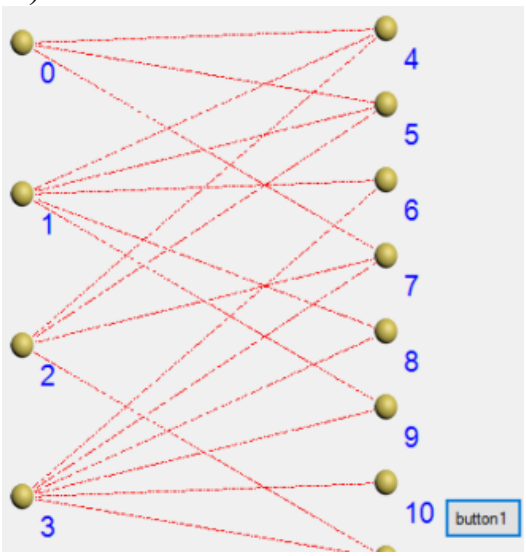
Упражнение (отображение двудольного графа). Информация о двудольном файле G=(X,Y,E), |X|=4, |Y|=8, задана в файле перечислением для каждого из $x_0, x_1, x_2, x_3$ списка смежных вершин (номера смежных вершин перечислены в отдельной строке с разделителем «запятая»):

4,5,7

4,5,6,8,9

4,5,7,11

6,7,8,9,10,11

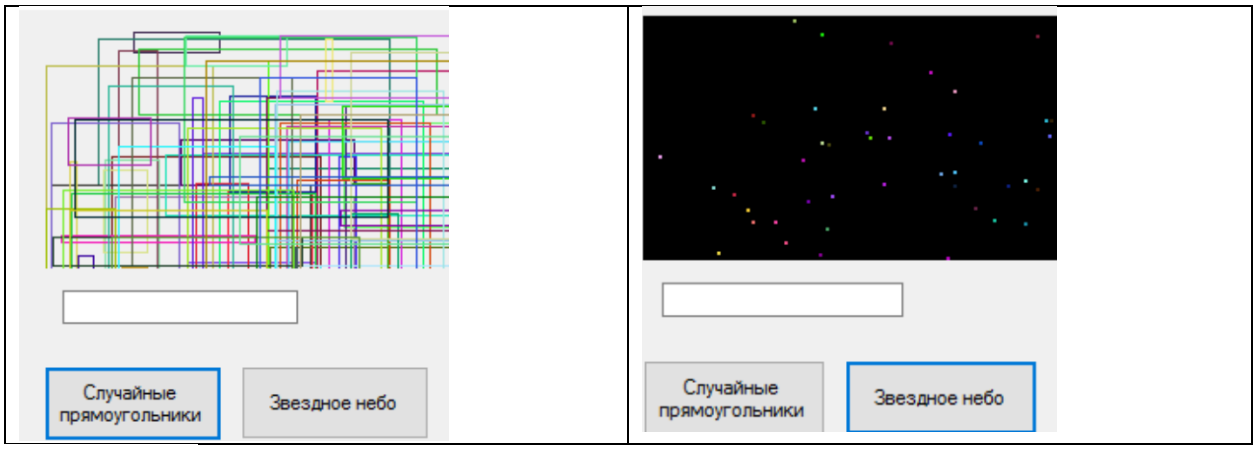



Вывести рисунок графа.

```
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 System.IO;
using System.Drawing.Drawing2D;
namespace WindowsFormsApplication9
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
    }
    private void button1_Click(object sender, EventArgs e)
    {
      Bitmap b = new Bitmap ("3.bmp");
      b = new Bitmap (b, new Size(20, 20));
      b.MakeTransparent(Color.White);
      int n1 = 4, n2 = 8;
      int [] x = new int [12];
      int [] y = new int [12];
      for (int i=0; i<n1; i++)
      {
```

```
            x[i] = this.Width/ 4;
            y[i]= this.Height / (2*n1)*2*i+30;
        }
        for (int i=n1; i<n1+n2; i++)
        {
            x[i] = this.Width/ 4*3;
            y[i]= this.Height / (2*n2)*(2*i-2*n1)+20;
        }
        StreamReader f = new StreamReader ("1.txt");
        string s;
        int[] d = new int [n1];
        int [,] m = new int [n1, 6];
  for (int i=0; i<n1; i++) for (int j=0; j<5; j++) m[i,j]=-1;
        for (int i=0; i<n1; i++)
        {
            s = f.ReadLine(); if (s == "") break;
            string[] t = s.Split(',');
            d[i] = t.Length;
            for (int j = 0; j < d[i]; j++)
            { m[i, j] = int.Parse(t[j]); }
        }
            Graphics g = CreateGraphics();
            for (int i = 0; i < n1; i++)
            {
                for (int j = 0; j < d[i]; j++)
                {
                    int k = m[i, j];
                    Pen p= new Pen (Color.Red, 1);
                    p.DashStyle = DashStyle.DashDot;
  g.DrawLine(p, new Point(x[i], y[i]), new Point(x[k], y[k]));
                }
            }
            for (int i = 0; i < n1 + n2; i++)
                g.DrawImage(b, x[i] - 10, y[i] - 10);
            for (int i = 0; i < n1 + n2; i++)
              g.DrawString (i.ToString(),
new Font ("Arial", 15), Brushes.Blue, x[i] + 10, y[i] + 10);

        }

    }
  }
```

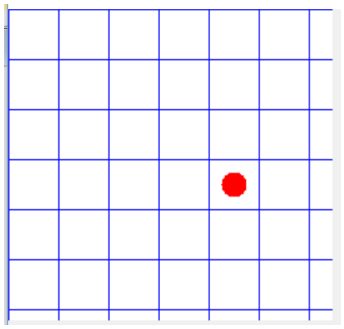
Упражнение (случайные фигуры). При нажатии на первую кнопку вывести случайные прямоугольники, при нажатии на вторую кнопку – звездное небо.




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;
namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      { InitializeComponent(); }
      Random r = new Random();
      Graphics g;
       private void button1_Click(object sender, EventArgs e)
      {
        //Вывод случайных прямоугольников
         g = pictureBox1.CreateGraphics();
         Pen p = new Pen(Color.FromArgb(255, r.Next(255), r.Next(255), 255), 1);
         for (int i = 0; i < 100; i++)
         {
            p = new Pen(Color.FromArgb(255, r.Next(255), r.Next(255), r.Next(255)), 1);
            g.DrawRectangle(p, r.Next(200), r.Next(200), r.Next(200), r.Next(200));
         }
      }
      private void button2_Click(object sender, EventArgs e)
      {
         g = pictureBox1.CreateGraphics();
//Вывод звездного неба
         Pen p = new Pen(Color.FromArgb(255, r.Next(255), r.Next(255), 255), 1);
         SolidBrush br= new SolidBrush
```

```
                    (Color.FromArgb(r.Next(255), r.Next(255), r.Next(255)));
        br.Color=Color.Black;
        g.FillRectangle(br, 0, 0, 300, 300);
        float x, y;
        for (int i = 0; i < 400; i++)
        {
           br.Color = Color.FromArgb(r.Next(255), r.Next(255), r.Next(255));
           x = r.Next(500);
           y = r.Next(500);
           g.FillEllipse (br, x, y, 3, 3);
           Thread.Sleep(10);
        }
     }
  }
}
```

Упражнение (шарик в сетке). Нарисовать таблицу, содержащий шарик в одной из своих ячеек. При щелчке мышью по другой ячейке шарик должен перемещаться в нее.



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      { InitializeComponent(); }
      const int d = 40, n = 8, r = d / 4;
      private void f(int x, int y, int r)
      {
         Graphics g = this.pictureBox1.CreateGraphics();
         g.Clear(Color.White);
         for (int i = 0; i < 8; i++)
         {
            g.DrawLine(Pens.Blue, new Point(0, i * d), new Point(n * d, i * d));
         }
         for (int j = 0; j < 8; j++)
         {
```

```
            g.DrawLine(Pens.Blue, new Point(j * d, 0), new Point(j * d, n * d));
            g.FillEllipse(Brushes.Red, x * d + r, y * d + r, 2 * r, 2 * r);
        }
    }
    private void button1_Click(object sender, EventArgs e)
    {
        int x = 1, y = 1;
        f(1, 1, r);
    }
    private void pictureBox1_MouseClick(object sender, MouseEventArgs e)
    {
        int x = e.X; int y = e.Y;
        f(x / d, y / d, r);
    }
  }
}
```

### 1.3. Основные методы класса Graphics

Рассмотрим основные методы класса. За исключением первого метода остальные разделим на две части: с префиксом Draw или с префиксом Fill.

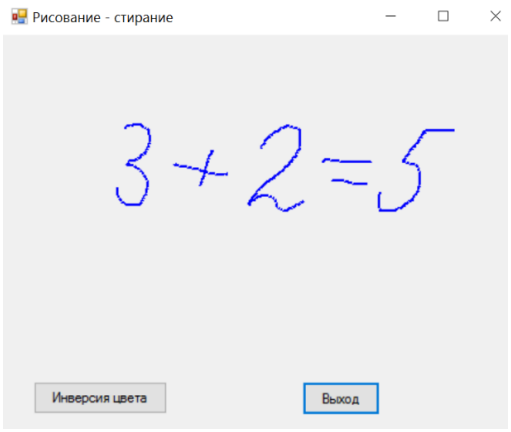
Clear (Color C) – очищает холст, заливая его заданным цветом (по умолчанию – белый цвет).
DrawArc (Pen P, Rectangle R, Single Start, Single Finish); пером P вычерчивается дуга эллипса, вписанного в прямоугольник R, в секторе, ограниченном углами Start и Finish.

DrawBezier (Pen P, Point T0, Point T1, Point T2, Point T3); кривая Безье проходит через точки T0 и T3 так, что отрезки T0—T1 и T2—T3 касаются ее в концевых точках.

DrawClosedCurve (Pen P, Point [] Points); замкнутая кривая, проходящая по заданным точкам массива Points.
DrawCurve (Pen P, Point [] Points); кривая, проходящая по заданным точкам массива Points.
DrawEllipse (Pen P, Rectangle R); эллипс, вписанный в прямоугольник R.
DrawImage (Image Img, Rectangle R); вычерчивает изображение img в области R.
DrawLine (Pen P, Point T0, Point T1); отрезок, соединяющий точки T0, T1.
DrawPolygon (Pen P, Points [] Points); замкнутый многоугольник.
DrawPath (Pen P, GraphicsPath Path); вычерчивает объект Path пером P.
DrawRectangle (Pen P, Rectange R); прямоугольник.
DrawString (string S, Font F, Brush B, Single Start); начиная с точки Start, выводит строку S текста шрифтом F, используя кист B.

FillClosedCurve (Brush B, Point[] Points); закрашивает кистью B внутреннюю область кривой, проходящей через точки массива P.
FillPath (Brush B, GraphicsPath Path); закрашивает внутреннюю область объекта Path.
FillEllipse (Brush B, Rectangle Rect);
FillPie (Brush B, Rectangle Rect, Single Angle1, Angle2); закрашивает внутреннюю область сектора эллипса
FillPolygon (Brush B, Point [] Points); закрашивает внутреннюю область полигона.
FillRectangles (Brush B, Rectangle [] Rects); закрашивает внутренние области нескольких прямоугольников.
FillRegion (Brush B, Region Reg); Закрашивает внутреннюю часть региона.

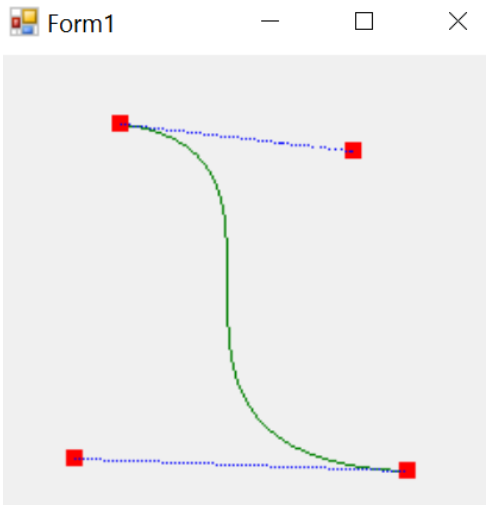
Упражнение (стираемая рукопись на форме). Составить программу рисования на форме курсором мыши, нажатие на кнопку должно переключать между режимами рисования и стирания.




```
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;

namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      { InitializeComponent(); }
      bool toggle = false;
      public bool EnableDraw = false;
      public Point prevPoint, curPoint;
      public Pen Pen1 = new Pen(Color.Blue, 2);
      public Graphics g;

      private void Form1_Load(object sender, EventArgs e)
      { g = this.CreateGraphics(); }

private void Form1_MouseDown(object sender, MouseEventArgs e)
   {
    EnableDraw = true; //Установим флаг нажатости левой кнопки
```

```
    prevPoint.X = e.X; //и запомним координаты точки нажатия
    prevPoint.Y = e.Y;
   }

   private void Form1_MouseUp(object sender, MouseEventArgs e)
      {
       EnableDraw = false;
       // Опустим флаг о нажатии левой кнопки
      }
 private void Form1_MouseMove(object sender, MouseEventArgs e)
    {
      if (EnableDraw)
       //Если левая кнопка мыши нажата, то
        {
          curPoint.X = e.X; curPoint.Y = e.Y;
          // 1) проведем линию к текущей точке
          g.DrawLine(Pen1, prevPoint, curPoint);
          prevPoint = curPoint;
          // 2) обновим координаты последнего нажатия.
        }
      }
      private void button1_Click(object sender, EventArgs e)
      {Application.Exit();}
      private void button2_Click(object sender, EventArgs e)
      {
        toggle = !toggle;
        if (!toggle)
        {Pen1.Color = Color.Blue; Pen1.Width = 2; } else
        {
          Pen1.Color = this.BackColor;
          Pen1.Width = 12;
        }
      }
   }
}
```

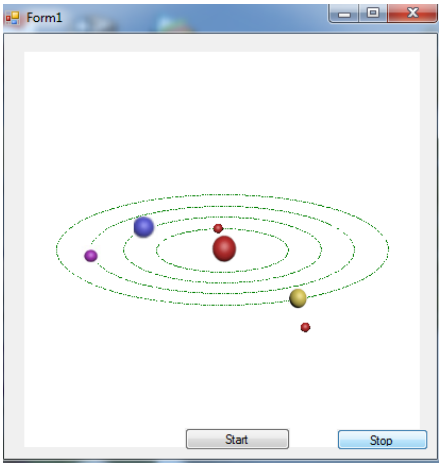
Упражнение (кривая Безье). Вывести анимированную кривую Безье: отобразите все 4 точки в виде маленьких прямоугольников, предусмотрите визуальное перетаскивание каждого из них с перерисовкой кривой.




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication11
{
   public partial class Form1 : Form
   {
      Graphics g;
      public Form1()
      {
         InitializeComponent();
         g = this.CreateGraphics();
      }
      int [] x={10, 100, 50, 200};
      int [] y={10, 50, 160, 200};
      int r=4;
      int d=6;
   void Dr (Graphics g)
   {
    g.Clear(this.BackColor);
    g.DrawBezier
(Pens.Green, x[0], y[0], x[1], y[1], x[2], y[2], x[3], y[3]);
    for (int i = 0; i < 4; i++)
    g.FillRectangle(Brushes.Red, x[i] - r, y[i] - r, 2*r, 2*r);
    Pen p = new Pen(Color.Blue, 1); p.DashStyle=DashStyle.Dot;
    g.DrawLine (p, x[0], y[0], x[1], y[1]); g.DrawLine(p, x[2], y[2], x[3], y[3]);
     }
      private void Form1_Activated(object sender, EventArgs e)
      {
       Application.DoEvents();
       Dr(g);
      }

      int k = 0;
      bool bmDown = false;
```

```
  private void Form1_MouseDown(object sender, MouseEventArgs e)
  {
   bmDown = true;
   for (int i = 0; i < 4; i++)
     if (Math.Abs(e.X - x[i]) + Math.Abs(e.Y - y[i]) < d) k = i;
  }

  private void Form1_MouseUp(object sender, MouseEventArgs e)
  { bmDown = false; }

  private void Form1_MouseMove(object sender, MouseEventArgs e)
  {
   if (!bmDown) return;
   x[k] = e.X; y[k] = e.Y;
   Dr(g);
  }
 }
}
```

Упражнение (планетная модель). Составьте программу вращения «планет» вокруг «солнца» по эллипсоидальным орбита. Для одной из планет предусмотрите спутник, вращающийся вокруг нее.

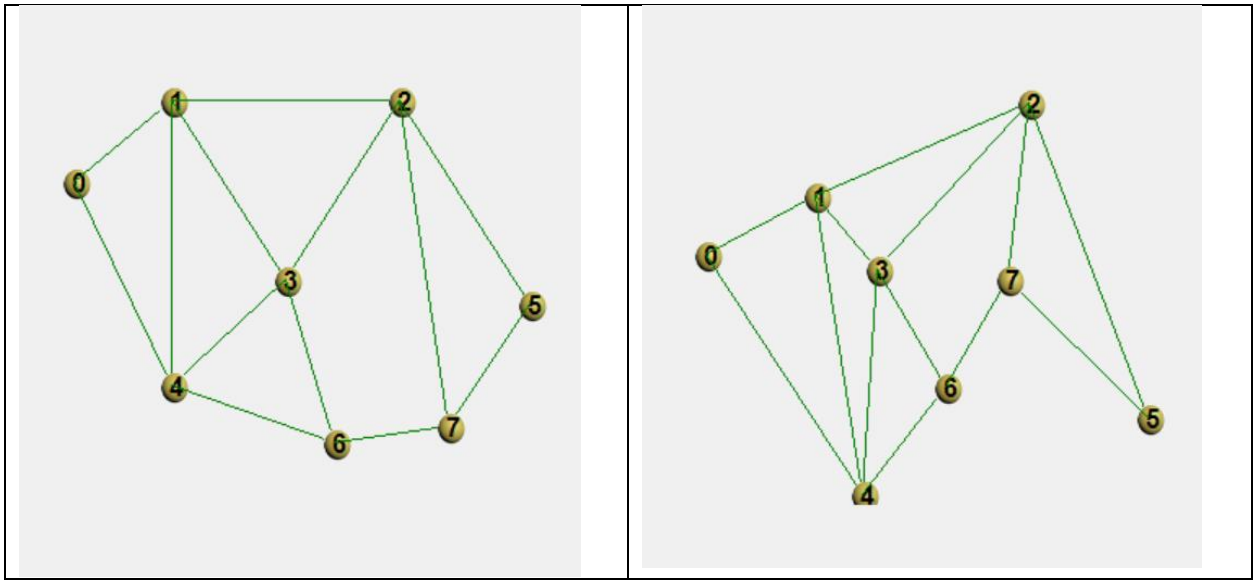



```
using System;

using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;

namespace WindowsFormsApplication10
{
   public partial class Form1 : Form
   {
      bool Rotate;
      public Form1()
      {
         InitializeComponent();
         Rotate = true;
      }
```

```
      private void button1_Click(object sender, EventArgs e)
      {
         int w = pictureBox1.Width;
         pictureBox1.Height = w;
         int x0 = w / 2, y0 = x0;
         int n = 4;
         int[] R = { x0 / 3, x0 / 6*3, x0 / 6*4, x0 / 7*6 };
         double[] a = { 0, 0, 0, 0 };
         double[] da = { 0.01, 0.015, 0.02, 0.024 };
         int [] r = {10, 22, 18, 20};
         int r0 = 35;
         //---------------------------------
         Bitmap [] pl = new Bitmap[n];
         for (int i = 0; i < n; i++)
         {
            pl[i] = new Bitmap(i.ToString() + ".bmp");
            pl[i] = new Bitmap(pl[i], new Size(r[i], r[i]));
            pl[i].MakeTransparent();
         }
         Bitmap b0 = new Bitmap("00.bmp");
         b0 = new Bitmap(b0, new Size(r0, r0));
         b0.MakeTransparent();
         //---------------------------------
         Graphics g = pictureBox1.CreateGraphics();
         Bitmap b = new Bitmap (w,w);
         Graphics gb = Graphics.FromImage(b);
         //---------------------------------
         Pen p = new Pen(Color.Green, 1);
         p.DashStyle = DashStyle.DashDotDot;
         while (Rotate)
         {
            gb.Clear(Color.White);
            for (int i=0; i<n; i++)
            {
               a[i] = a[i] + da[i];
int x = x0+(int)Math.Round(R[i] * Math.Cos(a[i]));
int y = y0 + (int)Math.Round(R[i]/3 * Math.Sin(a[i]));
 gb.DrawEllipse(p, x0 - R[i], y0 - R[i]/3, 2 * R[i], 2 * R[i]/3);
 gb.DrawImage(pl[i], x - r[i] / 2, y - r[i] / 2);
               if (i == n - 1)
               {
int mR = 30;
int x1 = x + (int)Math.Round(mR/2 * Math.Cos(a[i]));
int y1 = y + (int)Math.Round(mR * Math.Sin(a[i]));
gb.DrawImage(pl[0], x1 - r[0] / 2, y1 - r[0] / 2);
```

```
            }
          }
          gb.DrawImage(b0, x0 - r0 / 2, y0 - r0 / 2);
          g.DrawImage(b, 0, 0);
          Application.DoEvents();
          Thread.Sleep(10);
        }
      }
      private void button2_Click(object sender, EventArgs e)
      {
        Rotate = false;
      }
    }
}
```

Упражнение (подвижный граф). Нарисовать граф с заданными связями, вершины которого можно перетаскивать мышью.




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Graph
{
  public partial class Form1 : Form
```

```
{
   public Form1()
   {
      InitializeComponent();
      button1.Text = "Нарисовать граф";
   }
   const int n = 8;
   int[,] m = new int[n, n]
   {
   {0,1,0,0,1,0,0,0},   {1,0,1,1,1,0,0,0},
   {0,1,0,1,0,1,0,1},   {0,1,1,0,1,0,1,0},
   {1,1,0,1,0,0,1,0},   {0,0,1,0,0,0,0,1},
   {0,0,0,1,1,0,0,1},   {0,0,1,0,0,1,1,0}
   };
   int [] x=new int [n]{20, 80, 220, 150, 80, 300, 180, 250};
   int [] y=new int [n]{100, 50, 50, 160, 225, 175, 260, 250};
   int r = 4;
   Graphics g, g1;
   Bitmap bmp, vertex;

   bool b = false;
   private void DG(Graphics g2)
   {
      bmp = new Bitmap(pictureBox1.Width, pictureBox1.Height);
      g = Graphics.FromImage(bmp);
      g.Clear(this.BackColor);
      for (int i = 0; i < n; i++)
      {
         for (int j = 0; j < i; j++)
            if (m[i, j] == 1)
               g.DrawLine(Pens.Green, new Point(x[i], y[i]), new Point(x[j], y[j]));
         g.DrawImage(vertex, x[i] - 2 * r, y[i] - 2 * r, 20, 20);
         g.DrawString(i.ToString(),
            new Font("arial", 12,
            FontStyle.Bold),
            Brushes.Black,
            new Point(x[i]-r, y[i]-2*r));
      }
      g2.DrawImage(bmp, 0, 0);
   }
   int count = -1;
   private void button1_Click(object sender, EventArgs e)
   {
      DG(g1);
      count++;
```

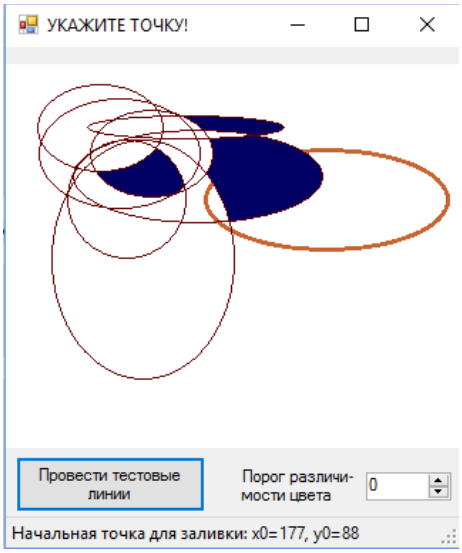
```
    }
    int k = -1;
    private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
    {
      b = true;
      for (int i = 0; i < n; i++)
        if ((Math.Abs(x[i] - e.X) < 2*r) && (Math.Abs(y[i] - e.Y) < 2*r))
          k = i;
    }
    private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
    {
      b = false; k=-1;
    }
    private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
    {
      if (b == false) return;
      if (k == -1) return;
      x[k] = e.X;
      y[k] = e.Y;
      DG(g1);
    }
    private void Form1_Resize(object sender, EventArgs e)
    {
      if (count < 1) return;
      Application.DoEvents();
      g1 = pictureBox1.CreateGraphics();
      DG(g1);
    }
    private void Form1_Activated(object sender, EventArgs e)
    {
      vertex = (Bitmap)pictureBox2.Image;
      vertex.MakeTransparent(Color.White);
      pictureBox2.Visible = false;
      Application.DoEvents();
      g1 = pictureBox1.CreateGraphics();
      DG(g1);
    }
  }
}
```

Упражнение (заливка). Составить функцию-процедуру

FloodFill (Bitmap b, Color FillColor, Color BorderColor, int x, int y, int d)

для заливки части холста объекта b цветом FillColor, начиная с точки (x,y), в качестве ограничителя заливки выступают точки заданного цвета BorderColor.

Дополнительное свойство: функция игнорирует различие цветов точек, если разница между соответствующими составляющими цветов (R, G, B) точек не превышает значение последнего параметра d.




```
/*
Использована очередь (Queue) точек, подлежащих заливке; точки, подлежащие заливке, заносятся (с закрашиванием) в очередь по принципу поиска в ширину.
В тестовом примере, приведенном для демонстрации возможностей процедуры, на форме будут размещены:
- объект PictureBox, на которой программа нарисует несколько линий цвета BorderColor, призванного ограничить заливку другим цветом FillColor, и один "специальный" контур с иным цветом (преодоление заливкой специального контура зависит от установленного порогового значения элемента NumericsUpDown (при пороговом значении 0 - преодолевает);
- кнопка Button, нажатие на которую приводит к рисованию тестовых линий;
- строка статуса для отображения координат точки щелчка кнопкой мыши с целью задания начальной точки заливки;
- элемент NumericUpDown для управления последним параметром.
*/
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;
namespace WindowsFormsApplication26
{
   public partial class Form1 : Form
```

```
  {
    Bitmap b;  //по размерам будет соответствовать pictureBox1
    // полезен тем, что обладает методами GetPixel, PutPixel
    Graphics g, gb; //для определения равно-размерных холстов:
    //для объекта b и для pictureBox1
    Color FillColor, BorderColor; //цвета заливки и границы
    Queue<Point> Q; //очередь точек, ожидающих заливки,
    //включение точки обусловлено наличием цепочки (без точек
    // ограничивающего цвета BorderColor), соединяющей с начальной
    // точкой заливки
    Pen p1; //для рисования тестовых линий
    int x0, y0; //начальная точка закрашивания
    //будет указана щелчком мыши
    const int r0 = 0, g0 = 0, b0 = 100, r1 = 100, g1 = 0, b1 = 0; //цветовые составляющие
    int dam_board = 0;
//пороговое значение различимости цветовых составляющих
//устанавливается в интерактивном режиме (см. ниже)
    public Form1()
    {
      InitializeComponent();
      b = new Bitmap(pictureBox1.Width, pictureBox1.Height);
      //создан Bitmap-объект, равный pictureBox1 по размерам
      g = pictureBox1.CreateGraphics();
      gb = Graphics.FromImage(b);    //созданы оба холста
      FillColor = Color.FromArgb(r0, g0, b0); // цвет заливки
      BorderColor = Color.FromArgb(r1, g1, b1);// цвет границы
      p1 = new Pen(BorderColor, 1);                    //для рисования тестовых линий
      this.button1.Text = "Провести тестовые линии";
    }
    public void IfAdd(Bitmap b, Color CF, Color CB, int x, int y, int d = 0)
    //Если хотя бы одна из трех цветовых составляющих
    //точки (x,y) объекта различается более чем на d от
    //соответствующих составляющих цветов CF и CB, точку
    //(x,y) добавить к очереди Q и закрасить цветом CF.
    {
     Color C = b.GetPixel(x, y);
     if (
     ((Math.Abs(C.R - CB.R) > d) || (Math.Abs(C.G - CB.G) > d) ||
      (Math.Abs(C.B - CB.B) > d)) &&
     ((Math.Abs(C.R - CF.R) > d) || (Math.Abs(C.G - CF.G) > d) ||
      (Math.Abs(C.B - CF.B) > d))
       )
      {
        b.SetPixel(x, y, CF);
        Q.Enqueue(new Point(x, y));
```

```
        }
    }
    public void FloodFill(Bitmap b, Color FillColor, Color BorderColor, int x, int y, int d = 0)
    {
        Q = new Queue<Point>();
        //инициализация очереди точек Q, подлежащих заливке
        b.SetPixel(x, y, FillColor);
        //точку начала заливки закрасили и
        Q.Enqueue(new Point(x, y));
        //включили в очередь в качестве начального элемента
        while (Q.Count > 0)
        //пока в очереди Q остаются элементы
        {
            Point P = Q.Dequeue();
            //извлекли и удалили элемент с начала очереди
            x = P.X; y = P.Y;
            //рассмотрим соседние (в пределах объекта b) элементы,
            //подлежащие заливке:
 if (x > 0) IfAdd(b, FillColor, BorderColor, x - 1, y, d);
 if (x < b.Width - 1) IfAdd(b, FillColor, BorderColor, x + 1, y, d);
 if (y > 0) IfAdd(b, FillColor, BorderColor, x, y - 1, d);
if (y < b.Height - 1) IfAdd(b, FillColor, BorderColor, x, y + 1, d);
        //если имеются по соседству закрашиваемые точки,
        //они заливаются цветом FillColor и добавляются в очередь,
        //но еще "не обработаны", т.е. не рассмотрены их соседи
        }
    }
    private void button1_Click(object sender, EventArgs e)
    {
        dam_board = (int)numericUpDown1.Value;
        gb.Clear(Color.White);
     //Построим для целей тестирования несколько замкнутых фигур
     //цвета BorderColor и одну специального цвета фигуру
 Pen pen2 = new Pen(Color.FromArgb(r1 + 100, g1 + 100, b1 + 50), 3);
        gb.DrawEllipse(pen2, 140, 60, 170, 70);
        Random r = new Random();
        for (int i = 0; i <= 6; i++)
        {
            int x1 = r.Next(10, pictureBox1.Width/4 - 10);
            int y1 = r.Next(10, pictureBox1.Height/4 - 10);
            int w1 = r.Next(10, 2*pictureBox1.Width/3 - 10);
            int h1 = r.Next(10, 2*pictureBox1.Height/3 - 10);
            gb.DrawEllipse(p1, x1, y1, w1, h1);
        }
        g.DrawImage(b, 0, 0);
```

```
        this.Text = "УКАЖИТЕ ТОЧКУ!";
    }
  private void pictureBox1_MouseClick(object sender, MouseEventArgs e)
    {
        //Щелчом кнопки мыши укажем начальную точку заливки
        x0 = e.X; y0 = e.Y;
        statusStrip1.Items[0].Text =
String.Format("Начальная точка для заливки: x0={0}, y0={1}", x0, y0);
        FloodFill(b, FillColor, BorderColor, x0, y0, dam_board);
        g.DrawImage(b, 0, 0);
    } } }
```

# Часть II. Развитые средства графики

## 2.1. Вывод линий с наконечниками

Ранее уже отмечалось, что класс AdjustableArrowCap из пространства имен System.Drawing.Drawing2D реализует наконечники для линий.

Приведем приме конструктора класса AdjustableArrowCap.

```
AdjustableArrowCap(//конструктор одноименного класса
float width, // ширина стрелки
float height,// высота стрелки
bool isFilled // заполнен ли наконечник стрелки
)
AdjustableArrowCap Arrow1 = new AdjustableArrowCap(6, 6, false);
    Pen pen1 = new Pen(Color.Black);
    pen1.CustomStartCap = Arrow1;
    pen1.CustomEndCap = Arrow1;
e.Graphics.DrawLine(pen1, 50, 50, 200, 50);
```

Применение показано ниже при выполнении упражнения масштабируемого вывода с перерисовкой формы.

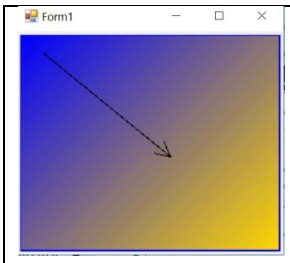
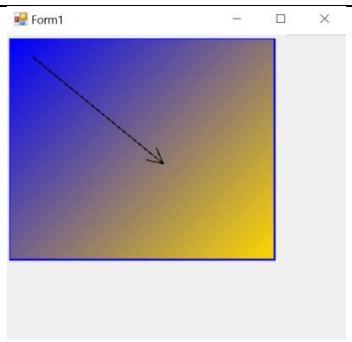
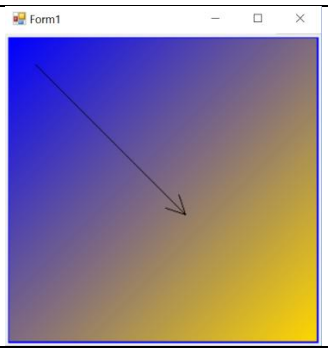
Если требуется вывести на форму (или элемент формы Control) рисунок, не искажающийся при минимизации формы или ее частичном сокрытии, целесообразно поместить весь код рисования в обработчик события Form.Paint или Control.Paint.

Объяснение: при последующем (после сокрытия) отображении формы Windows генерирует событие Paint, принуждая форму перерисовать себя. Таким образом, для правильного обновления окна весь собственный код рисования всегда следует помещать в обработчик события Paint. Для удобства программистов в обработчик события Paint передается параметр PaintEventArgse, который ссылается на объект e.Graphics, представляющий поверхность рисования элемента управления или формы.

Упражнение (масштабируемый вывод с градиентной заливкой и стрелой). Отобразить прямоугольник с градиентной заливкой – переходом двух цветов под углом 45 градусов и отрезок с наконечником (стрелка). Предусмотреть масштабирование при изменении размеров формы.

```
Using System.Drawing.Drawing2D;

Private void Form1_Paint (object sender, PaintEventArgs e)
```

```
    {
int w = this.ClientRectangle.Width - 10;
int h = this.ClientRectangle.Height - 10;
Rectangle r = new Rectangle(5, 5, w, h);
// Рисование границы прямоугольника.
Pen P1 = new Pen(Color.Blue, 3);
e.Graphics.DrawRectangle(P1, r);
// Градиентное закрашивание прямоугольника
LinearGradientBrush B1 = new LinearGradientBrush(r, Color.Blue, Color.Gold, 45);
e.Graphics.FillRectangle(B1, r);
AdjustableArrowCap Arrow1 = new AdjustableArrowCap(w/16,w/16, false);
Pen pen1 = new Pen (Color.Black);
//pen1.CustomStartCap = Arrow1;
        pen1.CustomEndCap = Arrow1;
e.Graphics.DrawLine (pen1, w/10, h/10, w/10*6, h/10*6);
}

Private void Form1_Resize(object sender, EventArgs e)
{ this.Invalidate(); }
```

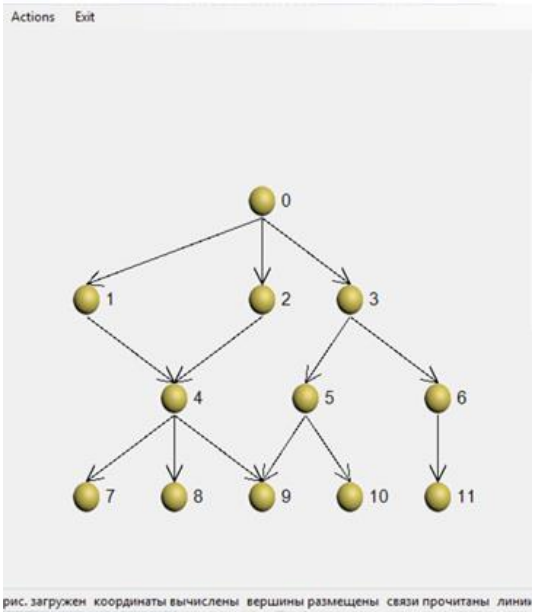
| | | |
|---|---|---|
|  |  |  |
| Вначале программа не содержитобработчик Form1_Resize | Поэтому при увеличении размеров формы не происходит ее перерисовка | Добавили обработчик Form1_Resizeи изменили размеры формы |

В данном примере при рисовании используются размеры формы. Поэтому целесообразно обновлять форму при перерисовке: `this.Invalidate()`.

Упражнение (ориентированный ациклический граф с заданным в файле списком связей). В очередной i-й строке файла с пробельным разделителем перечислены номера вершин, к которым от вершины i проведены дуги:

1 2 3 / 4 / 4 / 5 6 / 7 8 9 / 9 10 / 11

(здесь для обозначения переноса строки использован символ слэш). Нарисовать граф, предусмотреть вывод вершины в виде рисунка сферы (рисунок предварительно сохранить в файле).




```
using System;
usingSystem.Collections.Generic;
usingSystem.ComponentModel;
usingSystem.Data;
usingSystem.Drawing;
usingSystem.Linq;
usingSystem.Text;
usingSystem.Threading.Tasks;
usingSystem.Windows.Forms;
using System.IO;
usingSystem.Drawing.Drawing2D;

namespace Retro
{
public partial classForm1 :Form
{ //Изображение для вершин
Bitmap b; int r2 = 30; int r = 15; //новые размеры
Graphics g;
int n = 12; int md=3;
int[] x;
int[] y;
int h, w, dh, dw;
int[] d;
int[,] m;

public Form1()
      {
InitializeComponent();
this.Width = 600; this.Height = 600;
         g= this.CreateGraphics();
         d = new int[n];
         m = new int[n, md];
         x = new int[n];
         y = new int[n];
```

```
        w = this.ClientRectangle.Width - listBox1.Width-10;
        h = this.ClientRectangle.Height - 10;
        dh = h / 6; dw = w / 6;
x[7] = dw * 1; x[8] = dw * 2; x[9] = dw * 3; x[10] = dw * 4; x[11] = dw * 5;
x[4] = x[8]; x[5] = (x[9]+x[10])/2; x[6] = x[11]; x[3] = x[10]; x[0] = x[2]=x[9]; x[1] = x[7];
y[0] = 2 * dh; y[1] = y[2] = y[3] = 3 * dh; y[4] = y[5] = y[6] = 4 * dh;
y[7] = y[8] = y[9] = y[10] = y[11] = 5 * dh;
    }
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
    {Application.Exit(); }
private void loadBMPToolStripMenuItem_Click(object sender, EventArgs e)
    {
OpenFileDialog f = new OpenFileDialog();
f.InitialDirectory = Directory.GetCurrentDirectory();
f.ShowDialog();
        b = newBitmap(f.FileName);
        b = newBitmap(b, newSize(r2, r2));
        b.MakeTransparent();
        statusStrip1.Items[0].Text = "рис. загружен";
    }
private void setCoorlinatesToolStripMenuItem_Click(object sender, EventArgs e)
    { statusStrip1.Items[1].Text = "координатывычислены"; }
private void layOutVerticsToolStripMenuItem_Click(object sender, EventArgs e)
    {
for (inti = 0; i< n; i++)
        {
g.DrawImage(b, x[i] - r, y[i] - r);
g.DrawString(i.ToString(), newFont("Arial", 12), Brushes.Black, x[i]+r, y[i]-2*r/3);
        }
        statusStrip1.Items[2].Text = "вершиныразмещены";
    }
private void setLinksToolStripMenuItem_Click(object sender, EventArgs e)
    {
StreamReader f = newStreamReader("1.txt");
if(!File.Exists("1.txt")) MessageBox.Show("Ненайден");
string s;
for (inti = 0; i< n; i++)
         for (int j = 0; j < md; j++) m[i, j] = -1;
//вначале связи отсутствуют
```

```
for (inti = 0; i< n; i++)
{
 s = f.ReadLine(); //прочитали очередную строку
 if (s == "") break; //если строка пуста, дальше файл не смотрим
 string[] t = s.Split(' ');//t - массив подстрок в строке i
 d[i] = t.Length;
 for (int j = 0; j < d[i]; j++)
 {
  m[i, j] = int.Parse(t[j]); //j--я связь вершины i
  listBox1.Items.Add (string.Format("i={0}, j={1}, k={2}", i,j, t[j]));
}
 }
 statusStrip1.Items[3].Text = "связи прочитаны";
 }
private void draw ArrowsToolStripMenuItem_Click(object sender, EventArgs e)
{
for (inti = 0; i< n; i++)
        {
for (int j = 0; j < d[i]; j++)
            {
             int k = m[i, j];
             Pen p = new Pen(Color.Red, 1);
             p.DashStyle = DashStyle.DashDot;
AdjustableArrowCap Arrow1 = newAdjustableArrowCap(r, r, false);
Pen pen1 = newPen(Color.Black);
             pen1.CustomEndCap = Arrow1;
             g.DrawLine(pen1, x[i], y[i]+r, x[k], y[k]-r);
            }
        }
        statusStrip1.Items[4].Text = "линии проведены";
    }
private void Form1_Resize(object sender, EventArgs e)
    {
this.Invalidate();
    }
  }
}
```

## 2.2. Свойство Transform класса Graphics

Важнейшим из свойств класса Graphics можно считать свойство Transform, данному свойству можно присвоить матрицу преобразования координат – объект класса Matrix из System.Drawing.Drawing2D. Использование класса Matrix позволяет представить все аффинные преобразования однообразно, в виде произведения вектора из *однородных* координат (столбца (x, y, 1)) каждой точки плоскости на матрицу. Так, преобразование поворота вокруг начала координат на угол $\alpha$ можно представить как результат умножения матрицы A (см. ниже), сдвиг X=x-dx, Y=y-dy – как результат умножения матрицы B, масштабирование $X = x \cdot k_x, Y = y \cdot k_y$ – как результат умножения матрицы C:

A

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix},$$

Matrix m = new Matrix(); m.RotateAt (angle, new Point (x, y)); g.Transform = m;

B

$$\begin{pmatrix} 1 & 0 & -dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix},$$

Matrix m = new Matrix(); m. Translate (dx,dy); g.Transform = m;

$$\begin{pmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix},$$

Matrix m = new Matrix(); m. Scale (0.5, 0.5); g.Transform = m;

Объявление класса Matrix находится в пространстве имен System.Drawing.Drawing2D, поэтому требуется подключение данного пространства имен: using System.Drawing.Drawing2D.

На первый взгляд, представляется странным отсутствие видимой связи с тем объектом, который предстоит повернуть. Объяснение заключается в том, что поворачивается не рисунок, а система координат (вместе с ней – наблюдатель), а уже в повернутой системе координат рисуется объект.

## 2.3. Свойства и методы класса GraphicPath

GraphicsPath участвует, как мы ранее видели, в некоторых примитивах класса Graphics. Данный класс применяется также совместно с классами Region и Matrix.

Объект класса GraphicsPath называют контуром или фигурой. Контур полезен для отображения различных очертаний, заполнения внутренних

областей, создания областей отсечения, локализации действия аффинных преобразований – поворота, переноса, масштабирования.

Объявление объекта класса GraphicsPath:

GraphicsPath p1 = new GraphicsPath();

Значение свойства FillMode определяет, как заполняются внутренние области фигур в объекте GraphicsPath, свойство PathPoints – массив точек контура, свойство PointCount – их количество.

Контур может состоять из любого числа последовательно добавленных фигур (контуров). Каждая фигура либо составлена из последовательности линий и кривых, либо является геометрическим примитивом. Очередная добавленная фигура считается текущей.

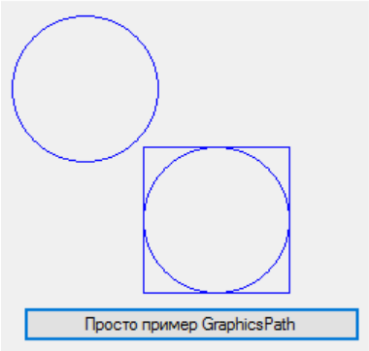
Например,

p1.AddEllipse(new Rectangle(50, 50, 50, 50));

Когда линии и кривые добавляются к контуру, то добавляется неявная линия, соединяющая конечную точку текущей фигуры с начальной точкой новых линий и кривых, чтобы сформировать последовательность соединенных линий и кривых.

Упражнение (построение простого контура GraphicsPath).

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace ПримерGraphicsPath1
{
  public partial class Form1 : Form
  {
    public Form1()
    {InitializeComponent(); }

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



```
    {
        Graphics g = this.CreateGraphics();
        GraphicsPath gP = new GraphicsPath(
                new Point[] {
new Point(10, 10), new Point(132, 154),
new Point(100,50), new Point(133,105)},
                new byte[] {
(byte)PathPointType.Start, ( byte) PathPointType.Line,
(byte) PathPointType.Line, (byte) PathPointType.Bezier
                        }
                                );
    gP.AddEllipse (new Rectangle (100, 100, 100, 100));
    gP.AddRectangle(new Rectangle(100, 100, 100, 100));
    g.DrawPath(Pens.Blue, gP);
 g.DrawEllipse(Pens.Blue, new Rectangle(10, 10, 100, 100));
    gP.Dispose();
     }
  }
}
```

У фигуры есть направление, определяющее, как отрезки прямых и кривых следуют от начальной точки к конечной. Направление задается либо порядком добавления линий и кривых к фигуре, либо геометрическим примитивом. Направление используется для определения внутренних областей контура для целей отсечения и заполнения.

Другие методы: AddArc, AddBezier, AddBeziers, AddClosedCurve, AddCurve, AddEllipse, AddLine, AddLines, AddPath, AddPie, AddPolygon, AddRectangle, AddRectangles, AddString. Об их смысле догадаться нетрудно.

Метод Flatten преобразует каждую кривую в данном контуре в последовательность соединенных отрезков прямых. IsVisible определяет, содержится ли указанная точка в контуре, Transform применяет матрицу преобразования к объекту GraphicsPath.

Отступление. Начальная точка фигуры – первая точка в последовательности соединенных линий и кривых. Конечная точка – последняя точка в последовательности.

Фигура, состоящая из последовательности соединенных линий и кривых (чьи начальная и конечная точки могут совпадать), является разомкнутой фигурой, если она не замкнута явно. Фигура может быть замкнута явно с

помощью метода CloseFigure, который замыкает фигуру путем соединения конечной и начальной точек линией. Фигура, состоящая из геометрического примитива, является замкнутой.

### 2.4. Фигурные вырезания

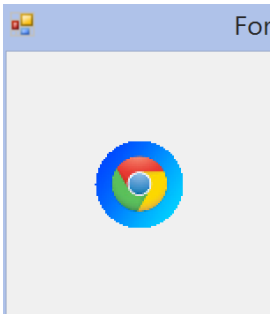
Фигурные изображения создаются с помощью технологии вырезания и создания масок. Если создать маску в объекте Graphics, то от любого изображения сохранится только часть, ограниченная маской. Маска создаётся с помощью метода объекта Graphics:

SetClip (Rectangle или GraphicsPath, CombineMode 1, 2, 3, 4, 5 значений);

Второй параметр необязателен и принимает одно из 5 значений типа CombineMode для определения, как новая вырезаемая область будет объединена с предыдущей.

Упражнение (отображение круглых изображений). Выбрать изображение из заданного графического файла 1.jpg и отобразить на форме его круглую часть, - круг, вписанный в прямоугольник с координатами левого верхнего угла (50,50) и размерами 50 x 50 пикселей.

Создадим круг на основе объекта GraphicsPath.




```
using System.Drawing.Drawing2D;
private void button1_Click(object sender, EventArgs e)
   {
    Image img = Image.FromFile("1.jpg");
    Graphics g = this.CreateGraphics();
    GraphicsPath p1 = new GraphicsPath();
   p1.AddEllipse(new Rectangle(50, 50, 50, 50));
    g.SetClip(p1);
   g.DrawImage(img, 50, 50, 50, 50);
   }
```

### 2.5. Аффинные преобразования холста и двойная буферизация

Выше, в п. 2.2, уже сказано, что масштабирование, перенос и поворот холста – объекта g класса Graphics могут быть однозначно представлены в виде преобразования вектора-столбца (x,y,1) (представляющего однородные координаты точки) -- его умножением слева на матрицу 3*3:

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & -\mathrm{dx} \\ 0 & 1 & -dy \\ 0 & 0 & 1 \end{pmatrix} \text{ или } \begin{pmatrix} kx & 0 & 0 \\ 0 & ky & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

В C# эти преобразования реализуются так:

1) создание холста g класса Graphics: Graphics g = CreateGraphics();
2) создание единичной матрицы – объекта m класса Matrix из пространства имен Drawing.Drawing2D:

```
Matrix m = new Matrix();
```

3) установка значение элементов матрицы m одним из трех способов в зависимости от запланированного действия:

а) поворот g вокруг точки (x,y) на угол alpha

```
m.RotateAt(alpha, new Point(x, y));
```

б) параллельный перенос g на dx, dy:

```
m.Translate(dx, dy);
```

в) масштабирование с коэффициентами kx, ky:

```
m.Scale(kx, ky);
```

4) установка свойства Transform объекта g в значение m:

```
g.Transform = m;
```

5) вывод изображений на g, например:

```
Rectangle r = new Rectangle(x1, y1, w, h);
g.DrawRectangle(Pens.Blue, r);
```

О двойной буферизации. Построение анимированной фигуры непосредственно на холсте вызывает мерцание экрана. Поэтому рисование сцены выполняют в памяти на «невидимом» холсте Bitmap, а законченные кадры сцены циклически выводят на «видимый» холст.

1. Создание видимого холста:

Graphics g = pictureBox1.CreateGraphics();

2. Создание невидимого холста тех же размеров:

int w=pictureBox1.Width; int h=pictureBox1.Height;
Bitmap b = new Bitmap(w, h);

Graphics bg = Graphics.FromImage(b);

Далее, в цикле выполняются следующие три пункта:

3. Очищение невидимого холста bg.Clear(Color.White); и последующее создание различных фигур на холсте на bg и выполнение преобразований над ними;

4. Вывод b на видимый холст (подчеркнем: на b, а не bg):

pictureBox1.Image = b; или g.DrawImage(b, 0, 0);

5. Завершение запланированных действий (рекомендуется!)

```
Application.DoEvents();
```

.6. Приостановка программы до визуального наблюдения:

```
Thread.Sleep(10);
```

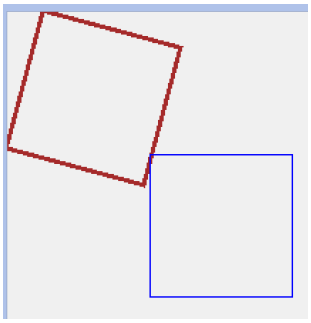
Для последнего требуется подключить пространство имен:

```
using System.Threading;
```

Подытожим сказанное. Три типа преобразования графики – масштабирование, перенос и поворот – осуществляются с использованием объекта Matrix, входящим в пространство имён System.Drawing.Drawing2D и определяющим параметры преобразования. Само преобразование осуществляется над графическим объектом Path, в который предварительно необходимо поместить нужные фигуры – прямоугольники, эллипсы, линии и изображения.

Упражнение (поворот). Нарисовать на форме прямоугольник размером 100 x 100 пикселей с координатами верхнего левого угла (10,10) коричневым пером толщиной в 3 пикселя, а затем повернуть его на 15 градусов.

```
private void button1_Click(object sender, EventArgs e)
{
 GraphicsPath path = new GraphicsPath();
 Rectangle rect = new Rectangle(10, 10, 100, 100);
        Graphics g = this.CreateGraphics();
        path.AddRectangle(rect);
        Matrix matr1 = new Matrix();
        matr1.RotateAt(15, new Point(60, 60));
        path.Transform(matr1);
        g.DrawPath(new Pen(Color.Brown, 3), path);
        g.DrawRectangle(Pens.Blue, 100, 100, 100, 100);
    }
```



Перенос осуществляется с использованием метода Translate. Аргументами данного метода являются координаты новой точки, в которую необходимо перенести левый верхний угол объекта Path.

Упражнение (перенос). Переместить прямоугольник размером 100 x 100 пикселей, с левым верхним углом с координатами (10,10) в точку с координатами (20,20).

```
GraphicsPath path = new GraphicsPath();
Rectangle rect = new Rectangle(10, 10, 100, 100);
Graphics g;
g = this.CreateGraphics();
path.AddRectangle(rect);
```

```
Matrix matr1 = new Matrix();
matr1.Translate(20, 20);
path.Transform(matr1);
g.DrawPath(new Pen(Color.Brown, 3), path);
```

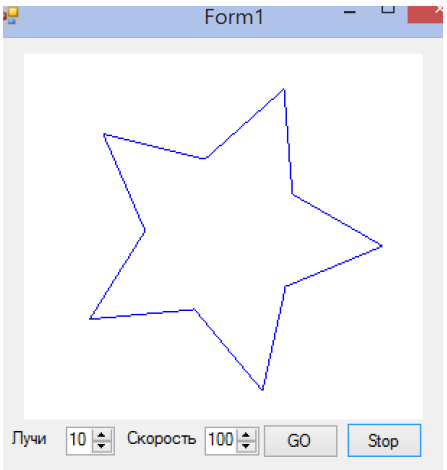
Сжатие или расширение рисунка осуществляется с помощью метода Scale объекта Matrix. Аргументами данного метода являются коэффициенты сжатия по горизонтали и вертикали. Значения меньше 1 приводят к сжатию, больше 1 – к расширению.

Упражнение (масштабирование). Сжать прямоугольник размером 100 х 100 пикселей с координатами левого верхнего угла (10,10) в два раза по горизонтали и вертикали. Для этого необходимо применить коэффициенты сжатия, равные 0.5:

```
GraphicsPath path = new GraphicsPath();
Rectangle rect = new Rectangle(10, 10, 100, 100);
Graphics g = this.CreateGraphics();
path.AddRectangle(rect);
Matrix matr1 = new Matrix();
matr1.Scale(0.5f, 0.5f);
path.Transform(matr1);
g.DrawPath(new Pen(Color.Brown, 3), path);
```

Упражнение (вращающаяся звезда). Вывести вращающуюся звезду с интерактивным управлением количеством лучей, скоростью, остановкой и стартом.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
namespace Stair
{
   public partial class Form1 : Form
   {
      int n;
      int x0, y0, r, r1;
```



```
    bool Stop = false;
    public Form1()
    {
       InitializeComponent();
       label1.Text = "Лучи";
       label2.Text = "Скорость";
       button1.Text = "GO";
       button2.Text = "Stop";
       numericUpDown1.Value = 10;
       numericUpDown2.Value = 90;
    }
   private void button1_Click(object sender, EventArgs e)
    {//инициализация вершин, центральной точки, радиусов
       Stop = false;
          n = (int)numericUpDown1.Value;
          Point[] P = new Point[n];
          // центр звезды:
          x0 = pictureBox1.Width / 2;
          y0 = pictureBox1.Height / 2;
          //внешний радиус звезды:
          r = (int)(x0 * 0.8);
          //внутренний радиус звезды
          r1 = r / 2;
          for (int i = 0; i < n; i++)
          {//Четные вершины соответствуют выступающим углам,
             //нечетные - внутренним углам
             if (i % 2 == 0)
             {
                P[i].X = (int)(x0 + r * Math.Cos(2 * Math.PI / n * i));
                P[i].Y = (int)(y0 + r * Math.Sin(2 * Math.PI / n * i));
             }
             else
             {
                P[i].X = (int)(x0 + r1 * Math.Cos(2 * Math.PI / n * i));
                P[i].Y = (int)(y0 + r1 * Math.Sin(2 * Math.PI / n * i));
             }
          }
          //при рисовании на Bitmap с переносом на pictureBox1
          //артефакты исчезают
          Bitmap b = new Bitmap(pictureBox1.Width, pictureBox1.Height);
          Matrix m = new Matrix();
//Важно!! Многократное создание матрицы в цикле замедляет работу!!
          for (int i = 0; i < 10 * 360; i++)
          {
             Graphics g1 = Graphics.FromImage(b);     //холст объекта Bitmap
```

```
            Graphics g = pictureBox1.CreateGraphics(); //холст объекта pictureBox1
            g1.Clear(Color.White);
 //очистка перед очередным выводом звезды
            m.RotateAt(1, new Point(x0, y0));        //матрица поворота на 1 градус
            g1.Transform = m;
               //де-факто поворачивается не рисунок, а система координат
            g1.DrawPolygon(Pens.Blue, P);
            //в повернутой системе координат рисуется звезда
            //pictureBox1.Image = b;  -- эквивалентно следующему оператору
            g.DrawImage(b, 0, 0);
//вывод объекта b на холст g (можно  добавить 100, 100)
            for (int k = 0; k < (100 - (int)numericUpDown2.Value) * 100000; k++) ;
            Application.DoEvents();
           //можно менять скорость вращения
            if (Stop) return;
          }
        }
     private void button2_Click(object sender, EventArgs e)
     { Stop = true; }
  }
}
```

## 2.6. Регионы. Формы произвольных очертаний

Класс Region создает регионы из прямоугольников и фигур, составленных из замкнутых линий. Регионы применяются для определения области изображения окон («вырезанные» области). Множество пикселей, входящих в состав региона, может состоять из нескольких несмежных участков. Перечислим основные методы.

Exclude - Обновляет объект Region, чтобы включить часть его внутренней части, не пересекающуюся с указанной структурой Rectangle.

FromHrgn - Инициализирует новый объект Region из дескриптора указанной существующей области GDI.

GetHrgn - Возвращает дескриптор Windows для объекта Region в указанном графическом контексте.

Intersect - Заменяет объект Region на его пересечение с указанным объектом Region.

IsEmpty - Проверяет, имеет ли объект Region пустую внутреннюю часть на указанной поверхности рисунка.

IsInfinite - Проверяет, имеет ли объект Region пустую внутреннюю часть на указанной поверхности рисунка.

IsVisible - Проверяет, содержится ли указанный прямоугольник в объекте Region.

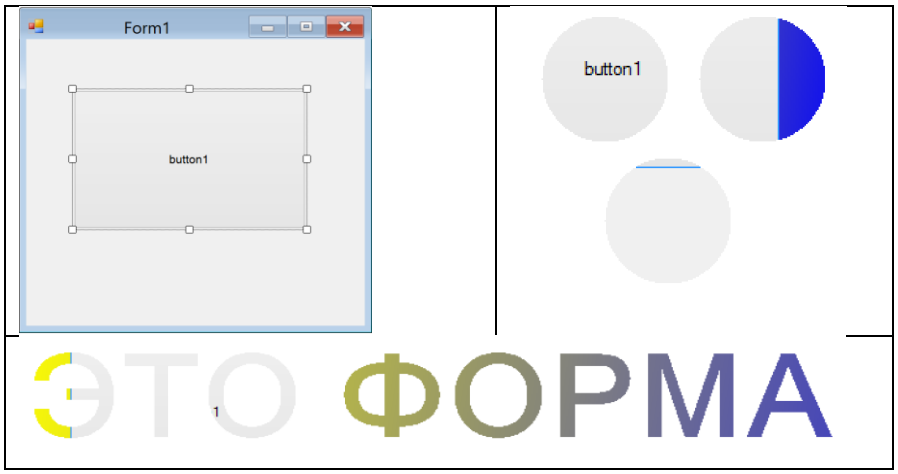
Transform - Преобразует этот объект Region с помощью указанного объекта Matrix.

Translate - Смещает координаты объекта Region на указанную величину.

Union - Заменяет объект Region на его объединение с указанным объектом GraphicsPath.

Xor - Заменяет объект Region на разность объединения и его пересечения с указанным объектом GraphicsPath.

Упражнение (форма с изменяющимися контурами). Требуется создать форму с единственной кнопкой (см. левый-верхний рисунок), чтобы при запуске форма приняла очертания букв «ЭТО ФОРМА». При щелчке кнопкой мыши на кнопку (более точно, на те ее участки, которые видны на первых буквах) форма должна принять очертания трех разрозненных кругов (см. правый-верхний рисунок).




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
//Объявления и инициализация
      const string PATH_STRING = "ЭТО ФОРМА";
```

```
      const int FONT_SIZE = 100;
      const FontStyle FONT_STYLE = FontStyle.Bold;
      SizeF stringSize;
      public Form1()
      {
         InitializeComponent();
//1. Получим оконные координаты верхней-левой точки
//   клиентской области
         Point origin = new Point(
         SystemInformation.Border3DSize.Width,
         SystemInformation.CaptionHeight+50);
//2. Создадим объект класса GraphicsPath
         GraphicsPath path = new GraphicsPath();
//и включим в него нашу строку:
         path.AddString(
            PATH_STRING,
            Font.FontFamily,
            (int) FONT_STYLE,
            FONT_SIZE,
            origin,
            StringFormat.GenericDefault
            );
//3. Установим регион для формирования
//   "буквенно-фигурной" структуры формы
          Region = new Region(path);
//4. Вычислим размеры прямоугольника, занимаемого строкой
         stringSize =
         CreateGraphics().MeasureString(PATH_STRING,
        new Font (Font.FontFamily,FONT_SIZE, FONT_STYLE)
                              );
         Width = (int) Math.Ceiling(stringSize.Width);
      }
//Конструктор формы завершен
protected override void OnPaint(PaintEventArgs e)
{//1. Создадим градиентную кисть
 LinearGradientBrush brush = new LinearGradientBrush(
 ClientRectangle, Color.Yellow, Color.Blue, 10);
 //2. Применим ее к области, созданной конструктором формы
 e.Graphics.FillRectangle(brush, 0, 0, stringSize.Width, stringSize.Height);
```

```
 }
//Обработчик события on_Paint завершен
private void button1_Click(object sender, EventArgs e)
 {
  this.Size = new System.Drawing.Size(300, 300);
  GraphicsPath path1 = new GraphicsPath();
  path1.AddEllipse(new Rectangle(110, 110, 80, 80));
  path1.AddEllipse(new Rectangle(210, 110, 80, 80));
  path1.AddEllipse(new Rectangle(150, 200, 80, 80));
 //Установим регион для формы
  this.Region = new Region(path1);
 } } }
```

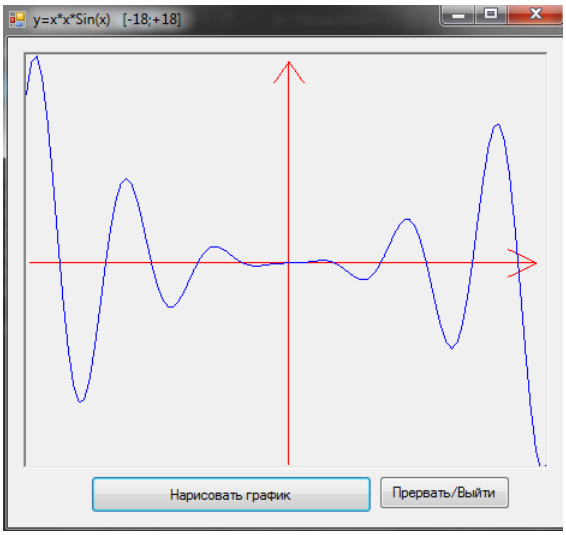
# Часть III. Графики и поверхности

## 3.1. График функции одной переменной

Для построения графика функции y = f(x) «на бумаге» выбирается прямоугольник с размерами (x1, x2)*(y1, y2), определяемыми промежутком изменения аргумента x и экстремальными значениями функции на этом промежутке. Основная проблема, возникающая при построении графиков функций на экране монитора, обусловлена необходимостью масштабировать прямоугольник (x1, x2)*(y1, y2) в прямоугольник на экране (i1, i2)*(j1, j2), а бумажные координаты (x,y) преобразовать в экранные по формулам

scX:=i1 + (x-x1)*(i2-i1)/(x2-x1);  scY:=j2 - (y-y2)*(j2-j1)/(y1-y2);

Упражнение (график функции одной переменной). Для заданной функции и заданной области определения вывести анимированный график функции.



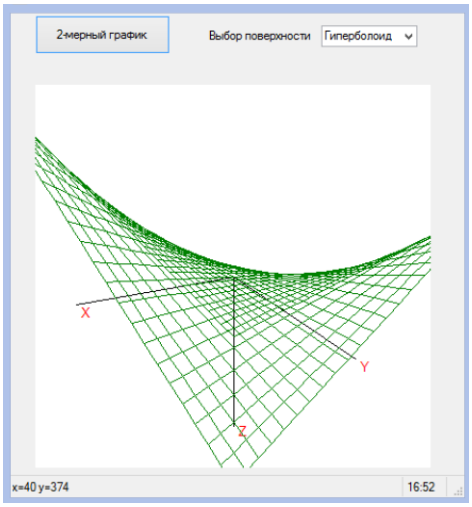

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;

namespace График_функции
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      Pen myPen;
      const int n = 100;
      const float xmin = -18f, xmax = 18, ymin = -30, ymax = 30;
      const float DeltaX = (xmax - xmin) / n;
      float koefX;
      float koefY;
    private float f(float x)
      { return (0.1f*x*x*(float)Math.Sin(x)); }
    private float scaleX (float x)
      { return ((float) (x-xmin)/(xmax-xmin)*pictureBox1.Width); }
```

```
    private float scaleY(float y)
      {
         return (
(float) ( pictureBox1.Height-(y - ymin) / (ymax - ymin) * pictureBox1.Height )
                  );
      }
    private void button1_Click(object sender, EventArgs e)
      {
         Graphics g = pictureBox1.CreateGraphics();
         float x1 = xmin, y1 = f(x1), x2, y2;
         myPen.Color = Color.Red;
         g.DrawLine(myPen, scaleX(xmin) + 3, scaleY(0.0f), scaleX(xmax - 1), scaleY(0.0f));
         //Провели ось OX
         g.DrawLine(myPen, scaleX(0.0f), scaleY(ymin + 1), scaleX(0), scaleY(ymax - 1));
         //Провели ось OY
         g.DrawLine(myPen, scaleX(xmax - 3), scaleY(0.0f - 2), scaleX(xmax - 1), scaleY(0.0f));
         g.DrawLine(myPen, scaleX(xmax - 3), scaleY(0.0f + 2), scaleX(xmax - 1), scaleY(0.0f));
         //Нарисовали острие оси OX
         g.DrawLine(myPen, scaleX(0.0f - 1), scaleY(ymax - 4), scaleX(0), scaleY(ymax - 1));
         g.DrawLine(myPen, scaleX(0.0f + 1), scaleY(ymax - 4), scaleX(0), scaleY(ymax - 1));
         //Нарисовали острие оси OY
         myPen.Color = Color.Blue;
         for (int i = 0; i < n; i++)
         {
            x2 = x1 + DeltaX;
            y2 = f(x2);
            g.DrawLine(myPen, scaleX(x1), scaleY(y1),
            scaleX(x2), scaleY(y2));
            x1 = x2;
            y1 = y2;
            Thread.Sleep(100);
            Application.DoEvents();
         }
      }
    private void Form1_Load(object sender, EventArgs e)
      {
         myPen = new Pen(Color.Blue, 1);
         koefX = pictureBox1.Width / (xmax - xmin);
         koefY = pictureBox1.Height / (ymax - ymin);
      }
    private void button2_Click(object sender, EventArgs e)
      {
         Application.Exit();
      }
   }
```

}

## 3.2. График функции двух переменных

Упражнение (график функции двух переменных). Построить график аналитически заданной функции двух переменных f(x,y). Предусмотреть: а) выбор функции из нескольких заданных, б) управление мышью.



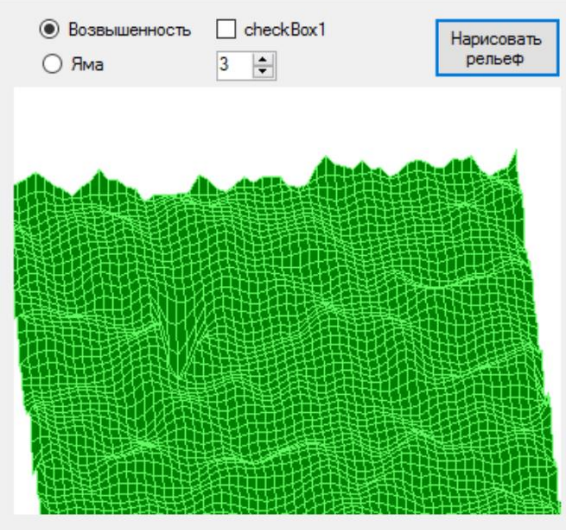

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      StatusBar statusBar1 = new StatusBar();
      private void CreateStatusLine()
      {
         StatusBarPanel panel1 = new StatusBarPanel();
         StatusBarPanel panel2 = new StatusBarPanel();
         //---------------------------------------
         panel1.ToolTipText = "Координаты мыши";
         panel2.ToolTipText = System.DateTime.Today.ToLongDateString();//
         //---------------------------------------
         panel2.Text = System.DateTime.Now.ToShortTimeString();//
         //---------------------------------------
         panel1.AutoSize = StatusBarPanelAutoSize.Spring;
//сообразно размерам формы
         panel2.AutoSize = StatusBarPanelAutoSize.Contents;
         //---------------------------------------
         statusBar1.Panels.Add(panel1);
         statusBar1.Panels.Add(panel2);
         //---------------------------------------
         statusBar1.ShowPanels = true;
         //-------------------------------
         Controls.Add(statusBar1);
      }//-----------------------------------------------------------------
      public Form1()
      {  InitializeComponent();  CreateStatusLine();  }
      bool DoDraw=false;
```

```
//включается-выключается при нажатии-отпускании кнопки мыши
      int[] x = new int [4];
      int[] y = new int [4];
      double x0 = 0.0, y0 = 0.0, z0 = 0.0,
            alpha = 3.4, beta = 0.8, a=-2.5;
      const double xMin = -1.1, xMax = 1.1, yMin = -1.1, yMax = 1.1,
      r=xMax;
      int i1, i2, j1, j2;
      const int N=10;
      double h =  (xMax-xMin)/(3*N);
      private double F1 (double x,double y)
      {
         if (comboBox1.SelectedIndex==0) return ( 2 * x * y - x - y);
         else return ( Math.Sqrt(r*r-x * x -y*y));
      }
      //Поворот на угол alpha вокруг OZ, на угол beta вокруг OX
      private void Rt(double x, double y, double z,  out int i, out int j)
      {
         double xn = (x - x0) * Math.Cos(alpha) - (y - y0) * Math.Cos(alpha);
         double yn = ((x - x0) * Math.Sin(alpha) + (y - y0) * Math.Cos(alpha)) *
Math.Cos(beta) - (z - z0) * Math.Sin(beta);
         double zn = ((x - x0) * Math.Sin(alpha) + (y - y0) * Math.Cos(alpha)) * Math.Sin(beta)
- (z - z0) * Math.Cos(beta);
         i = (int)Math.Round(pictureBox1.Width * (xn - xMin) / (xMax - xMin));
         j = (int)Math.Round(pictureBox1.Height * (yMax - yn) / (yMax - yMin));
      }
//Завершили функцию вращения точки (x,y,z) вокруг осей с последующим
//масштабированием. Следующая функция вычисляет проекцию поверхности
// на XOУ
      private void Show3D(double h)
      {
         Bitmap b = new Bitmap(pictureBox1.Width, pictureBox1.Height);
         Graphics g = Graphics.FromImage(b);
      Point[] M = new Point [4];
      g.FillRectangle(Brushes.White, -1, -1, pictureBox1.Width+1, pictureBox1.Height+1);

       Rt(0.0, 0.0, 0.0, out i1, out j1); Rt(0.9, 0.0, 0.0, out i2, out j2);
      g.DrawLine(Pens.Black, i1, j1, i2, j2);
      g.DrawString("X", new Font ("Arial",10), new SolidBrush(Color.Red), i2 + 3, j2);

      Rt(0.0, 0.7, 0.0, out i2, out j2);
      g.DrawLine(Pens.Black, i1, j1, i2, j2);
      g.DrawString("Y", new Font("Arial", 10), new SolidBrush(Color.Red), i2 + 3, j2);

      Rt(0.0, 0.0, 1.2, out i2, out j2);
```

```
    g.DrawLine(Pens.Black, i1, j1, i2, j2);
    g.DrawString("Z", new Font("Arial", 10), new SolidBrush(Color.Red), i2 + 3, j2-3);
       for (int j=-N; j<N; j++)
          for (int i=-N; i<N; i++)
          { double x1, y1;
             x1 = i * h; y1 = j * h;
        if ((x1 * x1 + y1 * y1 > r * r) && (comboBox1.SelectedIndex != 0)) continue;
        Rt(h * (i + 0), h * (j + 0), F1(h * (i + 0), h * (j + 0)), out x[0], out y[0]);
        Rt(h * (i + 0), h * (j + 1), F1(h * (i + 0), h * (j + 1)), out x[1], out y[1]);
        Rt(h * (i + 1), h * (j + 1), F1(h * (i + 1), h * (j + 1)), out x[2], out y[2]);
        Rt(h * (i + 1), h * (j + 0), F1(h * (i + 1), h * (j + 0)), out x[3], out y[3]);
            M[0].X = x[0]; M[0].Y = y[0];   M[1].X = x[1]; M[1].Y = y[1];
            M[2].X = x[2]; M[2].Y = y[2];   M[3].X = x[3]; M[3].Y = y[3];
             g.DrawPolygon(Pens.Green, M);
          }
       pictureBox1.Image = b;
  }
// завершили функцию вывода проекции сетки поверхности, повернутой и
// масштабированной
    private void button1_Click(object sender, EventArgs e)
    { Show3D(h); }
    private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
    { DoDraw = false; }
    private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
    { DoDraw = true; }//-----------------------------------------------------------
    private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
    {
       double a, b;
       statusBar1.Panels[1].Text = System.DateTime.Now.ToShortTimeString();
       statusBar1.Panels[0].Text = "x=" + e.X.ToString() + " y=" + e.Y.ToString();
       if (DoDraw)
       {
          a = e.X - pictureBox1.Width / 2; b = e.Y - pictureBox1.Height / 2;
          if (a != 0) alpha = Math.Atan2(a, b); else alpha = Math.PI / 2;
          beta = Math.Sqrt(Math.Pow(0.1 * a, 2) + Math.Pow(0.1 * b, 2));
          Show3D(h);
       }
    }
    private void Form1_Load(object sender, EventArgs e)
    {
       string [] a ={"Гиперболоид", "Часть сферы"};
       comboBox1.Items.AddRange(a); comboBox1.SelectedIndex = 0;
    }
  }}
```

### 3.3. Создание рельефа

Упражнение (создание рельефа поверхности). Нарисовать рельеф поверхности. Предусмотреть создание возвышенности (или впадины) и изменение гладкости рельефа.




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      int W, H;
      public Form1()
      {
         InitializeComponent();
         W = pictureBox1.Width;
         H = pictureBox1.Height;
      }
      //Часть 1 (объявление величин)
      //объявим размеры чертежа в "ученической тетради"
      const double xMin = -3.0, xMax = 3.0, yMin = -3.0, yMax = 3.0;
      // n - число разбиений отрезков
      const int n = 60;
      double[] x = new double[n];
      double[] y = new double[n];
      double[,] z = new double[n, n];
      const double h = (xMax - xMin) / n;
      double alpha = 0.2, beta = 0.5;
      Bitmap bmp;
      bool isDown = false;
      Random r = new Random();
      //Часть 2. Поворот и масштабирование каждой
      //узловой точки (x,y,z, alpha, beta) поверхности.
      private void RS(double x, double y, double z, double alpha,
         double beta, out int sX, out int sY)
      {
         double x1, y1, z1, x2, y2, z2;
         x1 = x * Math.Cos(alpha) - y * Math.Sin(alpha);
```

```
    y1 = x * Math.Sin(alpha) + y * Math.Cos(alpha);
    z1 = z;
    //Выполнили поворот на угол alpha вокруг оси OZ
    y2 = y1 * Math.Cos(beta) - z1 * Math.Sin(beta);
    z2 = y1 * Math.Sin(beta) + z1 * Math.Cos(beta);
    x2 = x1;

    //Выполнили поворот на угол beta вокруг оси OX
    sX = (int)((x2 - xMin) / (xMax - xMin) * W);
    sY = (int)((yMax - y2) / (yMax - yMin) * H);
}
//Часть 3. Отображение изображения
private void DrawSerfice()
{
    int[] x1 = new int[4];
    int[] y1 = new int[4];
    Pen PenGr = new Pen(Color.FromArgb(100, 250, 100), 1);
    bmp = new Bitmap(W, H);
    Graphics g = Graphics.FromImage(bmp);
    g.Clear(Color.White);
    //g.FillRectangle(Brushes.White, new Rectangle(-1, -1, W + 1, H + 1));
    for (int i = 0; i < n; i++) x[i] = xMin + i * h;
    for (int i = 0; i < n; i++) y[i] = yMin + i * h;
    Point[] P = new Point[4];
    for (int i = 1; i < n - 2; i++)
        for (int j = 1; j < n - 2; j++)
        {
            RS(x[i], y[j], z[i, j], alpha, beta, out x1[0], out y1[0]);
            RS(x[i], y[j + 1], z[i, j + 1], alpha, beta, out x1[1], out y1[1]);
            RS(x[i + 1], y[j + 1], z[i + 1, j + 1], alpha, beta, out x1[2], out y1[2]);
            RS(x[i + 1], y[j], z[i + 1, j], alpha, beta, out x1[3], out y1[3]);

            P[0].X = x1[0]; P[0].Y =  y1[0];
            P[1].X = x1[1]; P[1].Y =  y1[1];
            P[2].X = x1[2]; P[2].Y =  y1[2];
            P[3].X = x1[3]; P[3].Y =  y1[3];
            g.FillPolygon(Brushes.Green, P);

            g.DrawPolygon(PenGr, P);
        }
    pictureBox1.Image = bmp;
}
//Часть 4. Генерация и сглаживание высот
private void GenerationSmooth (int n, int Count, int i0, int j0)
{
```

```
for (int i=0; i<n; i++)
   for (int j=0; j<n;j++)
   {
      z[i, j] = 10 * r.NextDouble();
 //      if (z[i, j] < 0) MessageBox.Show
 //("i=" + i.ToString() + " j=" + j.ToString() + " z=" + z[i, j].ToString());
   }
z[i0, j0] = 100;
for (int k = 0; k < Count; k++ )
   for (int i = 1; i < n - 1; i++)
      for (int j = 1; j < n - 1; j++)
      {
            z[i, j] = (z[i - 1, j - 1] + z[i - 1, j] + z[i - 1, j + 1] +
            z[i, j - 1] + z[i, j] + z[i, j + 1] +
            z[i + 1, j - 1] + z[i + 1, j] + z[i + 1, j + 1]) / 9;

      }
      }
      private void button1_Click(object sender, EventArgs e)
      {
         alpha = 0.1; beta = 0.2;
         GenerationSmooth(n, (int) numericUpDown1.Value, n / 3, 2 * n / 3);
         DrawSerfice();
      }
      private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
      {
          double a, b;
         if (isDown)
         {
            a = e.X - pictureBox1.Width / 2; b = e.Y - pictureBox1.Height / 2;
            if (a != 0) alpha = 0.1*Math.Atan2(a, b); else alpha = Math.PI / 2;
            beta = Math.Sqrt(Math.Pow(0.001 * a, 2) + Math.Pow(0.001 * b, 2));
            DrawSerfice();
         }
      }
      private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
      {
         isDown = true;
         if (!checkBox1.Checked) return;
         int x1, y1, x2, y2;
         if (radioButton1.Checked)
         {
            alpha = 0.5; beta = 0.1;
            x1 = e.X * n / H; y1 = e.Y * n / W;
            GenerationSmooth(n, (int)numericUpDown1.Value, x1, y1);
```

```
            DrawSerfice();
        }
    }
    private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
    {
        isDown = false;
    }
  }
}
```

### 3.4. Рисование сферы

Пусть требуется вывести изображение каркасной сферы радиуса R – набор из n параллелей и меридиан. Приведем схему решения.

Шаг 1. Формулы для произвольного узла на поверхности сферы

Определим приращение угла: $da = 2 * \pi/n$ и объявим двумерные массивы x, y, z. Для вычисления узловой точки сферы, соответствующей широте b и долготе a используем формулы:

x = R *Cos(b) * Cos(a);

y = R *Cos(b) * Sin(a);

z = R *Sin(b);

Шаг 2. Поворот каждой точки вокруг двух осей на заданные углы b0 и g0

Y= y*Cos(b0) - z * Sin(b0);

Z= z*Cos(b0) + y * Sin(b0);

X = x* Cos(g0) - y* Sin(g0);

Y = y * Cos(g0) + x *Sin(g0);

Шаг 3. Отображение бумажного рисунка x1..x2 , y1..y2 на pictureBox1

Графический контейнер pictureBox1 обозначим для краткости pB1.

X’ = pB1.Width * (X - x1) / (x2 - x1);

Y’ = pB1.Height * (y2-Y) / (y2 - y1));

Шаги 1-3 выполняются для массива узлов (вложенные циклы по широтам и меридианам).

Замечание. Добавление циклов по углам b0 и g0 приведет к вращению сферы. Но перед рисованием очередной сферы необходимо выполнить стирание полотна методом Clear() или выводом прямоугольника белого цвета.

Шаг 4. Вывод каркасной сферы

Пусть координаты всех узловых точек сферы вычислены «для бумажного рисунка» и после поворотов вокруг двух осей, пересчитаны для размеров графического контейнера и сохранены. Требуется вывести каркасную сферу.

4.1. Объявим массив M из 4 точек типа Point.

4.2. Организуем новый цикл по параллелям (i) и меридианам (j) и координаты очередной пространственной ячейки занесем в точки M[0], M[1], M[2], M[3].

4.3. Видимость каждой ячейки определим по условию z>0 (произвольного угла) и для прорисовки ячейки вызовем метод

холст.DrawPolygon (перо, M);

Шаг 5. Удаление погрешностей вывода, двойная буферизация

Для удаления артефактов рекомендуется использовать для прорисовки ячеек не холст видимого графического контейнера, а холст невидимого объекта, например, объекта типа Bitmap.

После завершения рисования очередного экземпляра сферы (при организации вращения) отобразить объект Bitmap на холст pictureBox1.

Шаг 6. Элементы анимации

6.1. Для приостановки программы после каждого поворота сферы используется метод Sleep.

6.2. Для плавной деформации сферы в процессе движения используется видоизменение формул для текущей точки сферы.

6.3 Рекомендуется использовать перетекание поверхности сферы в другую поверхность, например, в поверхность тора.

Шаг 7 (дополнительный). Использование ComboBox с выбором фигур

7.1. Формулы для тора:

```
x = (R + r * Cos(b)) * Cos(a);
y = (R + r * Cos(b)) * Sin(a);
z = r *Sin(b);
```

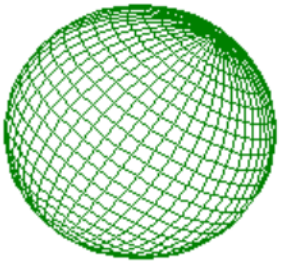
7.2. Выбор фигуры:

```
if (comboBox1.SelectedIndex != -1)  {
        cs = comboBox1.SelectedItem.ToString();
        this.Text = cs;
```

}

В зависимости от cs разветвлять вычислительный процесс.

Упражнение (сфера). Нарисовать вращающуюся каркасную сферу (несколько полных оборотов одновременно вокруг двух осей). Предусмотреть сохранение рисунка сферы по завершению вращения.



```
using System;

using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.IO;

namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      private void button1_Click(object sender, EventArgs e)
      {
         Bitmap bmp = new Bitmap(pB1.Width, pB1.Height);
         Graphics gBmp = Graphics.FromImage(bmp);

           int n = 50, R = 4, x1 = -6, x2 = 6, y1 = -6, y2 = 6;
           double da = 2 * Math.PI / n, db = da;
           double[,] x = new double[n, n];
           double[,] y = new double[n, n];
           double[,] z = new double[n, n];

           int[,] X = new int[n, n];
           int[,] Y = new int[n, n];
           Point[] M = new Point[4];
           double a, b, b0 = 0, g0=0;
           int j0=0;
           double t;
```

```
        while (j0 < 12*n)
         {
            j0++; b0 = j0 * da; g0 = 2*b0;
SolidBrush whiteBrush = new SolidBrush(Color.White);
Rectangle rect = new Rectangle(0, 0, bmp.Width, bmp.Height);
gBmp.FillRectangle(whiteBrush, rect);
            for (int j = 0; j <n; j++)
            {
               b = j * db;
               for (int i = 0; i < n; i++)
               {
                  a = i * da;
                  x[i, j] = R * Math.Cos(b) * Math.Cos(a);
                  y[i, j] = R * Math.Cos(b) * Math.Sin(a);
                  z[i, j] = R * Math.Sin(b);

                  t = y[i, j];
                  y[i, j] = t * Math.Cos(b0) - z[i, j] * Math.Sin(b0);
                  z[i, j] = z[i, j] * Math.Cos(b0) + t * Math.Sin(b0);

                  t = x[i, j];
                  x[i, j] = t * Math.Cos(g0) - y[i, j] * Math.Sin(g0);
                  y[i, j] = y[i, j] * Math.Cos(g0) + t * Math.Sin(g0);

                  X[i, j] = (int)Math.Round(pB1.Width * (x[i, j] - x1) / (x2 - x1));
                  Y[i, j] = (int)Math.Round(pB1.Height * (y2-y[i, j]) / (y2 - y1));
               }// for i
            }//for j
            for (int i = 1; i < n-1; i++)
               for (int j = 1; j < n-1; j++)
               {
                  M[0].X = X[i, j];         M[0].Y = Y[i, j];
                  M[1].X = X[i, j + 1];     M[1].Y = Y[i, j + 1];
                  M[2].X = X[i + 1, j + 1]; M[2].Y = Y[i + 1, j + 1];
                  M[3].X = X[i + 1, j];     M[3].Y = Y[i + 1, j];
                  if (z[i, j] > 0) gBmp.DrawPolygon(Pens.Green, M);
               }
            pB1.Image = bmp;
            Thread.Sleep(100);
            Application.DoEvents();
         }//while
         string path = Directory.GetCurrentDirectory() + "\\1.bmp";
         bmp.Save(path);
      }
   }
```

```
}//namespace
```

# Часть IV. Дополнительные средства. Действия с видео

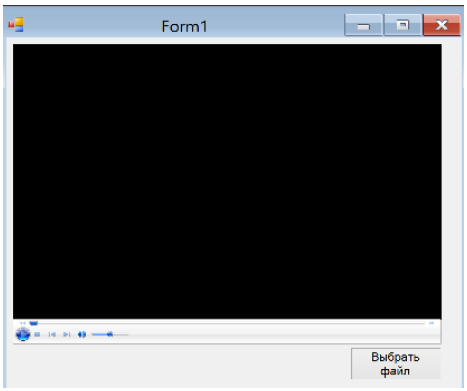
## 4.1. Использование Windows Media Player

Упражнение (воспроизведение мультимедиа). Нажатием на кнопку организовать диалог по выбору файла любых форматов, поддерживаемых WindowsMediaPlayer, и проиграть его содержимое в окне.

Решение. Поместим на форму кнопку и невидимый объект openFileDialog1, затем для размещения элемента управления WindowsMediaPlayer на форму внесем предварительно его в раздел General панели инструментов: вызовем контекстное меню раздела General панели инструментов, выберем пункт "Choose Items", в открывшемся окне для выбора компонент на вкладке COM Components поставим галочку рядом с Widows Media Player и нажмем ОК. Появившийся в разделе General появится элемент Windows Media Player перетащим на форму (или дважды щелкнем по этому элементу) и растянем его по ширине формы. Во вкладке Properties изменим имя полученного объекта на «WMP1».

В обработчике щелчка по кнопке (подпишемся на щелчок по кнопке) занесем имя выбранного файла в свойство WMP1.URL.

Если во время просмотра видео дважды щелкнуть по изображению, то проигрыватель развернется в полноэкранный режим.




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication5
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
    }
```

```
    private void button1_Click(object sender, EventArgs e)
    {
      if (openFileDialog1.ShowDialog() == DialogResult.OK)
        WMP1.URL = openFileDialog1.FileName;
    }
  }
}
```

Отступление. 1. Метод Open (Uri) открывает заданный Uri для воспроизведения мультимедиа, метод Pause() приостанавливает воспроизведение. Метод Play () воспроизводит мультимедиа с текущего значения, устанавливаемого в свойстве Position.

2. Простой подход к воспроизведению аудио- и видеофайлов – использовать VideoDrawing и MediaPlayer (другим подходом является создание собственных MediaTimeline для применения с MediaPlayer и VideoDrawing).
Пример.

```
MediaPlayer My_Player = new MediaPlayer();
MyPlayer.Open(new Uri(1.wmv", UriKind.Relative));
VideoDrawing My_aVideoDrawing = new VideoDrawing();
My_aVideoDrawing.Rect = new Rect(0, 0, 150, 150);
My_aVideoDrawing.Player = My_Player;
My_Player.Play();
```

### 4.2. Получение кадра из видеопотока

Упражнение (выхватывание одного видеокадра). Предполагается наличие одной видеокамеры (т.е. видеокамеры с номером 0). Требуется разместить на форме контейнер pictureBox1 и кнопка, при нажатии на которую из видеопотока выхватывается один-единственный кадр и отображается в pictureBox1.

Прежде всего убедимся, что на C: размещен каталог C#DLL, в который скопированы файлы из x64 – папки для поддержки библиотеки OpenCV, при этом путь C:\C#DLL должен быть указан в качестве значения переменной среды Path.

Создадим проект C# вида Windows Forms и добавим в список пространств имен по пункты

```
using System.Runtime.InteropServices;
using System.Threading;
using System.IO;
using OpenCvSharp;
using OpenCvSharp.Extensions;
```

Обратим внимание на то обстоятельство, что названия двух последних пространств имен, относящихся к библиотеке OpenCV, подчеркиваются системой красной волнистой линией. Это объясняется тем, что не установлены ссылки на динамические загружаемые библиотеки из папки Net40. Скопируем ее также в корневой каталог C:\, «рядом» с папкой C#DLL.

Нажмем правой кнопкой мыши на пункт References и выберем из контекстного меню пункт «Добавить ссылку». В результате откроется окно, где выберем кнопку «ОБЗОР» внизу окна. Распахнется окно диалога, где найдем нашу папку C:\C#net40 и откроем ее. Выберем все пять отображенные в окне dll и нажмем на кнопку ДОБАВИТЬ. В окне Менеджера ссылок заметим, что все пять dll внесены в список и выделены галочками. Остается подтвердить нажатием кнопки ОК. В результате исчезнет подчеркивание последних двух пространств имен.

Упражнение (загрузка одного кадра). В предположении, что описанные выше действия выполнены, программе требуется загрузить один видеокадр в pictureBox1.

```
using System.Runtime.InteropServices;
using System.Threading;
using System.IO;
using OpenCvSharp;
using OpenCvSharp.Extensions;

namespace WindowsFormsApplication1
{
  public partial class Form1 : Form
  {
    public Form1()
    { InitializeComponent();  }
    private void button1_Click(object sender, EventArgs e)
    {
      Bitmap bmp = new Bitmap(pictureBox1.Width, pictureBox1.Height);
          CvCapture cap = CvCapture.FromCamera(0);
          IplImage img;
              img = cap.QueryFrame();
              bmp = img.ToBitmap();
              pictureBox1.Image = bmp;
              Application.DoEvents();
       }
  }
}
```

Из библиотеки OpenCV используется класс CvCapture: создается объект cap этого класса и вызовом метода FromCamera инициализируется камера с номером 0:

```
CvCapture cap; cap = CvCapture.FromCamera(0);
```

затем создается объект img класса IplImage (чтобы избежать путаницы близких по начертанию букв, приведем запись строчными буквами: iplimage) и инициализируется текущим кадром, выхватываемым из видеопотока методом QueryFrame() класса CvCapture:

```
IplImage img; img = cap.QueryFrame();
```

важно подчеркнуть, что далее нет надобности прибегать к средствам библиотеки OpenCV. Выполняется преобразование изображения кадра img в «понятный» языку C# тип Bitmap:

```
bmp=img.ToBitmap();
```

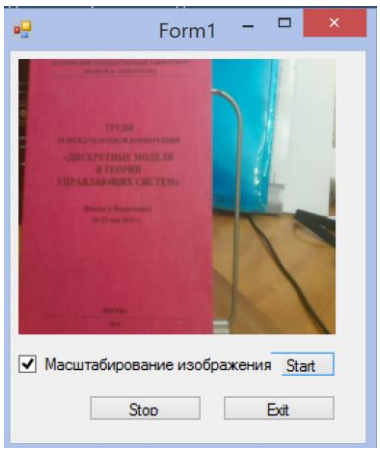
наконец, картинка загружается из переменной bmp в контейнер pictureBox1:

```
pictureBox1.Image = bmp;
```

Здесь, как и в других аналогичных случаях, рекомендуется приостановить выполнение последующих действий программы до завершения выполнения всех предписанных выше действий: Application.DoEvents().

Упражнение (загрузка кадров). Загрузить последовательные видеокадры в pictureBox1.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using System.Runtime.InteropServices;
using System.Threading;
using System.IO;
using OpenCvSharp;
using OpenCvSharp.Extensions;


namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
```



```
    public Form1()
    { InitializeComponent(); }
    bool Stop = true;
    private void button1_Click(object sender, EventArgs e)
    {
      if (checkBox1.Checked)
        pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
        Bitmap bmp1 = new Bitmap(pictureBox1.Width, pictureBox1.Height);
        IplImage img1;

        CvCapture cap1 = CvCapture.FromCamera(0);
        while (Stop)
        {
          img1 = cap1.QueryFrame();
          bmp1 = img1.ToBitmap();

          pictureBox1.Image = bmp1;
          Application.DoEvents();
        }
      }
    private void button2_Click(object sender, EventArgs e)
    {  Stop = true; }
    private void button3_Click(object sender, EventArgs e)
    {   Application.DoEvents();  Application.Exit();  }
    }
}
```

### 4.3. Отслеживание выбранной уникальной цветной точки в видеопотоке

Если программа распознает направление движения некоторого объекта, то несложно сопоставить движению влево, например, перемещение по слайдам презентации по убыванию номеров (связав с движением влево нажатие клавиши «Влево»).

Легко также предусмотреть одновременное манипулирование двумя цветными пятнами, нанеся их на обе ладони демонстратора и программным путем отслеживания не только направление движения, но и взаимное движение – сближение, удаление – сопоставляя этим перемещениям сложные манипуляции. Например, увеличение или уменьшение размеров картины.

Пусть в пределах видимости веб-камеры размещено уникальное для видимой области цветное пятно любой природы: например, кружочек цветной бумаги приклеен на ладони пользователя или веб-камере показывают ручку, у которой область пера выделена уникальным цветом, отличным от фона. Например, ручка - красного цвета, а поблизости от острия пера нанесена заметная точка

синего цвета – идентифицирующее пятно. Как будет отслеживаться движение данного пятна, если предположить на минуту, что это цвет, «строго» отличный от цветов всех видимых веб-камерой точек? Пусть даже при дополнительном предположении, что это – чисто синий цвет? Понятно, что чистый синий цвет (когда в представлении модели RGB красная и зеленая составляющие равны 0, а синяя составляющая равна 255) едва ли можно реализовать на практике. С другой стороны, если же значения реального цвета идентифицирующего пятна задать приблизительно, то трудно оценить последствия такого подхода. Следовательно, надо попытаться узнать, что сама программа «думает» о цвете данного пятна, т.е. вычислить цвет программным путем.

Во-первых, как «объяснить» веб-камере, какое из пятен окружения ей следует отслеживать? Напомним, что мы уже научились отображать выхватываемые из видеопотока кадры в графическом контейнере pictureBox1. Следовательно, указать на цветную точку (пятно) можно наведением курсора мыши на нее и щелкнув кнопкой мыши. В обработчике события pictureBox1_MouseDown узнаем цвет точки, вычленим его составляющие (красную, зеленую, синюю) и запомним их в глобальных переменных r0, g0 и b0 соответственно:

```
private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
    {
      Color CC;
      CC = (pictureBox1.Image as Bitmap).GetPixel(e.X, e.Y);
      r0 = CC.R;
      g0 = CC.G;
      b0 = CC.B;
```

Как и выше, организуем цикл вывода кадров видеопотока в pictureBox1, выхватывая очередной кадр в объект img, формат которого затем преобразуется в bitmap, после чего кадр помещается в pictureBox1:

```
IplImage img = cap.QueryFrame();
bmp = img.ToBitmap();
pictureBox1.Image = bmp;
```

Мы будем сканировать каждый выхватываемый из видеопотока кадр bmp построчно сверху-вниз, двигаясь в каждой строке i слева направо (j). В каждой точке вычисляем значения текущих цветовых составляющих r, g и b,

```
Color C = bmp.GetPixel(i, j);
r = C.R; g = C.G; b = C.B;
```

сравнивая каждое из них с запомненными в переменных r0, g0 и b0 соответственно с целью узнать: является ли данная точка отслеживаемой точкой (пятном, выбранным выше) или нет.

Здесь возникает коллизия следующего характера. В действительности достижение равенств типа r=r0 вряд ли возможно. Другими словами, целесообразно заменить точные равенства проверкой с некоторой ошибкой d: считать, что искомая точка в кадре обнаружена, если

```
(Math.Abs(r0 - r) < d) && (Math.Abs(g0 - g) < d) &&
(Math.Abs(b0 - b) < d).
```

Как выбрать значение d? Целесообразно в каждой итерации присваивать d значение из компоненты numericDownUp1, с помощью которой можно подобрать приемлемое значение d эмпирическим путем: увеличивая и уменьшая numericDownUp1.Value, пока не добьемся соответствия точки, признаваемой программой за искомую («прицел программы»), точке, выбранной нами («реальная мишень»).

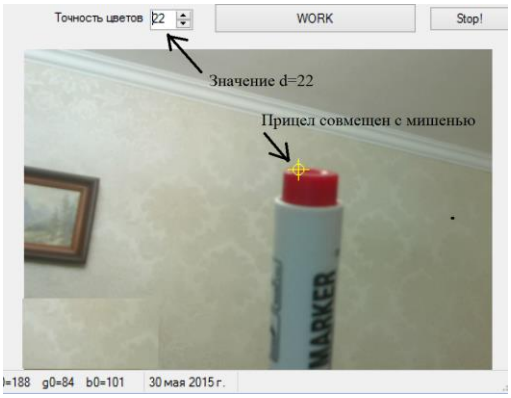
```
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
  {
    d = (byte) numericUpDown1.Value;
  }
```

Для визуальной проверки последнего предлагается выводить кружочек в точке «прицела программы» и изменять значение numericDownUp1.Value до совмещения «прицела программы» с «реальной мишенью».

```
gpB1.DrawEllipse (Pens.Yellow, i-radius, j-radius, 2*radius, 2*radius);
gpB1.DrawLine (Pens.Yellow, i-2*radius, j, i+2*radius, j);
gpB1.DrawLine (Pens.Yellow, i, j+2*radius, i, j-2*radius);
```

Упражнение (отслеживание точки). В начале в pictureBox1 загрузить видеокадр и щелчком кнопки мыши указать в нем уникальное по цвету пятно малых размеров (например, синий конец карандаша) и запомнить его цвет. Затем пятно можно перемещать перед видеокамерой. Последовательно выхватывая видеокадры в pictureBox1 и перебирая пиксели изображения, программа должна каждый раз обнаруживать искомое пятно и показывать на него, например, рисованием «прицела».

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
```




```
using System.Windows.Forms;

using System.Threading.Tasks;
using System.Runtime.InteropServices;
using OpenCvSharp;
using OpenCvSharp.Extensions;
using System.IO;
using System.Threading;

namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         StatusBarPanel panel1 = new StatusBarPanel();
         StatusBarPanel panel2 = new StatusBarPanel();

         panel1.Text = "r0=" + r0.ToString() + " g0=" +
            g0.ToString() + " b0=" + b0.ToString();
         panel1.AutoSize = StatusBarPanelAutoSize.Contents;
         panel2.ToolTipText = "Started: " +
            System.DateTime.Now.ToShortTimeString();
         panel2.Text = System.DateTime.Today.ToLongDateString();
         panel2.AutoSize = StatusBarPanelAutoSize.Contents;

         statusBar1.ShowPanels = true;

         statusBar1.Panels.Add(panel1);
         statusBar1.Panels.Add(panel2);

         this.Controls.Add(statusBar1);
      }
      //---------------------------------------------------------------------
      Byte r, g, b, r0 = 0, g0 = 0, b0 = 0, d = 20;
      bool getRGB = false;
      bool stop = false;
      StatusBar statusBar1 = new StatusBar();
      CvCapture cap = CvCapture.FromCamera(1);

      private void button1_Click(object sender, EventArgs e)
      {
         button1.Text = "Get Target!";
         Application.DoEvents();
```

```
    Graphics  gpB1 = pictureBox1.CreateGraphics();
    Bitmap bmp = new Bitmap(pictureBox1.Width, pictureBox1.Height);
    var key = CvWindow.WaitKey(10);
  V: while (key != 27) // цикл до нажатия клавиши ESC
    {
       if (stop) //или нажатия кнопки Button2 с надписью STOP
       {
          Thread.Sleep(200);
          this.Close();
       }

       key = CvWindow.WaitKey(5);
       IplImage img = cap.QueryFrame(); //выхваченный кадр
       bmp = img.ToBitmap();          //преобразовали в Bitmap
       pictureBox1.Image = bmp;       //и выставили в pictureBox1
      Application.DoEvents();
      if (!getRGB) continue;
      for (int j = 0; j < bmp.Height; j += 3) //при переборе пикселей bmp
      for (int i = 0; i < bmp.Width; i+=3)
      //в поисках цветного пятна с цветовыми составляющими r0, g0, b0
//пропускаем по две линии из последовательных трех для повышения скорости
         {
            Color C = bmp.GetPixel(i, j); //цвет текущего пикселя изображения
            int radius = 5;
            r = C.R; g = C.G; b = C.B;       // разложили на составляющие
if ((Math.Abs(r0 - r) < d) && (Math.Abs(g0 - g) < d) && (Math.Abs(b0 - b) < d))
               //если цвет "с точностью до d" совпадает с искомым,
               //то нарисуем прицел на найденной отслеживаемой точке:
            {
               gpB1.DrawEllipse(Pens.Black, i-radius, j-radius, 2*radius, 2*radius);
               gpB1.DrawLine(Pens.Black, i-2*radius, j, i+2*radius, j);
               gpB1.DrawLine(Pens.Black, i, j+2*radius, i, j-2*radius);
               Thread.Sleep(30);
               goto V;
            }
         }
    }
   return;
   //Invalidate();
   //Application.Exit();
  }
  private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
  {
  //При инициализации программы нажатием Button1
  //нажатием левой кнопки мыши укажем искомое цветное пятно на
```

```
//начальном кадре, загруженном в pictureBox1;
//цветовые составляющие точки запомним в глобальных
//переменных r0, g0, b0.
//факт "целеуказания" запомним в глобальной переменной getRGB
   Color CC;
CC = (pictureBox1.Image as Bitmap).GetPixel(e.X, e.Y);
r0 = CC.R; g0 = CC.G; b0 = CC.B;
this.statusBar1.Panels[0].Text = "   r0=" + r0.ToString() +
                                 "    g0=" + g0.ToString() + "    b0=" + b0.ToString();
Thread.Sleep(100);
Application.DoEvents();
getRGB=true;
button1.Text = "WORK";
this.Text = "       ПРОЦЕСС ПОШЕЛ!";
Application.DoEvents();
}
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{
   d = (byte) numericUpDown1.Value;
}
private void button2_Click(object sender, EventArgs e)
{
   stop = true;
}
}
```

# Список литературы

1. Албахари Джозеф, Албахари Бен. C# 5.0. Справочник. Полное описание языка.: Пер. с англ. – М.: ООО «И. Д. Вильямс», 2014. – 1008 с.

2. Мак-Дональд Мэтью. WPF 4: Windows Presentation Foundation в .NET 4.0 с примерами на C# 2010 для профессионалов.: Пер. с анг. – М. : ООО «И.Д. Вильямс», 2011. – 1024 с.

3. OpenCV шаг за шагом [электронный ресурс] // URL: http://robocraft.ru/ blog/computervision/420.html (дата доступа: 26.05.2016).

4. Магомедов А.М., Магомедов М.А. Веб-камеры в проектах Delphi (уч.-мет. пособие) // Изд-во ООО "Радуга-1", Махачкала, 2014. – 38 с.

5. Н.Б. Культин. Microsoft Visual C# в задачах и примерах: БХВ-Петербург, 2007. – 241 с.

6. М. Фленов. Библия C#: БХВ-Петербург, 2011. – 560 с.