

---

# Визуализация графических данных средствами GDI+

---

---

## Что такое GDI+

Платформа .NET обеспечивает целый набор пространств имен для поддержки визуализации двумерной графики.

Эти пространства имен формируют тот набор возможностей .NET, который называется GDI+ (Graphics Device Interface — интерфейс графических устройств, интерфейс GDI) и который является управляемой альтернативой Win32 GDI API (Application Programming Interface — программный интерфейс приложения).

---

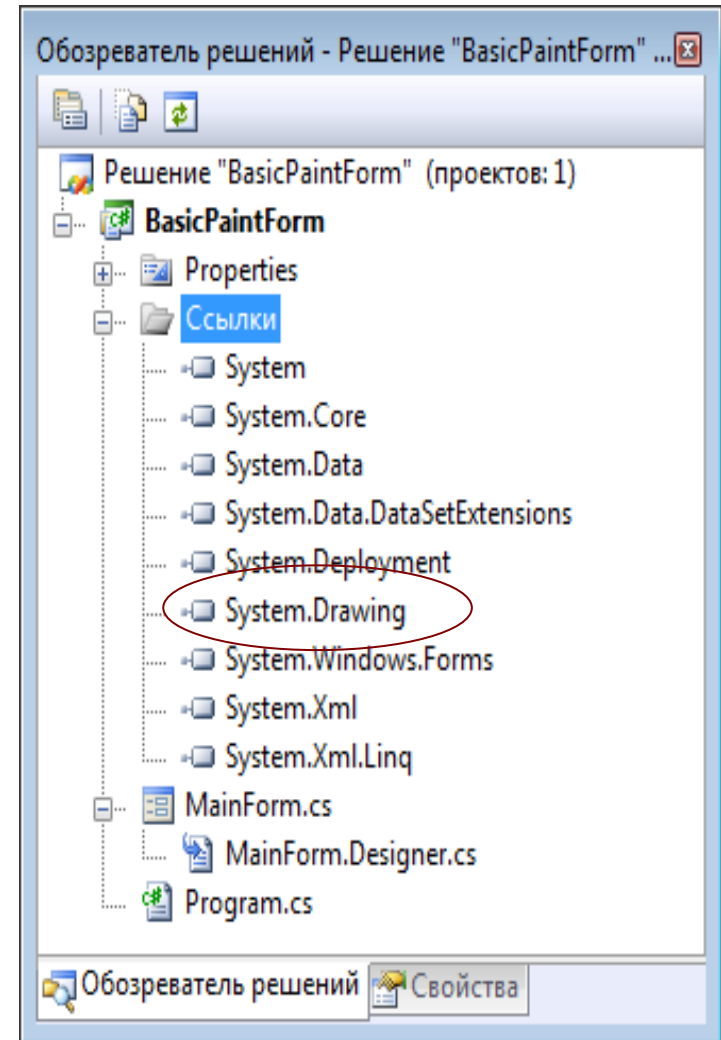
## Обзор пространств имен GDI+

Пространство имен	Описание
<b>System.Drawing</b>	Базовое пространство имен GDI+, определяющее множество типов для основных операций визуализации (шрифты, перья, основные кисти и т.д.), а также основной тип <b>Graphics</b>
<b>System.Drawing.Drawing2D</b>	Предлагает типы, используемые для более сложной двумерной/векторной графики (градиентные кисти, стили концов линий для перьев, геометрические трансформации и т.д.)
<b>System.Drawing.Imaging</b>	Предлагает типы, обеспечивающие обработку графических изображений (изменение палитры, извлечение метаданных изображения, работа с метафайлами и т.д.)
<b>System.Drawing.Printing</b>	Предлагает типы, обеспечивающие отображение графики на печатной странице, непосредственное взаимодействие с принтером и определение полного формата задания печати
<b>System.Drawing.Text</b>	Дает возможность управлять коллекциями шрифтов

# System.Drawing.dll

Все пространства имен GDI+ определены в компоновочном блоке **System.Drawing.dll**.

Многие типы проектов Visual Studio устанавливают ссылку на эту библиотеку программного кода автоматически, но вы можете при необходимости сослаться на **System.Drawing.dll** вручную, используя диалоговое окно **Add References** (Добавление ссылок).



---

## Пространство имен System.Drawing

- Большинство типов, которые придется использовать при создании GDI-приложений, содержится в пространстве имен **System.Drawing**.
  - Здесь есть классы, представляющие изображения, кисти, перья и шрифты.
  - Кроме того, **System.Drawing** определяет ряд связанных утилитарных типов, таких как **Color** (цвет), **Point** (точка) и **Rectangle** (прямоугольник).
-

## Базовые типы System.Drawing

Тип	Описание
<b>Bitmap</b>	Тип, инкапсулирующий данные изображения (*.bmp или какого-то другого)
<b>Brush, Brushes, SolidBrush, SystemBrushes, TextureBrush</b>	Объекты <b>Brush</b> используются для заполнения внутренних областей графических форм, например, таких как прямоугольники, эллипсы и многоугольники
<b>BufferedGraphics</b>	Тип <b>.NET 2.0</b> , обеспечивающий графический буфер для двойной буферизации, которая используется, для уменьшения или полного исключения влияния эффекта мелькания, возникающего при перерисовке изображений
<b>Color, SystemColors</b>	Типы <b>Color</b> и <b>SystemColors</b> определяют ряд статических свойств, доступных только для чтения и используемых для получения нужного цвета при использовании различных перьев и кистей

## Базовые типы System.Drawing (продолжение)

Тип	Описание
<b>Font, FontFamily</b>	<p>Тип <b>Font</b> инкапсулирует характеристики данного шрифта (название, плотность, начертание, размер и т.д.).</p> <p><b>FontFamily</b> предлагает абстракцию для группы шрифтов, имеющих аналогичный дизайн, но определенные вариации стиля</p>
<b>Graphics</b>	Представляет реальную поверхность нанесения изображения, а также предлагает ряд методов для визуализации текста, изображений и геометрических шаблонов
<b>Icon, SystemIcons</b>	Представляют пользовательские пиктограммы, а также набор стандартных пиктограмм, предлагаемых системой

## Базовые типы System (продолжение)

Тип	Описание
<b>Image, ImageAnimator</b>	<p>Тип <b>Image</b> — это абстрактный базовый класс, необходимый для поддержки функциональных возможностей типов <b>Bitmap, Icon</b> и <b>Cursor</b>.</p> <p>Тип <b>ImageAnimator</b> обеспечивает возможность выполнения цикла по набору типов <b>Image</b> из некоторого заданного интервала</p>
<b>Pen, Pens, SystemPens</b>	<p><b>Pens</b> — это объекты, используемые для построения линий и кривых. Тип <b>Pen</b> определяет ряд статических свойств, возвращающих новый объект <b>Pen</b> заданного цвета</p>
<b>Point, PointF</b>	<p>Структуры, представляющие отображение координаты (x, y) в соответствующее целое значение или значение с плавающей точкой, соответственно</p>
<b>Rectangle, RectangleF</b>	<p>Структуры, представляющие размеры прямоугольника (снова с отображением в соответствующее целое значение или значение с плавающей точкой)</p>



## Базовые типы System (продолжение)

Тип	Описание
<b>Size, SizeF</b>	Структуры, представляющие заданные высоту/ширину (снова с отображением в соответствующее целое значение или значение с плавающей точкой)
<b>StringFormat</b>	Тип, используемый для инкапсуляции различных характеристик размещения текста (выравнивание, промежутки между строками и т.д.)
<b>Region</b>	Тип, описывающий геометрический образ, скомпонованный из прямоугольников и траекторий

---

## Класс Graphics

Класс **System.Drawing.Graphics** — это "вход" в функциональные возможности визуализации GDI+.

Этот класс не только представляет поверхность, на которой можно разместить изображение (например, поверхность формы, поверхность элемента управления или область в памяти), но определяет также десятки членов, которые позволяют отображать текст, изображения (пиктограммы, точечные рисунки и т.д.) и самые разные геометрические формы

---

## Методы класса Graphics

Методы	Описание
<b>FromHdc()</b> , <b>FromHwnd()</b> , <b>FromImage()</b>	Статические методы, обеспечивающие возможность получения действительного объекта <b>Graphics</b> из данного изображения (например, пиктограммы, точечного рисунка и т.п.) или GUI-элемента
<b>Clear()</b>	Заполняет объект <b>Graphics</b> заданным цветом, выполняя в процессе заполнения очистку поверхности рисования
<b>DrawArc()</b> <b>DrawCurve()</b> <b>DrawEllipse()</b> <b>DrawIcon()</b> <b>DrawLine()</b> <b>DrawString()</b> <b>DrawRectangle()</b>	Эти методы используются для визуализации данного изображения или геометрического шаблона. Некоторые методы <b>DrawXXX()</b> требуют использования объектов <b>Pen</b> GDI+

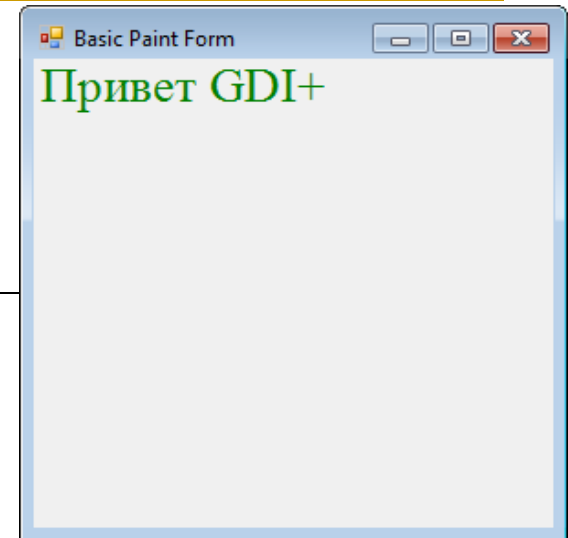
## Методы класса Graphics

Методы	Описание
<b>FillEllipse()</b> <b>FillPath()</b> <b>FillPie()</b> <b>FillPolygon()</b> <b>FillRectangle()</b>	Эти методы используются для заполнения внутренности данной геометрической формы. Методы <b>FillXXX()</b> требуют использования объектов <b>Brush GDI+</b>

## Свойства класса Graphics

Свойства	Описание
<b>Clip, ClipBounds, VisibleClipBounds, IsClipEmpty, IsVisibleClipEmpty</b>	Позволяют установить опции отсечения, используемые с текущим объектом <b>Graphics</b>
<b>Transform</b>	Позволяет трансформировать "мировые координаты"
<b>PageUnit, PageScale, DpiX, DpiY</b>	Позволяют указать начало координат для операций визуализации, а также единицу измерения
<b>SmoothingMode, PixelOffsetMode, TextRenderingHint</b>	Позволяют задать параметры гладкости геометрических объектов и текста
<b>CompositingMode, CompositingQuality</b>	Свойство <b>CompositingMode</b> задает режим визуализации: либо рисование поверх фона, либо сопряжение с фоном
<b>InterpolationMode</b>	Указывает режим интерполяции данных между конечными точками

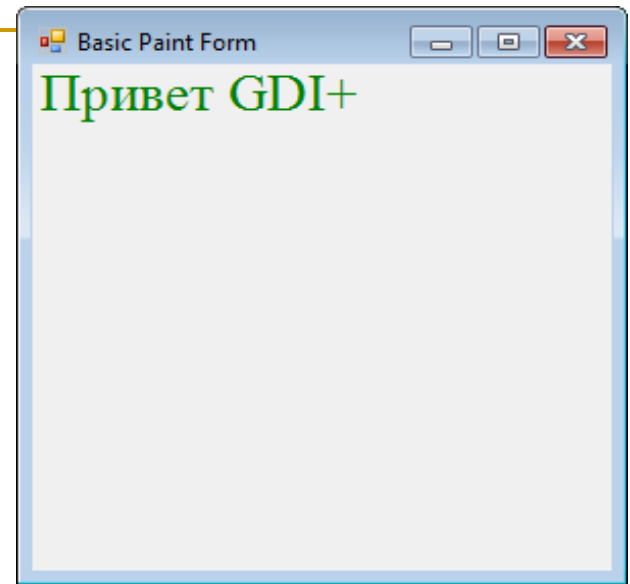
# Сеансы Paint



```
public partial class MainForm : Form
{
    public MainForm()
    {
        InitializeComponent();
        CenterToScreen();
        this.Text = "Basic Paint Form";
    }
    protected override void OnPaint(PaintEventArgs e)
    {
        //При переопределении OnPaint() не забудьте вызвать
        //реализацию базового класса
        base.OnPaint(e);
        //Получение объекта Graphics из поступившего на вход PaintEventArgs
        Graphics g = e.Graphics;
        //Визуализация текстового сообщения с заданными цветом и шрифтом
        string str = "Привет GDI+";
        g.DrawString(str, new Font("Times New Roman", 20), Brushes.Green, 0, 0);
    }
}
```

# Обработка события Paint

```
public partial class MainForm : Form
{
    public MainForm()
    {
        InitializeComponent();
        CenterToScreen();
        this.Text = "Basic Paint Form";
        this.Paint += new PaintEventHandler(MainForm_Paint);
    }
    private void MainForm_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics;
        string str = "Привет GDI+";
        string nameFont = "Times New Roman";
        g.DrawString
            (str, new Font(nameFont, 20), Brushes.Green, 0, 0);
    }
}
```



---

## О событии Paint

Событие **Paint** генерируется всегда, когда окно становится "грязным". Окно считается "грязным", если переопределяется его размер, окно (или его часть) открывается из-под другого окна, или окно сначала минимизируется, а затем восстанавливается.

Во всех случаях, когда требуется перерисовка формы, платформа .NET гарантирует, что обработчик события **Paint** (или переопределенный метод `OnPaint( )`) будет вызван автоматически

---



---

## Обновление области клиента формы

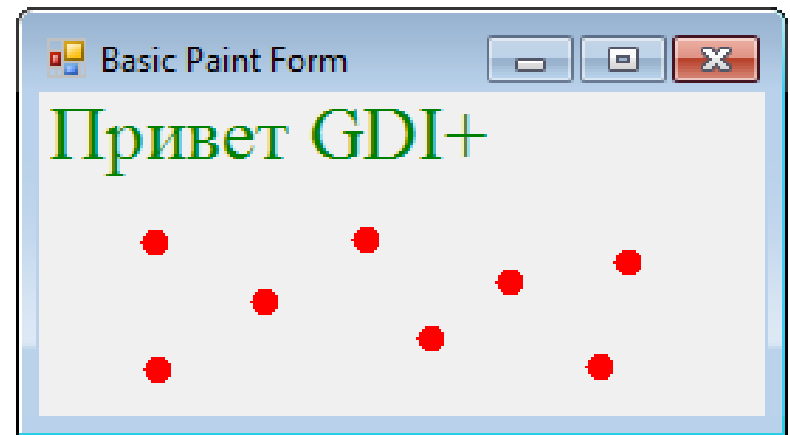
В ходе выполнения приложения GDI+ может возникнуть необходимость в явном вызове события Paint вместо ожидания того, что окно станет “естественно грязным”. Чтобы вызвать перерисовку окна программно, просто вызовите наследуемый метод **Invalidate( )**.

**public void Invalidate( )** - делает недействительной всю поверхность элемента управления и вызывает его перерисовку.

---

## Доступ к объекту Graphics вне обработчика Paint

```
private void MainForm_MouseDown(object sender, MouseEventArgs e)
{
    //Получение объекта Graphics через Hwnd.
    Graphics g = Graphics.FromHwnd(this.Handle);
    //Рисование круга 10*10 по щелчку мыши.
    g.FillEllipse(Brushes.Red, e.X, e.Y, 10, 10);
    //Освобождение объектов Graphics, созданных напрямую.
    g.Dispose();
}
```



# Доступ к объекту Graphics вне обработчика Paint

```
public partial class MainForm : Form
{
    //Используется для хранения всех Point
    private List<Point> myPts = new List<Point>();
    public MainForm()
    {
        InitializeComponent();
        CenterToScreen();
        this.Text = "Basic Paint Form";
        this.Paint += new PaintEventHandler(MainForm_Paint);
    }
    private void MainForm_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics;
        string str = "Привет GDI+";
        string nameFont = "Times New Roman";
        g.DrawString
            (str, new Font(nameFont, 20), Brushes.Green, 0, 0);
        foreach (Point p in myPts)
        {
            g.FillEllipse(Brushes.Red, p.X, p.Y, 10, 10);
        }
    }
    private void MainForm_MouseDown(object sender, MouseEventArgs e)
    {
        myPts.Add(new Point(e.X, e.Y));
        this.Invalidate();
    }
}
```

## Освобождение объекта Graphics

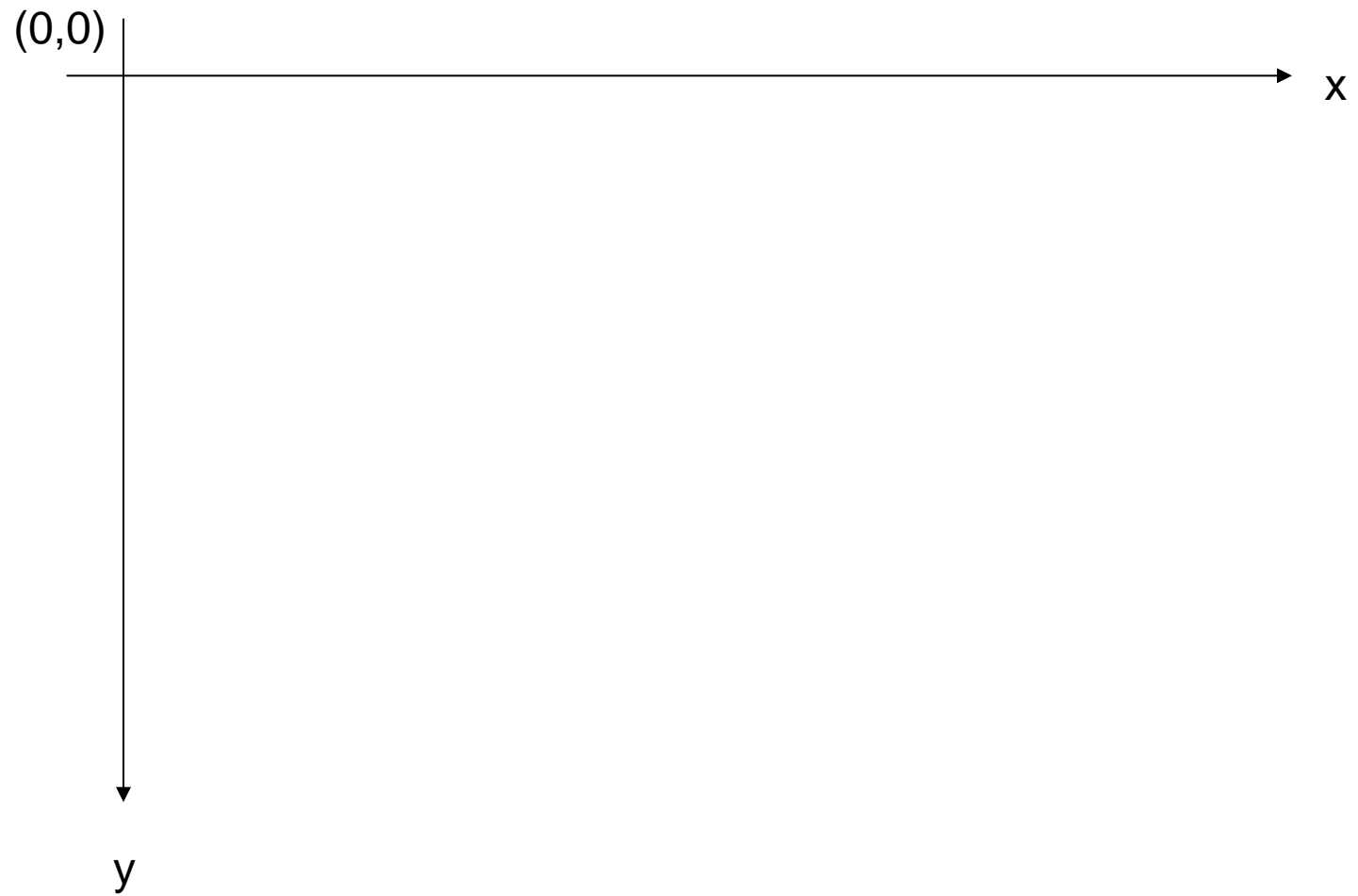
- Если объект **Graphics** был создан вами непосредственно, после окончания его использования его следует освободить
- Если вы ссылаетесь на существующий объект **Graphics**, его освобождать не следует

```
private void MainForm_Paint(object sender, PaintEventArgs e)
{
    //Загрузка локального файла *.jpg
    Image myImageFile = Image.FromFile("photo.jpg") ;
    //Создание нового объекта Graphics на основе изображения
    Graphics imgGraphics = Graphics.FromImage(myImageFile);
    //Визуализация новых данных
    imgGraphics.FillEllipse(Brushes.DarkOrange, 50, 50, 150, 150);
    //Нанесение изображения на форму.
    Graphics g = e.Graphics;
    g.DrawImage(myImageFile, new Point(0,0));
    //Освобождение созданного нами объекта Graphics.
    imgGraphics.Dispose();
}
```

## Системы координат GDI+

- *Мировые координаты* (или внешние координаты) - представляют абстракцию размеров данного типа GDI+, независимую от единиц измерения.
- *Страничные координаты* (координаты страницы) - представляют смещение в применении к оригинальным мировым координатам.
- *Приборные координаты* (координаты устройства) - представляют результат применения страничных координат к оригинальным мировым координатам. Эта координатная система используется для определения того, где именно будет показан соответствующий тип GDI+.

# Система координат

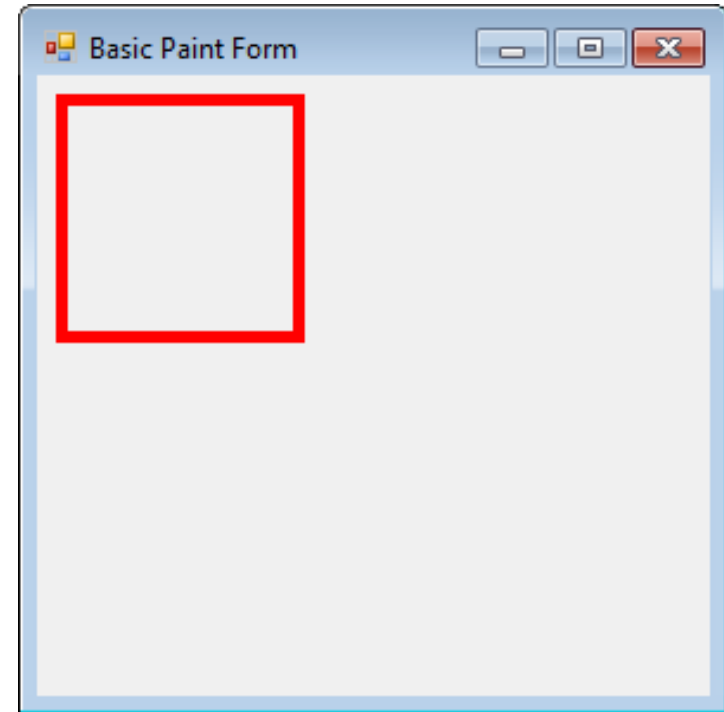


## Единицы измерения

В GDI+ единицей измерения по умолчанию является пиксель.

```
public enum GraphicsUnit
{
    //Мировые координаты.
        World,
    //Пиксель для видео дисплея и 1/100 дюйма для принтера
        Display,
    //Пиксель,
        Pixel,
    //Стандартная точка принтера (1/72 дюйма)
        Point,
    //Дюйм
        Inch,
    //Стандартная единица документа (1/300 дюйма)
        Document,
    //Миллиметр
        Millimeter
}
```

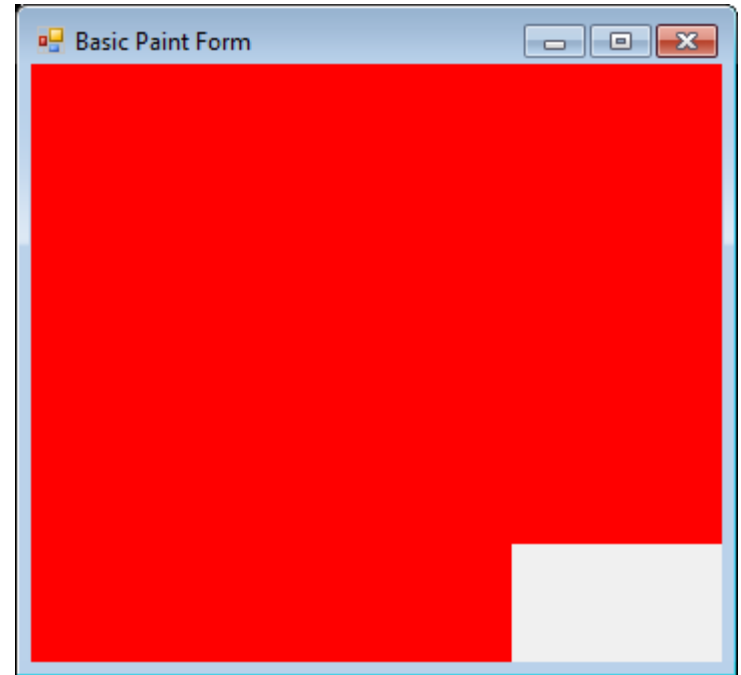
## Выбор единиц измерения



```
private void MainForm_Paint(object sender, PaintEventArgs e)
{
    //Установка мировых координат с использованием единиц измерения,
    //предлагаемых по умолчанию.
    Graphics g = e.Graphics;
    g.DrawRectangle(new Pen(Color.Red, 5), 10, 10, 100, 100);
}
```



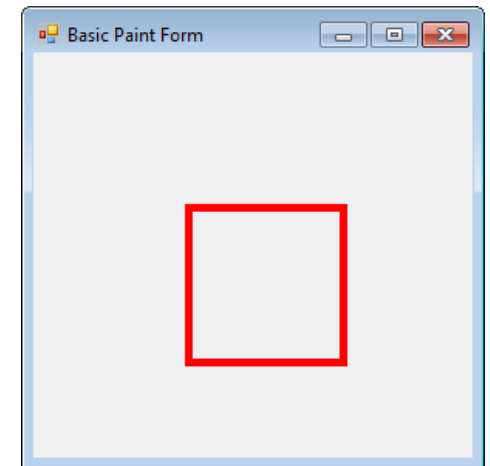
# Выбор единиц измерения



```
private void MainForm_Paint(object sender, PaintEventArgs e)
{
    //Отображение прямоугольника в дюймах, а не в пикселях...
    Graphics g = e.Graphics;
    g.PageUnit = GraphicsUnit.Inch;
    g.DrawRectangle(new Pen (Color.Red, 5), 0, 0, 100, 100);
}
```

## Изменение начала координат

```
private void MainForm_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    //Установка смещения (100, 100) для страничных координат
    g.TranslateTransform(100, 100);
    //Значениями мировых координат остаются (0, 0, 100, 100),
    //но приборные координаты теперь равны (100, 100, 200, 200).
    g.DrawRectangle(new Pen (Color.Red, 5), 0, 0, 100, 100);
}
```



## Определение цветовых значений

Структура **System.Drawing.Color** представляет цветовую константу ARGB (от Alpha-Red-Green-Blue— альфа, красный, зеленый, синий).

```
//Один из множества встроенных цветов...  
Color c = Color.RoyalBlue;  
//Указание ARGB вручную.  
Color myColor = Color.FromArgb(0, 255, 128, 64);
```

### Члены типа **Color**:

**GetBrightness** ( ) — возвращает значение яркости типа **Color** на основании измерения HSB (Hue-Saturation-Brightness — оттенок, насыщенность, яркость).

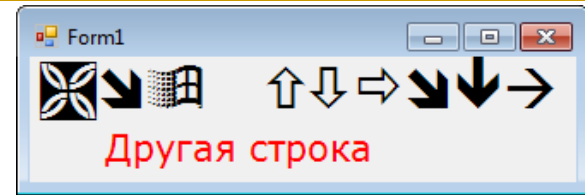
**GetSaturation** ( ) — возвращает значение насыщенности типа **Color** на основании измерения HSB.

**GetHue** ( ) — возвращает значение оттенка типа **Color** на основании измерения HSB.

**IsSystemColor** — индикатор того, что данный тип **Color** является зарегистрированным системным цветом.

A, R, G, B — возвращают значения, присвоенные для альфа, красной, зеленой и синей составляющих типа **Color**.

# Работа со шрифтами



Тип **System.Drawing.Font** представляет шрифт, установленный на машине ПОЛЬЗОВАТЕЛЯ.

```
//Создание Font с заданными именем типа и размером.  
Font f = new Font("Times New Roman", 12);  
//Создание Font с заданными именем типа, размером и начертанием  
Font f2 =  
    new Font("WingDings", 50, FontStyle.Bold|  
FontStyle.Underline);
```

```
private void Form1_Paint(object sender, PaintEventArgs e)  
{  
    Graphics g = e.Graphics;  
    g.DrawString  
        ("Моя строка", new Font("WingDings", 25),  
        Brushes.Black, new Point(0, 0));  
    g.DrawString  
        ("Другая строка", new Font("Verdana", 16),  
        Brushes.Red, 40, 40);  
}
```

## Классы System.Drawing.Drawing2D

Классы	Описание
<b>AdjustableArrowCap</b> <b>CustomLineCap</b>	Используются для изменения формы концов линий для перьев. Данные типы задают, соответственно, регулируемую стрелку и пользовательскую форму конца линии
<b>Blend</b> <b>ColorBlend</b>	Позволяют определить шаблон смешивания (и цвет) для использования с <b>LinearGradientBrush</b>
<b>GraphicsPath</b> <b>GraphicsPathIterator</b> <b>PathData</b>	Объект <b>GraphicsPath</b> представляет серию линий и кривых. Этот класс позволяет добавлять в траектории геометрические шаблоны практически любого вида (дуги, прямоугольники, линии, строки, многоугольники и т.д.). <b>PathData</b> содержит графические данные, формирующие траекторию
<b>HatchBrush</b> <b>LinearGradientBrush</b> <b>PathGradientBrush</b>	Экзотические типы кистей

---

## Работа с типами Pen

Типы **Pen** GDI+ используются для построения линий, соединяющих конечные точки.

Для выполнения визуализации геометрической формы на поверхности производного от **Control** типа действительный тип **Pen** следует направить подходящему методу визуализации, определенному классом **Graphics**.

С объектами **Pen** обычно используются методы **DrawXXX()**, позволяющие отобразить некоторый набор линий на соответствующей графической поверхности.

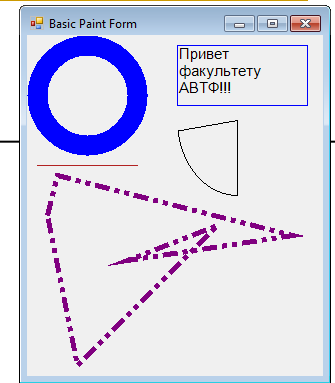
---

## Свойства Pen

Свойства	Описание
<b>Brush</b>	Определяет тип <b>Brush</b> для использования с данным типом <b>Pen</b>
<b>Color</b>	Определяет тип <b>Color</b> для использования с данным типом <b>Pen</b>
<b>CustomStartCap</b> <b>CnstomEndCap</b>	Читает или устанавливает параметры пользовательского стиля концов линий, создаваемых с помощью данного типа <b>Pen</b> , Стиль <i>концов</i> линий — это просто термин, используемый для обозначения того, как должен выглядеть начальный и заключительный "штрих" данного пера.
<b>DashCap</b>	Читает или устанавливает параметры стиля концов линий, используемого для прерывистых линий, создаваемых с помощью данного типа <b>Pen</b>
<b>DashPattern</b>	Читает или устанавливает массив пользовательской маски для рисования прерывистых линий. Соответствующие "тире" складываются из сегментов линий
<b>DashStyle</b>	Читает или устанавливает параметры стиля, используемого для прерывистых линий, создаваемых с помощью данного типа <b>Pen</b>
<b>StartCap</b> <b>EndCap</b>	Читает или устанавливает встроенный стиль концов линий, создаваемых с помощью данного типа <b>Pen</b> . Стиль концов линий <b>Pen</b> устанавливается в соответствии с перечнем <b>LineCap</b> из пространства имен <b>System.Drawing.Drawing2D</b>
<b>Width</b>	Читает или устанавливает ширину данного <b>Pen</b>
<b>DashOffset</b>	Читает или устанавливает расстояние от начала линии до начала шаблона прерывистой линии

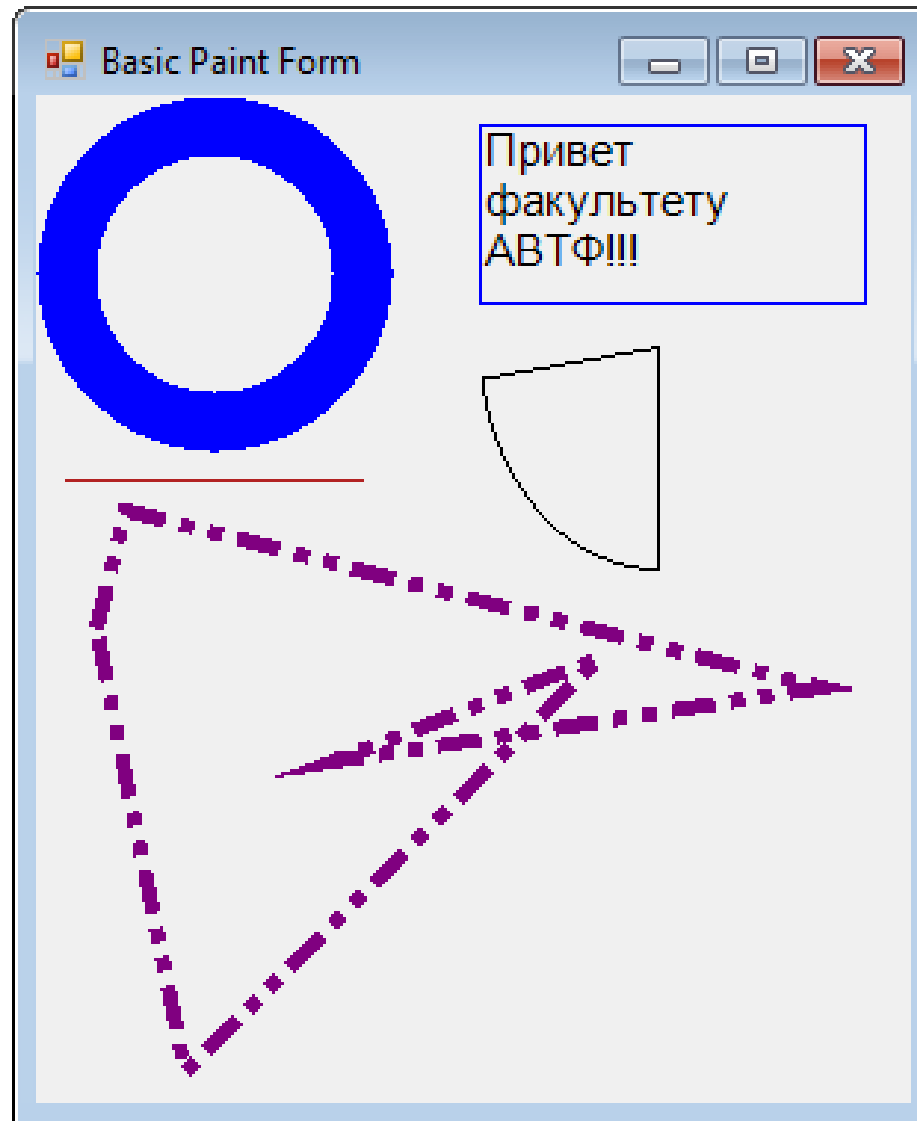
# Пример кода

```
private void MainForm_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    //Создание большого пера синего цвета.
    Pen bluePen = new Pen(Color.Blue, 20);
    //Получение готового пера из типа Pens.
    Pen pen2 = Pens.Firebrick;
    //Визуализация некоторых шаблонов.
    g.DrawEllipse (bluePen, 10, 10, 100, 100);
    g.DrawLine(pen2, 10, 130, 110, 130);
    g.DrawPie(Pens.Black, 150, 10, 120, 150, 90, 80);
    //Рисование пурпурного полигона с пунктирной границей...
    Pen pen3 = new Pen(Color.Purple, 5);
    pen3.DashStyle = DashStyle.DashDotDot;
    g.DrawPolygon (pen3, new Point[]
        {
            new Point(30, 140), new Point(265, 200),
            new Point(100, 225), new Point(190, 190),
            new Point(50, 330), new Point (20, 180)
        });
    //... и прямоугольника, содержащего текст...
    Rectangle r = new Rectangle(150, 10, 130, 60);
    g.DrawRectangle (Pens.Blue, r);
    string str = "Привет факультету АВТФ!!!";
    g.DrawString(str, new Font("Arial", 11), Brushes.Black, r) ;
}
}
```



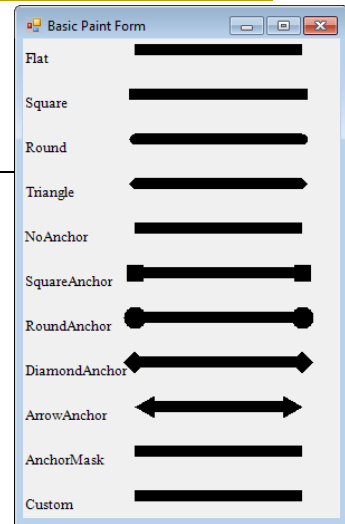


# Пример кода

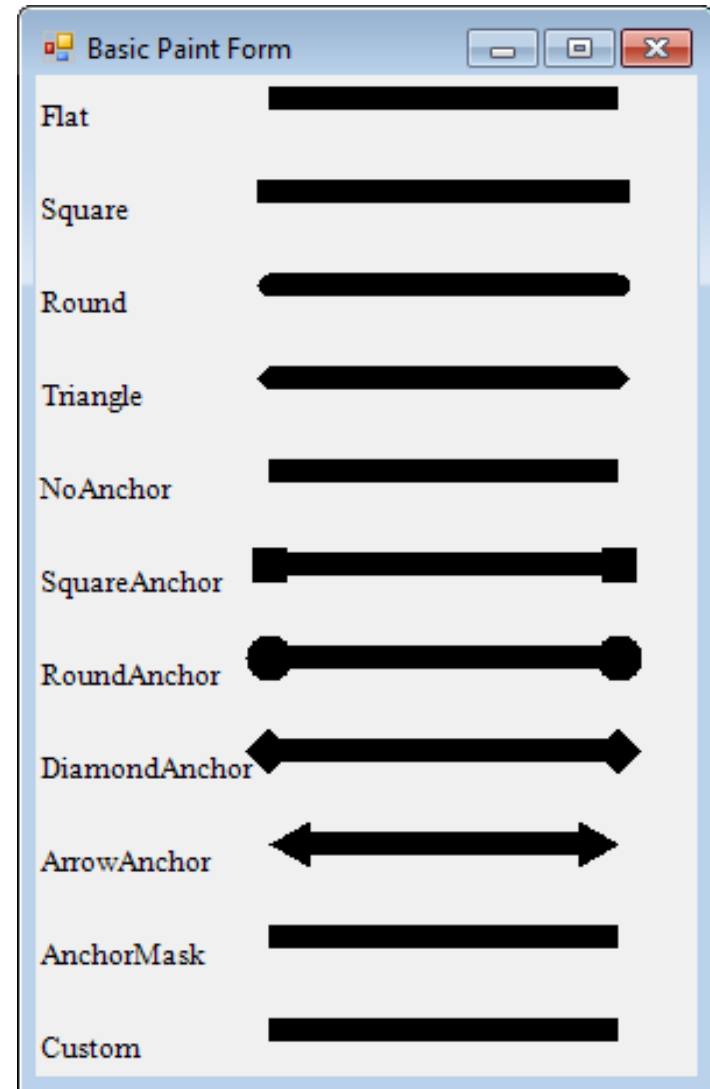


# КОНЦЫ ЛИНИЙ

```
private void MainForm_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Pen thePen = new Pen(Color.Black, 10);
    int yOffset = 10;
    //Получение всех членов перечня LineCap.
    Array obj = Enum.GetValues(typeof(LineCap));
    //Рисование линии для данного члена LineCap.
    for (int x = 0; x < obj.Length; x++)
    {
        //Получение следующего стиля конца линии и настройка
пера.
        LineCap temp = (LineCap)obj.GetValue(x);
        thePen.StartCap = temp; thePen.EndCap = temp;
        //Вывод имени из перечня LineCap.
        g.DrawString(temp.ToString(),
            new Font("Times New Roman", 10),
            new SolidBrush(Color.Black), 0, yOffset);
        //Рисование линии с соответствующим стилем концов.
        g.DrawLine(thePen, 100, yOffset, Width - 50, yOffset);
        yOffset += 40;
    }
}
```



# КОНЦЫ ЛИНИЙ



---

## Работа с типами **Brush**

Типы, производные от **System.Drawing.Brush**, используются для заполнения имеющегося региона заданным цветом, узором или изображением.

Сам класс **Brush** является абстрактным типом, поэтому он не позволяет создать соответствующий экземпляр непосредственно.

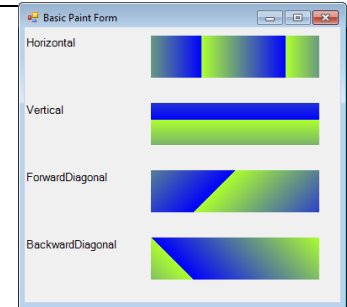
Однако **Brush** может играть роль базового класса для родственных ему типов кисти (например, **SolidBrush**, **HatchBrush**, **LinearGradientBrush** и др.).

Имея кисть, вы получаете возможность вызвать любой из методов **FillXXX()** типа **Graphics**.

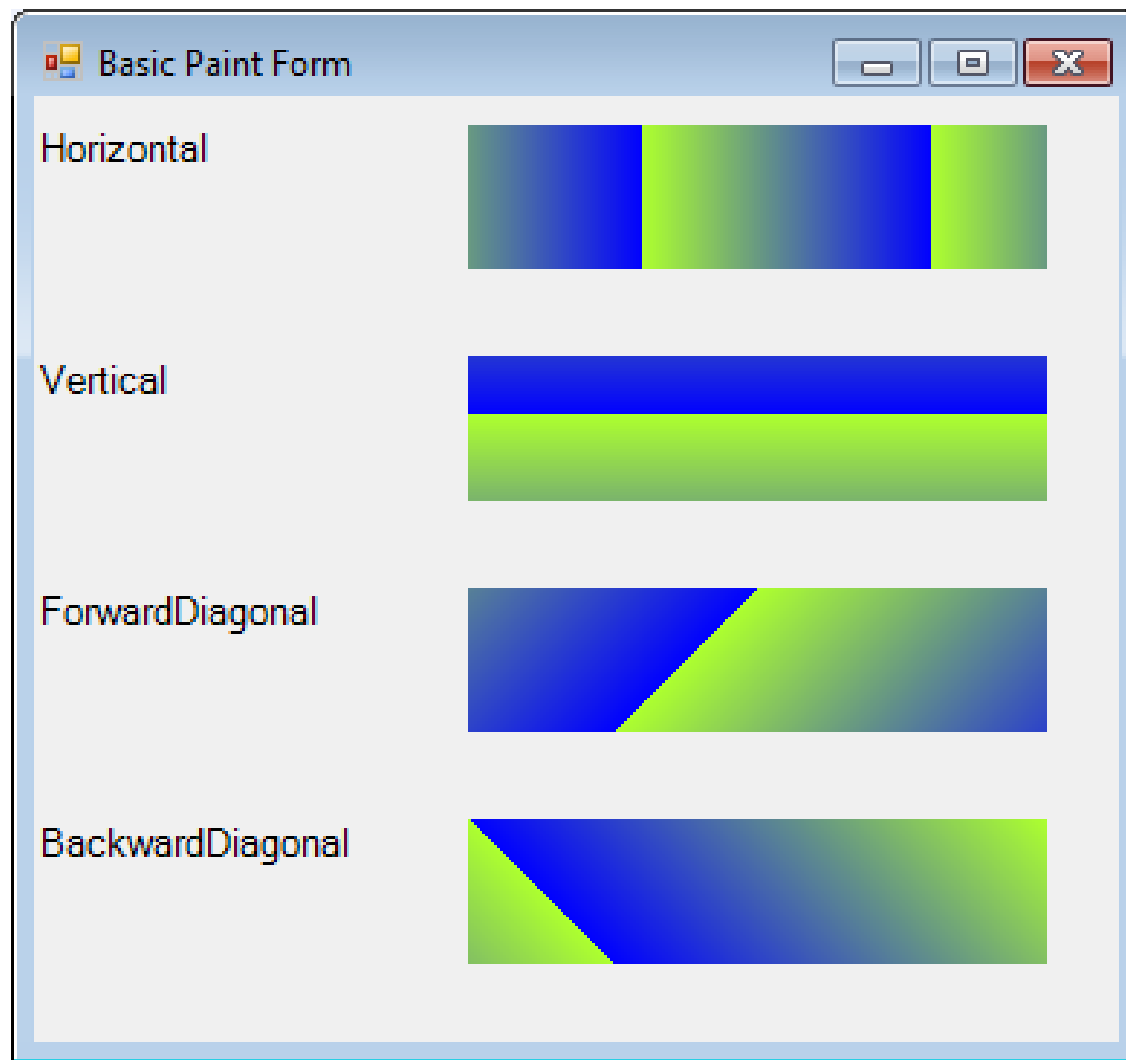
---

# Работа с LinearGradientBrush

```
private void MainForm_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Rectangle r = new Rectangle (10, 10, 100, 100);
    //Градиентная кисть.
    LinearGradientBrush theBrush = null;
    int yOffset = 10;
    //Получение членов перечня LinearGradientMode.
    Array obj = Enum.GetValues (typeof (LinearGradientMode));
    //Отображение прямоугольников для членов LinearGradientMode.
    for (int x = 0; x < obj.Length; x++)
    {
        //Конфигурация кисти.
        LinearGradientMode temp = (LinearGradientMode)obj.GetValue (x);
        theBrush = new LinearGradientBrush
            (r, Color.GreenYellow, Color.Blue, temp);
        //Вывод имени из перечня LinearGradientMode.
        g.DrawString (temp.ToString (), new Font ("Times Sew Raman", 10),
            new SolidBrush (Color.Black), 0, yOffset);
        //Закраска прямоугольника подходящей кистью.
        g.FillRectangle (theBrush, 150, yOffset, 200, 50);
        yOffset += 80;
    }
}
```



# Работа с LinearGradientBrush



## Визуализация изображений

Абстрактный тип **System.Drawing.Image** определяет ряд методов и свойств, хранящих различную информацию о том изображении, которое этот тип представляет.

Члены типа Image	Описание
<b>FromFile()</b>	Статический метод, создающий объект <b>Image</b> из указанного файла
<b>FromStream()</b>	Статический метод, создающий объект <b>Image</b> из указанного потока данных
<b>Height, Width, Size, HorizontalResolution, VerticalResolution</b>	Свойства, возвращающие информацию о размерах данного объекта <b>Image</b>
<b>Palette</b>	Свойство, возвращающее тип данных <b>ColorPalette</b> , который представляет палитру, используемую для данного объекта <b>Image</b>
<b>GetBounds</b>	Метод, возвращающий объект <b>Rectangle</b> , который представляет текущие размеры данного объекта <b>Image</b>
<b>Save()</b>	Метод, сохраняющий в файл данные, содержащиеся в производном от <b>Image</b> типе

# Класс Bitmap

```
public partial class MainForm : Form
{
    private Bitmap[] myImages = new Bitmap[3];
    public MainForm()
    {
        // Загрузка локальных изображений.
        myImages[0] = new Bitmap ("image1.jpg");
        myImages[1] = new Bitmap ("image2.jpg");
        myImages[2] = new Bitmap ("image3.jpg");
        InitializeComponent();
        this.Text = "Basic Paint Form";
        this.Paint += new PaintEventHandler(MainForm_Paint);
    }
    private void MainForm_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics;
        //Визуализация изображений.
        int xOffset = 10;
        foreach (Bitmap b in myImages)
        {
            g.DrawImage(b, xOffset, 10, 150, 150);
            xOffset += 160;
        }
    }
}
```





## Двойная буферизация

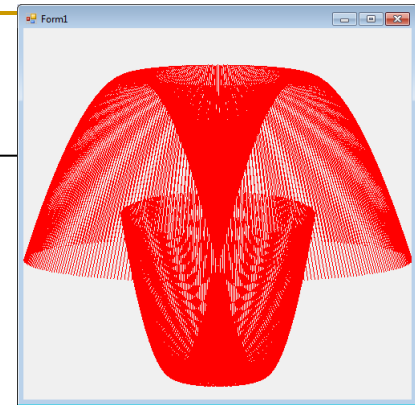
Мерцание является распространенной проблемой при программировании графики. Графические операции, требующие нескольких сложных операций рисования могут привести к тому, что визуализируемые изображения будут мерцать или иметь неприемлемый внешний вид. Чтобы устранить эти неполадки, **.NET Framework** предоставляет доступ к двойной буферизации.

При двойной буферизации все операции рисования сначала выполняются в памяти, а лишь затем на экране компьютера. После завершения всех операций рисования содержимое буфера копируется из памяти непосредственно на связанную с ним область экрана.

```
public Form1 ()
{
    InitializeComponent ();
    //Включение двойной буферизации
    this.DoubleBuffered = true;
}
```

# Двойная буферизация

```
public Form1 ()
{
    InitializeComponent();
    this.Size = new Size(500, 500);
    this.DoubleBuffered = true;
}
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    int num = 300;
    int cx = ClientSize.Width;
    int cy = ClientSize.Height;
    Pen pen = new Pen(Color.Red);
    PointF[] aptf = new PointF[4];
    for (int i = 0; i < num; i++)
    {
        double dAngle = 2 * i * Math.PI / num;
        aptf[0].X = cx / 2 + cx / 2 * (float)Math.Cos(dAngle);
        aptf[0].Y = 5 * cy / 8 + cy / 16 * (float)Math.Sin(dAngle);
        aptf[1] = new PointF(cx / 2, -cy);
        aptf[2] = new PointF(cx / 2, 2 * cy);
        dAngle += Math.PI;
        aptf[3].X = cx / 2 + cx / 4 * (float)Math.Cos(dAngle);
        aptf[3].Y = cy / 2 + cy / 16 * (float)Math.Sin(dAngle);
        g.DrawBeziers(pen, aptf);
    }
}
```



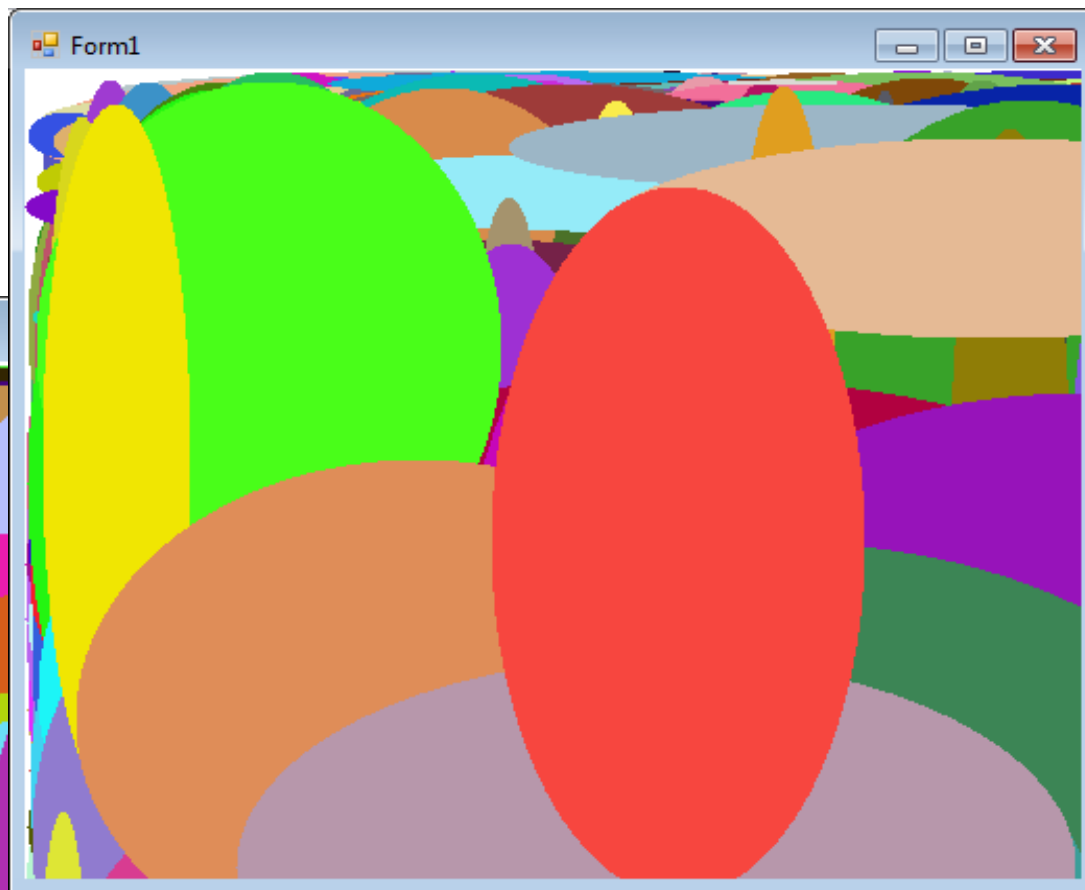
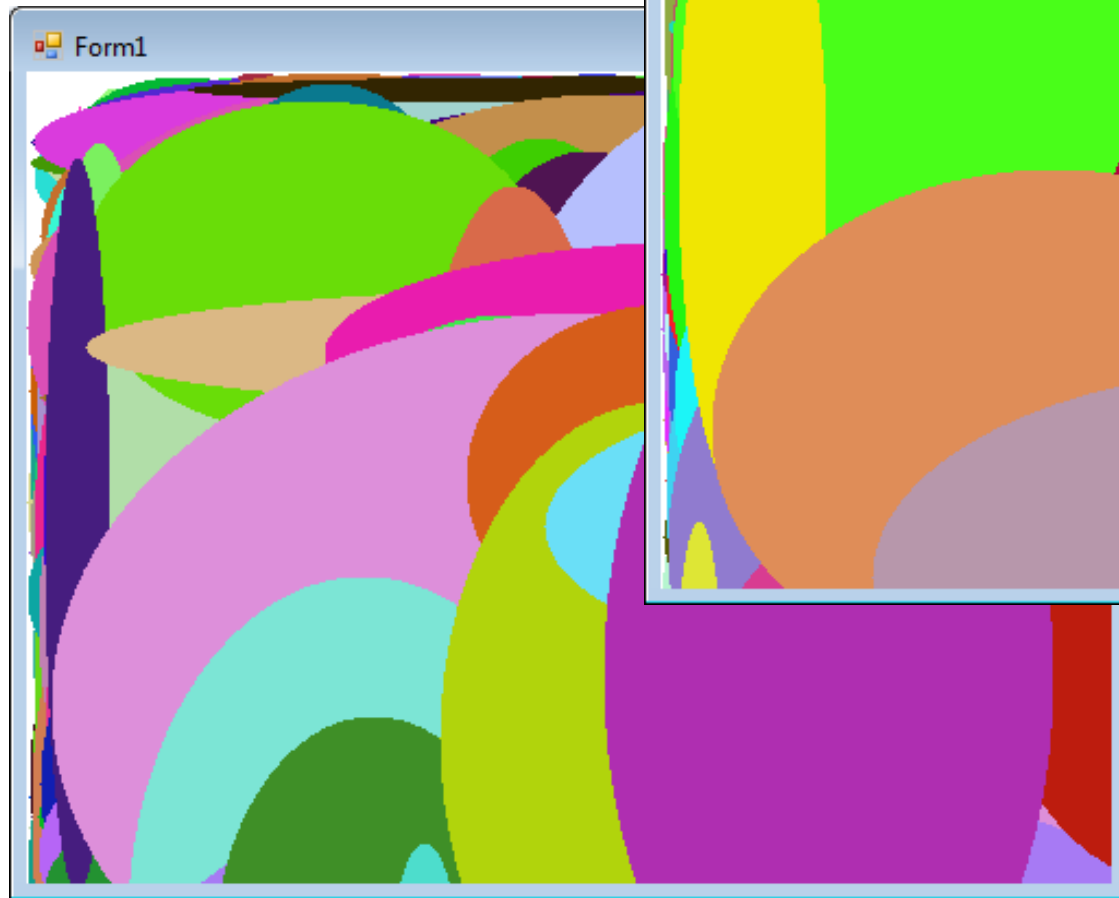
# Пример кода

```
public partial class MyForm : Form
{
    Random rnd = new Random();
    int n = 5000;
    Bitmap bitmap;
    public MyForm()
    {
        InitializeComponent();
        bitmap = new Bitmap(1, 1);
    }
    private void MyForm_Paint
        (object sender, PaintEventArgs e)
    {
        Graphics grFrom = e.Graphics;
        grFrom.DrawImage(bitmap, 0, 0);
    }
}
```

```
private void MyForm_MouseDown(object sender,
MouseEventArgs e)
{
    int x1, y1, x2, y2;
    Color col;
    SolidBrush sb;
    int r, g, b;
    int maxC = 255;
    int maxX = this.ClientRectangle.Width;
    int maxY = this.ClientRectangle.Height;
    bitmap = new Bitmap(maxX, maxY);
    Graphics grImage = Graphics.FromImage(bitmap);
    grImage.Clear(Color.White);
    for (int i = 0; i < n; i++)
    {
        r = rnd.Next(maxC);
        g = rnd.Next(maxC);
        b = rnd.Next(maxC);
        col = Color.FromArgb(r, g, b);
        sb = new SolidBrush(col);
        x1 = rnd.Next(maxX); x2 = rnd.Next(maxX);
        y1 = rnd.Next(maxY); y2 = rnd.Next(maxY);
        grImage.FillEllipse(sb, x1, y1, x2, y2);
    }
    this.Invalidate();
}
```



# Пример кода



# АНИМАЦИЯ

```
public partial class Form1 : Form
{
    Bitmap myImage;
    int h = 30;
    int w = 30;
    SolidBrush sbFon;
    SolidBrush sbFill;
    Thread thread;
    public Form1()
    {
        InitializeComponent();
        this.Size = new Size(500, 500);
        myImage = new Bitmap
            (ClientRectangle.Width, ClientRectangle.Height);
        this.DoubleBuffered = true;
        sbFon = new SolidBrush(this.BackColor);
        sbFill = new SolidBrush(Color.Red);
    }

    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        lock (this)
        {
            Graphics g = e.Graphics;
            g.DrawImage(myImage, 0, 0);
        }
    }
}
```

```
private void Animator()
{
    int x = 0, y = 0;
    Rectangle r;
    Graphics g = Graphics.FromImage(myImage);
    while (x < ClientRectangle.Width)
    {
        r = new Rectangle(x, y, w, h);
        lock (this)
        {
            g.FillEllipse(sbFill, r);
        }
        this.Invalidate();
        Thread.Sleep(5);
        x++;
        y++;
        lock (this)
        {
            g.FillEllipse(sbFon, r);
        }
    }
}

private void Form1_MouseDown
    (object sender, MouseEventArgs e)
{
    thread = new Thread(this.Animator);
    thread.Start();
}
}
```

# Анимация

