Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тульский государственный университет»

Институт прикладной математики и компьютерных наук
Кафедра «Вычислительная техника»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К КУРСОВОЙ РАБОТЕ

по дисциплине

Векторная графика

Уровень профессионального образования: *(высшее образование – бакалавриат.)*
Направление подготовки*: 09.03.01 «Информатика и вычислительная техника»*

Профиль подготовки: *Системы автоматизированного проектирования*

Квалификация выпускника: 62 - *бакалавр*

Форма обучения: *очная, очно-заочная, заочная, заочная сокращенная*

Тула 2016 г.

Методические указания к курсовой работе составлены *к.т.н. доц. Ю.В. Французовой* и обсуждены на заседании кафедры *вычислительной техники,* института *прикладной математики и компьютерных наук*
протокол № 1 от " 29 " августа 2016 г.
Зав. кафедрой_____А.Н. Ивутин

Цели работы

1. Познакомиться с таким понятием, как векторная графика, основными ее достоинствами и недостатками.

2. Изучить возможности технологии GDI+ для создания статических и динамических векторных изображений.

3. Разработать программу на языке C#.

Теоретический материал

Векторная графика – это способ представления изображений, основанный на использовании элементарных геометрических объектов: точек, линий, сплайнов и многоугольников.

Эту графику часто называют объектно-ориентированной, так как ее основе лежат примитивные объекты, такие как окружности, линии, сферы, кубы и тому подобное, и на их основе создаются более сложные объекты, образуя своего рода иерархическое дерево.

Любая векторная иллюстрация, то есть сама картинка, – это верхний уровень иерархии.

Ниже идут объекты, представляющие собой разнообразные векторные формы.

Каждый из этих объектов, в свою очередь, состоит из открытых или замкнутых контуров. Контуром в данном случае называется кривая, представляющая собой очертания некоторого графического объекта. Если начальная и конечная точки кривой совпадают, то такой контур называется замкнутым. Если же контур имеет четко обозначенные концевые точки, то он называется открытым.

Каждый контур состоит из сегментов, занимающих в контуре определенное положение, фиксируемое опорными точками или узлами – началом и концом сегмента. Замкнутые контуры могут иметь свойство заливки – цвета или узора, выводимого в замкнутой области, ограниченной кривой.

Самый нижний уровень иерархии образуют основные элементы векторных изображений: узлы и отрезки линий, их соединяющие.

Таким образом, упрощенную иерархию векторного изображения можно представить следующим образом (рис. 1).

Рис. 1. Иерархия векторного изображения

Информация о любом объекте в векторной графике хранится в описательной форме. Например, окружность будет храниться следующим образом:

- число, соответствующее координате x центра окружности;
- число, соответствующее координате y центра окружности;
- величина радиуса окружности;
- цвет и толщина контура;
- цвет и характер заливки (при ее наличии).

Векторная графика широко используется в рекламных и дизайнерских агентствах, редакциях и издательствах, так как выполняемые ими оформительские работы основаны на применении шрифтов и простейших геометрических элементов. Применение именно векторной графики позволяет более эффективно решать задачи подобного рода.

GDI (GraphicsDeviceInterface, GraphicalDeviceInterface) является интерфейсом Windows, предназначенным для представления графических объектов и вывод их на монитор или принтер. Он отвечает за отрисовку линий, кривых, отображение шрифтов и обработку палитры и обеспечивает унификацию работы с различными устройствами вместо прямого доступа к оборудованию, что позволяет одними и теми же функциями рисовать на различных устройствах, получая при этом практически одинаковые изображения.

Данная технология предоставляет богатые возможности для работы с векторной графикой. Основной ее элемент (линия) имеет такие атрибуты, как толщина, рисунок пунктира, цвет, которые собраны в одном объекте. Кроме того, существуют различные способы заливки областей,

также собранные в одном объекте, и поддержка матрицы поворотов и растяжений изображений векторной графики.

На базе технологии GDI была разработана GDI+. Это улучшенная среда для 2D-графики, расширенная возможностями сглаживания линий, использования координат с плавающей точкой, градиентной заливки, использованием ARGB-цветов и т. п. Рассмотрим основные возможности для работы с этой технологией, предоставляемые C#.

В рамках данной работы мы освоим технику рисования картинок в оперативной памяти с последующим выводом их на форму. Для рисования мы будем использовать объект класса Bitmap, а для вывода на форму – элемент управления PictureBox.

Рассмотрим написание простейшего приложения, когда по нажатию на кнопку на форме будет отображаться красный кружок. Для этого создаем новый проект типа WindowsFormsApplication и размещаем на форме PictureBox и Button. Переходим к методу обработки нажатия на кнопку. В нем нам нужно, во-первых, создать объект класса Bitmap, с размерами, совпадающими с нашим PictureBox. В данном случае Bitmapвыполняет роль холста, а PictureBox – рамки, в которую этот холст повесят. Поэтому размеры должны совпадать. В итоге получаем код:

```
Bitmapbmp = newBitmap (pictureBox1.Width, pictureBox1.Height);
```

Теперь мы должны создать объект класса Graphics – основного класса, предоставляющего доступ к возможностям GDI+. Для данного класса не определено ни одного конструктора. Его объект создается в ходе выполнения ряда методов применительно к конкретным объектам, у которых есть поверхность для рисования. Одним из таких объектов и является Bitmap. Поэтому создаем объект класса Graphics следующим образом:

```
Graphics gr = Graphics.FromImage(bmp);
```

Теперь все вызовы методов отображения фигур будут отрабатывать на нашей битовой карте. Класс Graphics содержит множество методов рисования вида Fill* или Draw*, отвечающих за отображение закрашенных или не закрашенных фигур. Первая группа методов в качестве одного из параметров принимает объект типа Brush (кисть), а вторая – объект типа Pen (карандаш). Исключение – метод DrawString, который отображает текст. Этот метод в качестве одного из параметров принимает объект Brush. Для того чтобы отобразить красный круг, как мы задумали ранее, необходимо написать следующий код:

```
gr.FillEllipse(Brushes.Red,10,10,40,40);
```

Обратите внимание, что координаты 10,10 – это не координаты центра эллипса , а координаты левого верхнего угла прямоугольника, в который будет вписан эллипс. Brushes.Red – стандартная красная кисточка. 40,40 – высота и ширина эллипса. Если вы сейчас запустите приложение и нажмете на кнопку, то ничего не произойдет. Вернее, эллипс будет нарисован, но

мы его не увидим, так как сейчас он находится в оперативной памяти. Для того чтобы отобразить нарисованную картинку на форме, необходимо добавить следующую строчку кода:

pictureBox1.Image = bmp;

В итоге на экране вы должны увидеть примерно следующее(рис. 2):

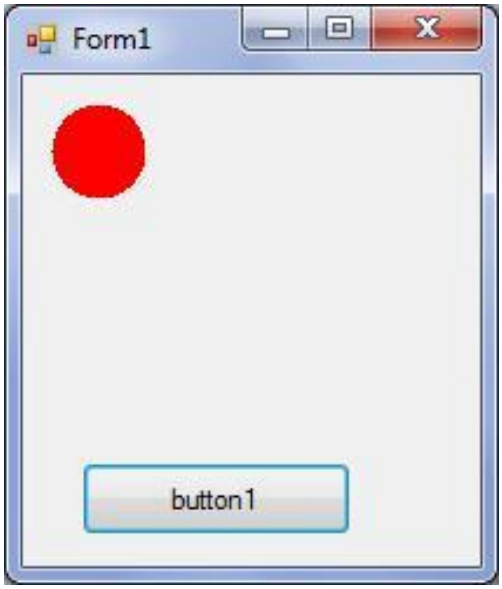

Рис. 2. Красный круг

Рисование в технологии GDI+ осуществляется слоями, которые перекрывают друг друга, поэтому если мы напишем вот такой код:

gr.FillEllipse(Brushes.Red,10,10,40,40); gr.FillEllipse(Brushes.Green,10,10,40,40); ,то на форме увидим только зеленый круг.

Теперь рассмотрим способы создания различных кисточек. Во-первых, создадим кисточку случайного цвета. Для этого напишем следующий код:

Randomr=newRandom();

SolidBrushsb=newSolidBrush(Color.FromArgb(r.Next(0,255), r.Next(0,255),

r.Next(0,255), r.Next(0,255)));

Метод FromArgb принимает в качестве параметров 4 числа, кодирующих цвет в системе ARGB: A (альфа) – прозрачность; R (red) – красный цвет; G (green) – зеленый цвет; B (blue) – синий цвет.Для создания кисточек с более интересными характеристиками необходимо подключить namespace System.Drawing.Drawing2D. Рассмотрим создание градиентной кисти. Напишемследующийкод:

LinearGradientBrushgradientBrush = new LinearGradientBrush(new Point(0, 0),new Point(40, 40), Color.Green, Color.Blue);

Как мы знаем, линейная градиентная заливка – это вид заливки,которой необходимо задать цвет и координаты двух ключевых точек, расположенных на одной прямой, а цвет точек между

ними рассчитывается относительно них по определенным математическим алгоритмам. В итоге получают плавные переходы из одного цвета в другой.

Следующая кисть, которую мы рассмотрим, – штриховая. Создается она следующим образом:

HatchBrushhatchBrush = newHatchBrush(HatchStyle.Cross,

Color.Red, Color.Yellow);

Первый параметр конструктора задает вид штриховки. Это системное перечисление с множеством вариантов. Второй параметр отвечает за цвет штрихов, а третий – цвет фона.

При использовании этой кисти в рисовании нашего круга, мы получим следующий результат (рис. 3):

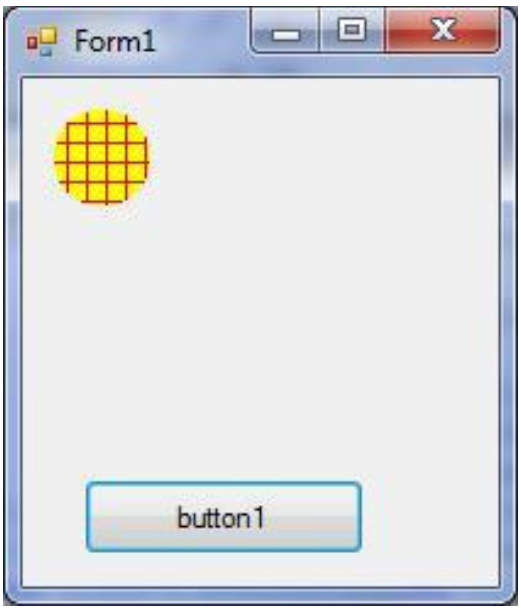

Рис. 3. Штрихованный круг

Теперь рассмотрим создание карандашей. Их можно создавать на основе цвета или на основе ранее созданной кисти. Можно указывать толщину линии, а также форму ее конечной точки (только для незамкнутых контуров) и прочее. Ниже приведен код для создания различных карандашей:

Pen p1=newPen(Color.Red); // Обычное красное перо

Pen p2=new Pen(Color.Green, 4); // Ширина 4 пикселя

p2.EndCap = LineCap.ArrowAnchor; //Стрелочканаконце

Pen p3=new Pen(gradientBrush, 6); // Градиент

Pen p4=new Pen(hatchBrush, 6); // Штриховка

Для вывода текста необходимо использовать метод DrawString, принимающий в качестве параметров строку, которую нужно вывести, объект класса Font, кисть и координаты вывода. Конструктор класса Font, в свою очередь, принимает название и размер шрифта. Приведемкодвыводанадписи «Hello, world!»:

gr.DrawString(“Hello, world!”, new Fonr(“Arial”,15),hatchBrush,20,20);

Мы рассмотрели методы вывода различных векторных примитивов. Теперь рассмотрим методы их преобразования:

- перенос;
- поворот;
- масштабирование;
- сдвиг.

Данные методы широко применяются в ситуациях, когда вам необходимо изменить размер, положение или форму фрагмента рисунка, состоящего из большого числа объектов. Перечисленные выше методы применяют преобразования сразу ко всем отображаемым объектам, что экономит вам время.

Перенос – это линейное перемещение объекта, при котором размер и ориентация не меняются. Напишем код отрисовки красного обведенного контуром прямоугольника, затем вызовем метод переноса и повторим код:

```
gr.FillRectangle(Brushes.Red, 10, 10, 20, 40);
gr.DrawRectangle(p2, 10, 10, 20, 40);
gr.TranslateTransform(20, 20);
gr.FillRectangle(Brushes.Red, 10, 10, 20, 40);
gr.DrawRectangle(p2, 10, 10, 20, 40);
```

В итоге на экране вы должны увидеть примерно следующее(рис. 4):

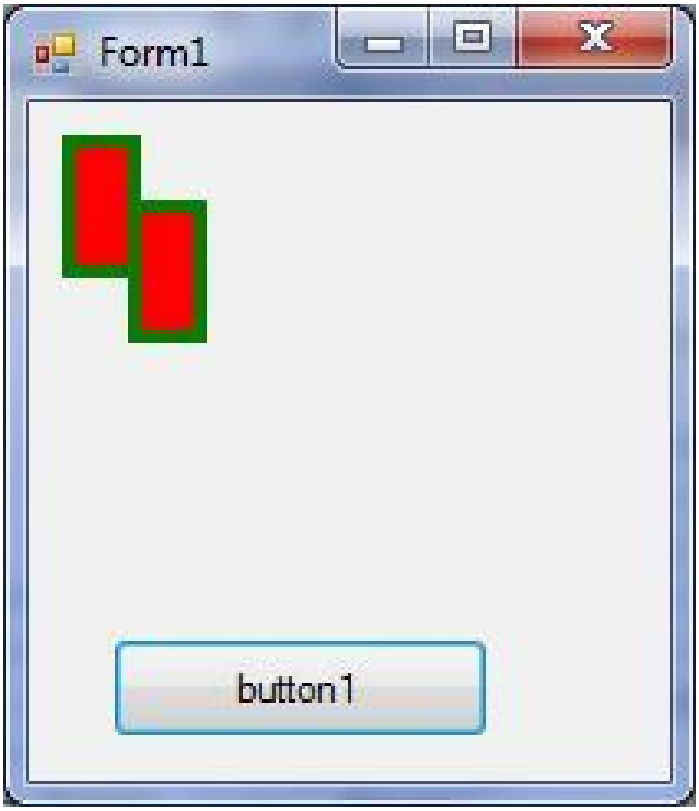

Рис. 4. Трансформация «Перенос»

Метод TranslateTransform принимает в качестве параметров величины переноса по обеим осям. По сути он сдвигает начало координат для отрисовки объекта.

Поворот – это изменение угла отрисовки объекта относительно начала координат. Величина поворота задается в радианах. Код данной трансформации следующий:

```
gr.FillRectangle(Brushes.Red,50, 50, 20, 40);
gr.DrawRectangle(p2, 50, 50, 20, 40);
gr.RotateTransform(20);
gr.FillRectangle(Brushes.Red, 50, 50, 20, 40);
gr.DrawRectangle(p2, 50, 50, 20, 40);
```

Обратите внимание, что поворот выполняется не относительно оси объекта, а относительно начала координат. В итоге у вас на форме должно появиться следующее изображение (рис. 5):

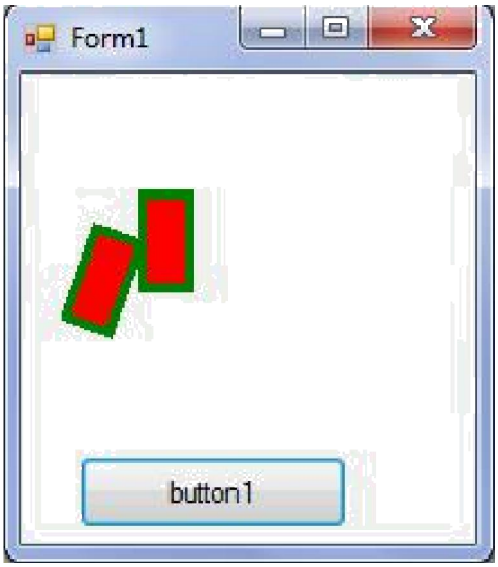

Рис. 5. Трансформация «Поворот»

Масштабирование – это изменение размеров объекта по обеим осям. Увеличим наш прямоугольник в два раза в высоту и ширину:

```
gr.FillRectangle(Brushes.Red,10, 10, 20, 40);
gr.DrawRectangle(p2, 10, 10, 20, 40);
gr.ScaleTransform(2,2);
gr.FillRectangle(Brushes.Red, 10, 10, 20, 40);
gr.DrawRectangle(p2, 10, 10, 20, 40);
```

В итоге у вас на форме должно появиться следующее (рис. 6):

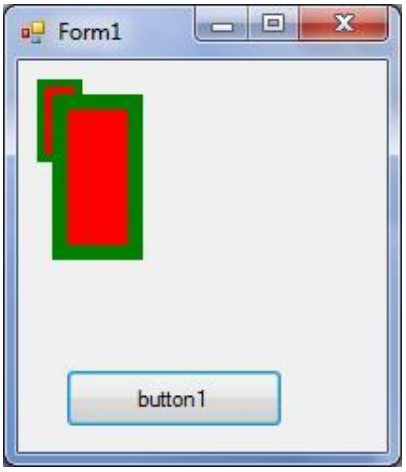

Рис. 6. Трансформация «Масштабирование»

Сдвиг – практически произвольное искажение объекта. Для него не существует какого -то отдельного метода. В данном случае необходимо напрямую задавать матрицу трансформации:

```
gr.FillRectangle(Brushes.Red,10, 10, 20, 40);
gr.DrawRectangle(p2, 10, 10, 20, 40);
Matrix matrix = new Matrix();
matrix.Shear(0.5f, 0.25f);
gr.Transform = matrix;
gr.FillRectangle(Brushes.Red, 10, 10, 20, 40);
gr.DrawRectangle(p2, 10, 10, 20, 40);
```

В итоге на экране должна получиться следующая картинка (рис. 7):

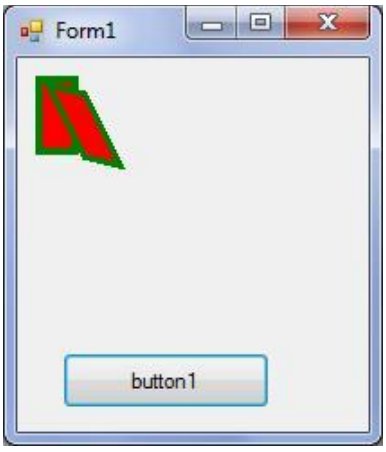

Рис. 7. Трансформация «Сдвиг»

Нужно заметить, что все рассмотренные трансформации основываются на преобразовании матрицы координат. И при вызове нескольких методов подряд их эффекты будут складываться. Для того чтобы вернуть матрицу координат в исходное состояние, необходимо вызвать метод ResetTransform.

Задание на курсовую работу

№ варианта	Объект для первого режима	Направление движения для второго режима	Первый фоновый объект для второго режима	Второй фоновый объект для второго режима
1.	Рыбка	Слева направо	Куст водорослей	Груда камней
2.	Паровоз	Справа налево	Дом	Дерево
3.	Грузовик	По диагонали слева направо	Дерево	Знак «Главная дорога»
4.	Пароход	Слева направо	Айсберг	Остров с деревом
5.	Легковой автомобиль	Слева направо	Лиственное дерево	Ель
6.	Парашютист	Сверху вниз	Солнце	Облако
7.	Ракета	Снизу вверх	Воздушный шар	Облако
8.	Парусник	Слева направо	Облако	Айсберг
9.	Самолет	По диагонали снизу вверх	Дом	Облако
10.	Летающая тарелка	Сверху вниз	Луна	Дом