Основные понятия компьютерной графики

При работе с графическими данными решаются следующие задачи:

1. Создание изображения;
2. Редактирование изображения;
3. Выбор структур данных для хранения графической информации в оперативной и внешней памяти (внутреннее и внешнее представление данных);
4. Ввод графических данных и воспроизведение по прочитанным данным изображения.

Создаваемые графические изображения с целью их повторного использования записываются в файлы. Часто файлы с графическими данными содержат и дополнительную неграфическую информацию. Например, в пакетах анимации в файл добавляются алгоритмы изменения изображения и его отдельных частей.

Графические данные могут отображать двумерные, трёхмерные и в принципе любые *n*-мерные объекты. В практических приложениях используются двухмерные и трёхмерные модели.

Независимо от того, какие объекты хранятся в графических данных (двухмерные или трёхмерные), вывод изображения осуществляется на плоскость (экран, принтер и др.).

Графический дисплей

Графические возможности дисплеев современных компьютеров определяются размером экрана, зерном экрана, частотой регенерации и разрешением.

Размер экрана дисплея может варьироваться от 14 до 21 дюймов и более.

Вывод изображения выполняется в результате подсветки электронным лучом отдельных регулярно расположенных точек экрана. Такая точка является наименьшим элементом графической информации и называется *видеопикселем*или просто *пикселем.*

Число таких точек по горизонтали и вертикали экрана определяет его *разрешение*. Стандартными являются разрешения 640х480, 800х600, 1024х768, 1280х1024 и более высокие разрешения. На одном дисплее могут быть установлены разные разрешения. Управление разрешением осуществляется графическим адаптером.

Луч последовательно пробегает все точки экрана, выполняя заданную программой для каждой точки подсветку. Затем цикл обхода повторяется. Полный цикл обхода всех точек экрана приводит к воспроизведению изображения и называется *регенерацией изображения.* Число регенераций в секунду называется *частотой регенерации*. Низкая частота регенерации приводит к заметному для человеческого глаза мерцанию изображения. Хорошим показателем можно считать частоту регенерации не менее 60 Гц.

Расстояние между соседними точками экрана определяет его *зерно*. Размеры зерна варьируются от 0,31 мм до 0,21 мм и менее. Чем меньше зерно, тем выше качество изображения.

Виды графических изображений

Все графические изображения подразделяются на*векторные* и *растровые*. Векторные изображения генерируются с помощью специальных графических методов. В основе любого векторного изображения лежат математические определения различных геометрических фигур. В растровом изображении отдельно определяется каждый пиксель. Растровые рисунки часто размещают на различных элементах управления. Важное различие между векторной и растровой графикой состоит в том, что векторная графика не привязана к конкретному разрешению монитора. Это означает, что при любом разрешении векторное изображение будет иметь один и тот же размер, заданные в его определении.

Растровое изображение включает строго определённое количество пикселей. Например, изображение размером 500 х 500 пикселей при любом разрешении будет содержать 250 тыс. пикселей. Если необходимо заполнить таким изображением весь экран при разрешении, например, 1280 х 1024 пикселей, то придется растянуть его, повторяя в нём одни и те же пиксели, отчего оно будет выглядеть зернистым. Иллюстрацией этого эффекта является, например, процедура изменения размера рисунка с помощью меню в редакторе Paint.

Несмотря на некоторые ограничения, растровая графика используется чаще. Векторным способом формируются только те изображения, которые можно представить в виде набора геометрических фигур, с его помощью трудно изобразить реалистичный пейзаж.

Текст относится к категории векторной графики, поскольку символы различных шрифтов описываются математически, и их можно масштабировать без потери качества.

VisualStudio .Net позволяет выводить изображения практически на любом элементе управления, однако на практике графика выводится в двух объектах: собственно на форме и на специальном объекте, предназначенном для вывода графики – PictureBox. Изображения можно помещать на элементы управления как во время разработки приложения, так и во время выполнения.

Формирование растрового изображения

В графике экран монитора представляется в виде прямоугольной сетки на дискретной плоскости с шагом по осям *x*и *y*, равным единице. Такая модель называется *растровой плоскостью* или *растром*. Массив прямоугольных ячеек плоскости называется *растровым массивом.*

i,j

Каждый квадрат сетки соответствует одному пикселю экрана. Растровое изображение создаётся путём закраски ячеек растрового массива в тот или иной цвет. Растровое изображение можно сравнить с изображением на листе клетчатой бумаги, получаемым в результате закраски отдельных клеточек листа.

Каждому пикселю растра могут быть независимым образом заданы цвет, интенсивность и другие характеристики.

При создании чёрно-белых изображений для хранения информации о цвете одного пикселя достаточно 1 бита памяти, записывая в соответствующую ячейку памяти 1, если элемент изображения закрашен в чёрный цвет, и 0 – если в белый.

Цветное изображение создаётся комбинацией базовых цветов, в качестве которых могут быть использованы, например, красный, зелёный и синий (RGB). Задавая различную интенсивность при наложении базовых цветов, можно воспроизвести все различимые человеческим глазом цвета и оттенки. При работе с палитрой, содержащей 256 цветов, для хранения информации о цвете одного пикселя потребуется 1 байт или 8 бит памяти.

В системе RGB каждый пиксель содержит 3 точки, каждая из которых отвечает за свой цвет. Комбинируя эти цвета и их интенсивность, можно управлять цветом каждого пикселя, и, следовательно, всего изображения. Человеческий глаз не в состоянии различать отдельные точки пикселя в силу их маленького размера и воспринимает их как целое получаемое в результате наложения цветов изображение.

Видеоадаптер

Данные, подлежащие воспроизведению на экране монитора, формируются программой, функционирующей на центральном процессоре компьютера. Далее они передаются видеоадаптеру, который размещает эти данные в видеопамяти, преобразовывает и передаёт устройству управления лучом – электронной лучевой трубке.

Таким образом, *видеоадаптер* является устройством сопряжения центрального процессора и устройства отображения.

Видеоадаптеры могут обеспечить разные режимы отображения данных, которые разделяются на*текстовые* и *графические*. Как правило, видеоадаптер поддерживает несколько графических режимов. Режим отображения данных на экране называется *видеорежимом.*

Различные видеорежимы отличаются количеством выводимых на экран данных и цветов. Единицами измерения количества выводимых данных являются: в текстовом режиме – символ (литера), в графическом – пиксель. Например, видеоадаптер SVGA позволяет установить текстовый режим, в котором выводится 37 строк по 100 символов в каждой (100х37) и несколько графических режимов, минимальным из которых является 800х600 пикселей.

В текстовом режиме 100х37 символ имеет размер 8х8 пикселей. Матрица 8х8 пикселей, используемая для отображения символа на экране, называется *знакоместом.* Символ передаётся в видеоадаптер в виде двухбайтовой последовательности. Первый байт – код символа, второй – его атрибуты: цвет символа и фона.

В *графическом* режиме отображаются два типа объектов: литеры (символы) и геометрические объекты.

При отображении литер используются шаблоны (шрифты). В видеоадаптер для отображения строки литер следует передать коды литер, атрибуты строки (высота, угол наклона, цвет и другие), а также имя шрифта (имя файла, в котором размещены формы литер). При отображении литеры будет, во-первых, найдена форма литеры в указанном шрифте, выполнено преобразование (масштабирование и изменение угла наклона) и отображение преобразованной формы литеры на экране монитора в соответствии с её атрибутами.

Построение геометрических объектов выполняется по заданным точкам. Для геометрических объектов, так же, как и для литер, могут быть определены атрибуты: тип, цвет, толщина линии и другие.

Элемент управления PictureBox

Данный элемент используется для размещения в нём графических изображений, которые имеются в файлах компьютера. Расширение графического файла может быть любым (.BMP, .JPG и др.). Кроме этого, в PictureBox можно создавать свои рисунки.

Загрузка готового рисунка осуществляется с помощью свойства Image. Можно выбрать два способа задания изображения:

1. В качестве значения данного свойства необходимо ввести имя графического файла. В этом случае этот файл должен распространяться вместе с приложением, причём расположение файла должно быть постоянным.
2. Импортировать изображение в папку проекта в скрытый файл, имеющий то же имя, что и форма, с расширением .resx. При этом не требуется распространять исходный файл изображения вместе с приложением.

Для программной загрузки графического файла используют следующий код:

PictureBox1.Image = Image.FromFile("имяфайла")

Имя файла необходимо указывать полностью со всеми папками. Для Visual Basic имя файла пишется как обычно, для C# в качестве разделителя папок используется двойной обратный слеш, например:

**Visual Basic .Net:**

pictureBox1.Image = Image.FromFile("C:\Users\Admin\Desktop\DSCN1448.JPG")

**Visual C#:**

pictureBox1.Image = Image.FromFile("C:\\Users\\Admin\\Desktop\\DSCN1448.JPG");

Выбор имени файла можно осуществлять с помощью элемента управления OpenFileDialog.

Для определения соотношением размеров элемента управления PictureBox и рисунка используется свойство SizeMode. Оно может принимать значения:

* Normal - рисунок отображается без изменения независимо от размеров элемента управления PictureBox;
* StretchImage - рисунок изменяет размеры, подстраиваясь под размеры элемента управления;
* AutoSize - элемент управления меняет размеры в соответствии с размерами рисунка;
* СenterImage - центр рисунка помещается в центре элемента управления без изменения размеров.

Свойство BorderStyle определяет вид границ элемента управления и имеет следующие значения:

* None - границы отсутствуют;
* FixedSingle - граница принимает вид тонкой линии черного цвета;
* Fixed3D- граница принимает объемный (трехмерный) вид.

К другим важнейшим свойствам элемента управления PictureBox относятся:

* Location – определяет координаты левого верхнего угла элемента управления;
* Size – определяет размеры элемента управления (значения этих свойств измеряются в пикселях);
* BackColor – определяет фоновый цвет элемента управления (когда в нём не загружено изображение);
* InitialImage – определяет изображение, которое отображается в то время, пока в элемент загружается другое изображение;
* ErrorImage – определяет изображение, которое отображается в случае ошибки загрузки другого изображения. Обычно значения этих свойств фиксируются.

Программное изменение значений этих свойств осуществляется следующим образом:

**Visual Basic .Net:**

PictureBox1.SizeMode = PictureBoxSizeMode.AutoSize

PictureBox1.Location = New Point(100, 100)

PictureBox1.BackColor = Color.Blue

PictureBox1.BorderStyle = BorderStyle.Fixed3D

PictureBox1.Size = New Size(250, 259)

**Visual C#:**

pictureBox1.SizeMode = PictureBoxSizeMode.AutoSize;

pictureBox1.Location = new Point(100, 100);

pictureBox1.BackColor = Color.Blue;

pictureBox1.BorderStyle = BorderStyle.Fixed3D;

pictureBox1.Size = new Size(25, 25);

Свойство Image элемента управления PictureBox представляет собой объект Image, который представляет собой растровое изображение и имеет набор свойств и методов для манипулирования этим изображением. Важнейшими свойствами объекта Image являются:

* HorizontalRezolution и VerticalRezolution – свойства, доступные только для чтения, возвращают разрешение изображения по горизонтали и вертикали в пикселях на дюйм.
* Width и Height – свойства, доступные только для чтения, возвращают высоту и ширину изображения в пикселях. Если разделить значения этих свойств на значения предыдущих свойств, то получатся реальные размеры этого изображения в дюймах, что необходимо знать, например, при печати этого изображения.

Важнейшим методом объекта Image является метод RotateFlip, который поворачивает рисунок или выполняет его зеркальное отображение Способ поворота определяется аргументом данного метода из перечисления RotateFlipType. При этом обязательно надо перерисовать элемент управления с помощью метода Refresh, например:

**Visual Basic .Net:**

PictureBox1.Image.RotateFlip(RotateFlipType.Rotate180FlipY)

PictureBox1.Refresh()

**Visual C#:**

pictureBox1.Image.RotateFlip(RotateFlipType.Rotate180FlipY);

pictureBox1.Refresh();

Создание анимации в формах Windows

Под анимацией понимается динамическая смена изображения на форме или в элементе управления PictureBox. Для отсчёта интервалов времени используется невизуальный элемент Timer. Важнейшее свойство этого элемента – Interval, которое определяет частоту срабатывания таймера. Значение свойства измеряется в миллисекундах, по умолчанию установлено значение 100.

Нормальной скоростью смены кадра считается 24 кадра в секунду, но это используется в основном при демонстрации фильмов (т.е. огромного числа кадров). При небольшом числе кадров вполне приемлемым оказывается значение 1000, т.е. 1 кадр в секунду.

Важнейшее событие таймера – Tick,генерируется при истечении интервала срабатывания таймера. Вобработчик этого события помещается код смены изображения. Старт таймера осуществляется с помощью метода Start, остановка – с помощью метода Stop.

Сами изображения могут помещаться в элемент управления ImageList с помощью изменения значения свойства Images. Порядковые номера рисунков (индексы) начинаются с нуля.

**Пример.** Пусть в ImageList имеется 5 изображений. Обеспечить их смену с частотой 1 кадр в секунду в элементе управленияPictureBox. Объявим переменную img типа Integer с атрубутомPublic, в которую будет помещён порядковый номер изображения. Необходимо создать форму, нанести на неё кнопку, элементы PictureBox, ImageList, и Timer. На кнопку необходимо нанести надпись, например «Начать показ». Затем необходимо установить значение свойства Interval у элемента Timer, равным 1000. После этого необходимо поместить 5 изображений в ImageList.

Вобработчик события Timer1\_Tick необходимо записать следующий код:

**Visual Basic .Net:**

img = img + 1

PictureBox1.Image = ImageList1.Images(img)

If img = 4 Then img = -1

**Visual C#:**

img = img + 1;

pictureBox1.Image=imageList1.Images[img];

if (img==4) img=-1;

Последняя строка обеспечивает цикличность демонстрации, как только будет продемонстрировано последнее изображение, будет автоматически осуществлён переход к первому.

Вобработчик события нажатия кнопки необходимо записать код старта таймера:

**Visual Basic .Net:**

Timer1.Start()

**Visual C#:**

timer1.Start();

Остановка таймера будет осуществлена при закрытии формы. Можно обеспечить остановку с помощью программного кода.

Таймер можно стартовать автоматически. Для этого нужно установить его свойство Enabled, равное True.

Недостатками такого способа являются ограничение числа изображений, которое можно загрузить в ImageList (определяется объёмом памяти), а также то, что размер изображения не может превышать 256 х 256 пикселей. Поэтому чаще всего изображения для анимации помещают в определённую папку, из которой затем выбирают их.

Для работы с папками компьютера используют специальный объект Directoty. Этот объект принадлежит базовому классу System.IO, который необходимо импортировать в проект. Для получения списка файлов в заданной папке используется метод GetFiles, имеющий синтаксис:

**Visual Basic .Net:**

Files=Directory.GetFiles(имяпапки, фильтр)

**Visual C#:**

Files=Directory.GetFiles(имяпапки, фильтр);

Имя папки следует указывать полностью, для C# в качестве разделителя папок используется двойной слеш, фильтр является необязательным параметром и задаёт шаблон для отбора файлов, например, по расширению. Переменная, в которую помещается результат, объявляется как динамический массив:

**Visual Basic .Net:**

Dim Files as String()

**Visual C#:**

String [] files;

Размерность массива будет указана автоматически при заполнении его методом GetFiles. Для доступа к списку файлов из всех процедур ему необходимо присвоить атрибут Public в области описаний модуля. Пример использования данного метода:

**Visual Basic .Net:**

files = Directory.GetFiles("C:\DocumentsandSettings\User\Моидокументы\VisualStudio 2005\Projects\графика1\Resources", "\*.jpg")

**Visual C#:**

files=Directory.GetFiles("("C:\\Documents and Settings\\User\\Моидокументы\\Visual Studio 2005\\Projects\\графика1\\Resources"","\*.jpg");

Массив файлов является объектом, его свойство Length определяет число файлов в заданной папке.

**Пример.**Обеспечить в элементе изображения PictureBox смену изображений из некоторой папки с частотой 1 кадр в секунду Объявим переменные img (порядковый номер изображения) и img\_count (число изображений в папке) типа Integer с атрибутом Public. Также объявим с данным атрибутом массив строк, куда поместим имена всех графических файлов из папки:

**Visual Basic .Net:**

Public img, img\_count AsInteger

Public files() AsString

**Visual C#:**

Public int img,img\_count;

Public string [] files;

Необходимо создать форму, нанести на неё элементы PictureBox, Timer и Button с соответствующей надписью. В код обработчика события Click кнопки необходимо записать код, сканирующий заданную папку, выбирающую из неё все файлы изображений, вычисляющий их количество и стартующий таймер:

**Visual Basic .Net:**

files = Directory.GetFiles("C:\графика1", "\*.jpg")

img\_count = files.Length

Timer1.Start()

**Visual C#:**

files=Directory.GetFiles("C:\\графика1","\*.jpg");

img\_count=files.Length;

timer1.Start();

Фильтрация файлов необходима, т.к. даже если в папке имеются только файлы изображений, в ней могут оказаться скрытые неграфические файлы, что может привести к ошибке.

В обработчик события Timer1\_Tick необходимо записать следующий код:

**Visual Basic .Net:**

PictureBox1.Image = Image.FromFile(files(img))

img = img + 1

If img = img\_count Then img = 0

**Visual C#:**

pictureBox1.Image=Image.FromFile(files[img]);

img=img+1;

if (img==img\_count) img=0;

Данный код работает в том случае, если в папке с файлами имеется хоть один графический файл. Но вообще говоря, эту ситуацию необходимо проверять, и в случае отсутствия графических файлов выдавать сообщение об ошибке.

Папка с изображениями здесь задается вручную. Для выбора папки в процессе работы с программой можно воспользоваться элементом управления FolderBrowserDialog. События этого объекта используются редко. Реакция пользователя оценивается значением переменной DialogResult, имя выбранной пользователем папки записывается в качестве значения свойства SelectedPath. Здесь также следует проверять, имеются ли в папке графические файлы или нет, например, сравнивая число обнаруженных графических файлов с нулём. В этом случае код обработчика события нажатия кнопки «Начать показ» будет примерно следующим:

**Visual Basic .Net:**

Dim result As DialogResult

result = FolderBrowserDialog1.ShowDialog()

If (result = Windows.Forms.DialogResult.OK) Then

files = Directory.GetFiles(FolderBrowserDialog1.SelectedPath, "\*.jpg")

img\_count = files.Length

If img\_count = 0 Then

MsgBox("В данной папке нет графических файлов", MsgBoxStyle.OkOnly)

Else

Timer1.Start()

EndIf

EndIf

**Visual C#:**

DialogResult result;

result = folderBrowserDialog1.ShowDialog();

if (result == DialogResult.OK)

{

files = Directory.GetFiles(folderBrowserDialog1.SelectedPath, "\*.jpg");

img\_count = files.Length;

if (img\_count == 0) MessageBox.Show("В данной папке нет графических файлов", "Ошибка!", MessageBoxButtons.OK);

else timer1.Start();

}

С помощью данных методов и объектов можно также осуществлять движение путем перемещения объекта PictureBox по форме или изменяя его размеры.

Замечание: в качестве фильтра для объекта FolderBrowserDialog можно использовать только одно значение, следовательно, можно отбирать только графические данные одного типа.

Оптимальный подбор размеров элемента PictureBox

Если в качестве значения свойства SizeMode элемента управления PictureBox выбрать значение StretchImage, а размеры элемента PictureBox не пропорциональны размерам загружаемого изображения, то изображение по ширине или высоте может исказиться. Особенно это заметно при динамической смене изображения. В этом случае лучше расположить изображение по центру (установить свойство SizeMode, равное CenterImage), а размеры PictureBox изменять динамически. В случае, когда изображение полностью помещается в элементе управления, ничего не происходит. Если же размеры изображения превышают размеры элемента управления, то можно изменять размеры элемента программно. Для этого необходимо написать специальную программу, например:

**Visual Basic .Net:**

Private Sub ResizeImage()

PictureBox1.Width = 594

PictureBox1.Height = 360

If PictureBox1.Image.Width < PictureBox1.Width And PictureBox1.Image.Height < PictureBox1.Height Then

PictureBox1.SizeMode = PictureBoxSizeMode.CenterImage

Else

Dim k AsSingle

If PictureBox1.Image.Width > PictureBox1.Image.Height Then

k = PictureBox1.Image.Width / PictureBox1.Image.Height

PictureBox1.Height = PictureBox1.Width / k

Else

k = PictureBox1.Image.Height / PictureBox1.Image.Width

PictureBox1.Width = PictureBox1.Height / k

End If

End If

End Sub

**Visual C#:**

Private void resizeimage()

{

pictureBox1.Width = 594;

pictureBox1.Height = 360;

if (pictureBox1.Image.Width < pictureBox1.Width & pictureBox1.Image.Height < pictureBox1.Height) pictureBox1.SizeMode = PictureBoxSizeMode.CenterImage;

else

{

Single k;

k = 1;

int k1 = (int) k;

if (pictureBox1.Image.Width > pictureBox1.Image.Height)

{

k1 = pictureBox1.Image.Width / pictureBox1.Image.Height;

pictureBox1.Height = pictureBox1.Width / k1;

}

else

{

k1 = pictureBox1.Image.Height / pictureBox1.Image.Width;

pictureBox1.Width = pictureBox1.Height / k1;

}

}

}

В этой программе в случае, когда один из размеров изображения превысит соответствующий размер элемента управления, размер элемента управления будет изменён в соответствии с пропорциями изображения. Данную программу необходимо вызывать из процедуры загрузки изображения, например, при динамической смене изображения из обработчика события Timer1\_Tick.

Логика данных процедур на разных языках несколько различается. Это связано с тем, что в C# невозможно явное преобразование типов Floatи Singleв Int, а размеры изображения в пикселях как раз имеют тип Int. Поэтому на C# вводится вспомогательная переменная k, которая по умолчанию принимает значение 1, что означает исходное соотношение размеров. Коэффициент преобразования же обозначен как k1, который имеет тип Int, но с помощью переменной kприводится к типу Single.

Работа с буфером обмена

Изображения, содержащиеся в элементе PictureBox, можно копировать в буфер обмена для последующего обмена с другими приложениями. В данный элемент также можно вставлять изображения из буфера обмена. Для этого используется специальный объект ClipBoard. Данный объект не требуется наносить на форму, он обрабатывается только программно.

Для копирования изображения в буфер обмена используется метод SetDataObject. Например, для копирования в буфер обмена изображения, размещённого на элементе PictureBox1, необходимо написать код:

**Visual Basic .Net:**

Clipboard.SetDataObject(PictureBox1.Image)

**Visual C#:**

Clipboard.SetDataObject(pictureBox1.Image);

Этот код может быть помещён, например, в обработчик события Click пункта контекстного меню «Копировать», которое предварительно должно быть создано и привязано к элементу PictureBox1.

Метод SetDataObject может иметь два аргумента. Второй, необязательный, аргумент, представляющий собой логическое выражение, определяет, должно ли помещённое в буфер обмена значение остаться там после завершения работы приложения, которое его туда поместило, например:

**Visual Basic .Net:**

Clipboard.SetDataObject(PictureBox1.Image,False)

**Visual C#:**

Clipboard.SetDataObject(pictureBox1.Image,false);

Для копирования изображения из буфера обмена используется метод GetDataObject. Он возвращает объект IDataObject, у которого имеются три метода:

GetData – считывает содержимое буфера обмена;

GetDataPresent – возвращает логическое значение True, если в буфере обмена содержатся данные указанного типа;

GetFormat – возвращает перечень форматов, поддерживаемых буфером обмена.

Последние два метода необходимы, т.к. в запросе на чтение данных из буфера обмена обязательно нужно указать, какие именно данные необходимо получить. Если там содержатся данные другого типа, необходимо выдать соответствующее сообщение. Например, для чтения из буфера обмена изображения и размещения его на элементе управления PictureBox1 необходимо записать код:

**Visual Basic .Net:**

PictureBox1.Image = Clipboard.GetDataObject.GetData(DataFormats.Bitmap)

**Visual C#:**

IDataObject idata = Clipboard.GetDataObject();

pictureBox1.Image = (Bitmap)idata.GetData(DataFormats.Bitmap);

Перечисление DataFormats содержит константы, представляющие все поддерживаемые буфером обмена форматы – Text,HTML и др. Графическим данным соответствует формат Bitmap. Для проверки данного формата необходимо проверить, вернёт ли следующий код значение True:

**Visual Basic .Net:**

Clipboard.GetDataObject.GetDataPresent(DataFormats.Bitmap)

**Visual C#:**

IDataObject idata = Clipboard.GetDataObject();

idata.GetDataPresent(DataFormats.Bitmap);

**Пример.** Вставить изображение из буфера обмена в элемент управления PictureBox1. Данный код может быть записан, например, в обработчике события Click пункта «Вставить», контекстного меню, которое предварительно должно быть создано и привязано к элементу управления PictureBox:

**Visual Basic .Net:**

If Clipboard.GetDataObject.GetDataPresent(DataFormats.Bitmap) Then

PictureBox1.Image = Clipboard.GetDataObject.GetData(DataFormats.Bitmap)

Else

MsgBox("В буфере обмена не содержится изображений", MsgBoxStyle.OkOnly)

EndIf

**Visual C#:**

IDataObject idata = Clipboard.GetDataObject();

if (idata.GetDataPresent(DataFormats.Bitmap)) pictureBox1.Image = (Bitmap)idata.GetData(DataFormats.Bitmap);

else MessageBox.Show("В буфере обмена не содержится изображений", "Ошибка", MessageBoxButtons.OK);

Сохранение изображений из элемента управления PictureBox

Для сохранения изображений используется метод Save объекта Image, являющегося свойством объекта PictureBox. По умолчанию изображение сохраняется в формате .BMP, например:

**Visual Basic .Net:**

PictureBox1.Image.Save("C:\MyProject\f1.bmp")

**Visual C#:**

pictureBox1.Image.Save("C:\\MyProject\\f1.bmp");

Имя файла необходимо указывать полностью. Для выбора имени файла можно использовать элемент управления SaveFileDialog.

Если необходимо сохранить данные в другом графическом формате, то в данный метод добавляется второй аргумент, определяющий формат. Этот формат является одним из членов перечисления ImageFormat, которое принадлежит классу System.Drawing.Imaging. Этот класс необходимо импортировать в проект или записывать полное имя перечисления, например:

**Visual Basic .Net:**

PictureBox1.Image.Save("C:\My Project\f1.jpg",ImageFormat.Jpeg)

**Visual C#:**

pictureBox1.Image.Save("C:\\MyProject\\f1.jpg",ImageFormat.Jpeg);

Векторная графика в VisualBasic .Net

Векторная графика создаётся с помощью интерфейса для создания графики – GDI (Graphic Design Interface).

GDI – это графическое ядро Windows, в Visual Studio .Net содержится его последняя версия, называемая GDI+, оно используется во всех языках .Net. Одной из важнейших особенностей GDI+ является то, что это графическое ядро не сохраняет информацию о своём состоянии, т.е. каждая следующая операция осуществляется независимо от предыдущей. Например, если нужно нарисовать два отрезка прямой, то второй отрезок не будет автоматически нарисован в том же формате, что и первый.

Объекты GDI+ располагаются в нескольких классах, которые предварительно необходимо импортировать в свой проект. Важнейшим классом является класс System.Drawing, который подключается по умолчанию. К другим классам относятся System.Drawing.Drawing2D, System.Drawing.Imaging и System.Drawing.Text, которые необходимо импортировать в свой проект. Существуют базовые объекты рисования, которые определяют параметры рисунков – цвет, расположение, толщину линии и др., а также специальные объекты для рисования конкретных фигур.

Базовые графические объекты

Базовые графические объекты обязательно используются при создании векторной графики. Самым важным объектом является объект Graphics, представляющий собой поверхность для рисования. У каждого элемента, на котором можно рисовать, имеется свойство Graphics, возвращающее объект Graphics. Объект Graphics представляет собой набор методов, необходимых для рисования на поверхности элемента управления. Для создания объекта Graphics, связанного с конкретным элементом управления, необходимо вызвать метод СreateGraphics. Чаще всего рисование осуществляется на форме, для этого необходимо записать код:

**Visual Basic .Net:**

Dim g As Graphics

g = Me.CreateGraphics

**Visual C#:**

Graphics g;

g = this.CreateGraphics();

Для элемента управления, например, для кнопки, код будет следующим:

**Visual Basic .Net:**

Dim g As Graphics

g = Button1.CreateGraphics

**Visual C#:**

Graphics g;

g = button1.CreateGraphics();

Чаше всего рисование осуществляется на форме.

Все размеры выводимых на объекте Graphics изображений по умолчанию измеряются в пикселях. Если необходимо сменить единицу измерения, то можно воспользоваться свойствомPageUnit объекта Graphics, например (хотя это используется очень редко):

**Visual Basic .Net:**

g.PageUnit = GraphicsUnit.Millimeter

**Visual C#:**

g.PageUnit = GraphicsUnit.Millimeter;

Двумя важнейшими инструментами для рисования являются перо (Pen) и кисть (Brush).

С помощью перьев рисуют геометрические фигуры, состоящие из линий (контуры) – прямые, кривые, прямоугольные и др. Важнейшими характеристиками объекта Pen являются цвет и толщина линии, оставляемые пером при рисовании. Цвет определяется объектом Color, толщина линии задается в пикселях, например, для создания красного пера толщиной в три пикселя необходимо написать код:

**Visual Basic .Net:**

Dim redpen As Pen

redpen = New Pen(Color.Red, 3)

**Visual C#:**

Pen redpen;

redpen = new Pen(Color.Red, 3);

Если толщина линии опущена, то по умолчанию она равна 1 пикселю.

Кистями рисуют заполненные фигуры. Главными характеристиками кисти являются цвет и образец заполнения фигуры. Собственно говоря, в чистом виде объекта Brush не существует, т.к. это абстрактный класс, наследуемый кистями различных типов.

Основными кистями являются:

SolidBrush(принадлежит классу Drawing) – заполняет фигуру одним цветом, который определяется объектом Color, например:

**Visual Basic .Net:**

Dim redbrush As SolidBrush

redbrush = New SolidBrush(Color.Red)

**Visual C#:**

SolidBrush redbrush;

redbrush = new SolidBrush(Color.Red);

HatchBrush (принадлежит классу Drawing2d) – заполняет фигуру шаблоном из штрихов, задаётся способом штриховки и двумя объектами Color – цветом линий и цветом фона, например:

**Visual Basic .Net:**

Dim hbrush As HatchBrush

hbrush = New HatchBrush(HatchStyle.DarkVertical, Color.Black, Color.Red)

**Visual C#:**

HatchBrush hbrush;

hbrush = new HatchBrush(HatchStyle.DarkVertical, Color.Black, Color.Red);

Кроме этого, существуют градиентные и текстурные кисти. Градиентные заполняют фигуру переходом цветов, а текстурные – цветом, созданным на основе растрового изображения.

Для определения координат рисованных объектов используется объект Point, представляющий собой точку на рабочей поверхности, определяемую парой координат (*х* и *y*). Отсчёт начала координат (точка (0,0) идёт от левого верхнего угла объекта (формы и др.). Для создания нового объекта Point необходимо указать его координаты, представленные свойствами X и Y, например:

**Visual Basic .Net:**

Dim p1 As Point

p1 = New Point(10, 20)

**Visual C#:**

Point p1;

p1 = new Point(10, 20);

или

**Visual Basic .Net:**

Dim p1 As Point

p1.X = 100

p1.Y = 200

**Visual C#:**

Point p1;

p1.X = 100;

p1.Y = 200;

Координаты измеряются в пикселях и имеют тип Integer. Если используются единицы измерения, отличные от пикселей, то координаты могут иметь тип Single, а вместо объекта Point используется объект PointF.

Рисование геометрических фигур

Геометрические фигуры рисуются с помощью специальных методов объекта Graphics. Следует отметить, что все изображения, созданные с помощью этих методов, являются непостоянными, и при любых действиях с объектом (изменении размера формы, переключении в другую форму, закрытии формы и др.) данные изображения теряются.

За этой ситуацией необходимо следить, и, при необходимости, повторять коды рисования в обработчиках событий Resize, GotFocus и других. Однако, если изображение формируется динамически, то данный код может быть очень сложным. Чтобы изображение выводилось на форме при её вызове на экран, соответствующий код необходимо записывать в обработчик события Paint, а не Load.

Изображение, нанесённое на форму, сохранить нельзя.

Графические методы, используемые для рисования, можно разделить на две группы: методы для рисования контуров геометрических фигур и для рисования заполненных фигур. Имена первой группы методов начинаются с префикса Draw, имена второй группы методов – с префикса Fill. Для некоторых методов из первой группы не существует эквивалентов второй группы, например, нельзя заполнить линию или незамкнутую кривую.

Для заполнения фигур используется объект Brush, для вычерчивания их контуров –объект Pen.

Прямая линия

Создаётся с помощью метода DrawLine по координатам двух точек. Аргументами являются объект Pen и либо 4 координаты точек, заданные явно, либо два объекта Point, например:

**Visual Basic .Net:**

Dim g As Graphics

g = Me.CreateGraphics

Dim redpen As Pen

redpen = New Pen(Color.Red, 3)

g.DrawLine(redpen, 10, 10, 400, 400)

**Visual C#:**

Graphics g;

g = this.CreateGraphics();

Pen redpen;

redpen = new Pen(Color.Red, 3);

g.DrawLine(redpen, 10, 10, 400, 400);

или:

**Visual Basic .Net:**

Dim g As Graphics

g = Me.CreateGraphics

Dim redpen As Pen

redpen = New Pen(Color.Red, 3)

Dim p1, p2 As Point

p1 = New Point(10, 10)

p2 = New Point(400, 400)

g.DrawLine(redpen, p1, p2)

**Visual C#:**

Graphics g;

g = this.CreateGraphics();

Pen redpen;

redpen = new Pen(Color.Red, 3);

Point p1, p2;

p1 = new Point(10, 10);

p2 = new Point(400, 400);

g.DrawLine(redpen,p1,p2);

Можно нарисовать несколько линий за одинраз (разомкнутую ломанную лиию) с помощью метода DrawLines. Для этого создаётся массив объектов Point, который передаётся в качестве аргумента данного метода. При этом первая линия соединяет первую и вторую точки, вторая – вторую и третью и т.д., например:

**Visual Basic .Net:**

Dim points As Point() = {New Point(10, 10), New Point(10, 100), New Point(200, 50), New Point(250, 300)}

g.DrawLines(redpen, points)

**Visual C#:**

Point[] points={new Point(10,10), new Point(10,100), new Point(200,50), new Point(250,300)};

g.DrawLines(redpen, points);

Прямоугольник

Контур прямоугольника рисуется с помощью метода DrawRectangle. Прямоугольник задаётся координатами левого верхнего угла и размерами – шириной и высотой. При этом может быть предварительно создан объект Rectangle, либо явно заданы его характеристики, например:

**Visual Basic .Net:**

Dim g As Graphics

g = Me.CreateGraphics

Dim redpen As Pen

redpen = New Pen(Color.Red, 3)

g.DrawRectangle(redpen, 10, 10, 200, 100)

**Visual C#:**

Graphics g;

g = this.CreateGraphics();

Pen redpen;

redpen = new Pen(Color.Red, 3);

g.DrawRectangle(redpen, 10, 10, 200, 100);

Заполненный прямоугольник создаётся с помощью метода FillRectangle. При этом первым аргументом является объект Brush, например:

**Visual Basic .Net:**

Dim g As Graphics

g = Me.CreateGraphics

Dim redbrush As SolidBrush

redbrush = New SolidBrush(Color.Red)

g.FillRectangle(redbrush, 10, 10, 200, 100)

**Visual C#:**

Graphics g;

g = this.CreateGraphics();

SolidBrush redbrush;

redbrush = new SolidBrush(Color.Red);

g.FillRectangle(redbrush, 10, 10, 200, 100);

Эллипс

Эллипсы создаются с помощью методов DrawEllipse и FillEllipse. Эллипс определяется, как овал, вписанный в прямоугольник. При этом также может быть предварительно создан объект Rectangle, либо явно заданы его характеристики. Синтаксис данных методов аналогичен синтаксису аналогичных методов для прямоугольников.

Сегмент эллипса может быть создан с помощью методов DrawPie и FillPie. Он состоит из дуги и двух прямоугольных отрезков, соединяющих конечные точки дуги с центром круга или эллипса. Данные методы, кроме характеристик пера или кисти и прямоугольника, в который вписан эллипс, принимают два дополнительных аргумента – начальный угол и внутренний угол сегмента. Углы измеряются в градусах, их значения увеличиваются по часовой стрелке. Угол, равный 0, соответствует горизонтальной оси координат, значение внутреннего угла прибавляется к значению начального угла, например:

**Visual Basic .Net:**

Dim g As Graphics

g = Me.CreateGraphics

Dim redpen As Pen

redpen = New Pen(Color.Red, 3)

g.DrawPie(redpen, 10, 10, 200, 200, 0, 60)

**Visual C#:**

Graphics g;

g = this.CreateGraphics();

Pen redpen;

redpen = new Pen(Color.Red, 3);

g.DrawPie(redpen, 10, 10, 200, 200, 0, 60);

Многоугольник

Многоугольник представляет собой замкнутую ломанную линию. Он создается с помощью методов DrawPolygon и FillPolygon. Аргументы и синтаксис данных методов аналогичны аргументам и синтаксису методов DrawLines, например:

**Visual Basic .Net:**

Dim g As Graphics

g = Me.CreateGraphics

Dim redpen As Pen

redpen = New Pen(Color.Red, 3)

Dim points As Point() = {New Point(10, 10), New Point(10, 100), \_

New Point(200, 50), New Point(250, 300)}

g.DrawPolygon(redpen, points)

**Visual C#:**

Graphics g;

g = this.CreateGraphics();

Pen redpen;

redpen = new Pen(Color.Red, 3);

Point[] points = { new Point(10, 10), new Point(10, 100), new Point(200, 50), new Point(250, 300) };

g.DrawPolygon(redpen, points)

Начальную точку в конце массива повторять не обязательно, данный метод автоматически соединит первую и последнюю точки массива.

Сплайны

Сплайн представляет собой гладкую незамкнутую кривую. Понятие сплайна пришло из инженерной графики, где под сплайном понимался гибкий прут, с помощью которого конструкторы вычерчивали кривые, изгибая его на плоскости вокруг торчащих из неё гвоздей. Сплайн должен пройти через фиксированный набор точек так, чтобы получающаяся линия была гладкой и не имела изломов. Форма линии зависит от гибкости прута. Если прут окажется недостаточно гибким, он может сломаться.

В современной компьютерной графике для вычерчивания сплайнов используются специальные формулы, в которых учитывается гибкость сплайна. Более гибкий сплайн имеет более крутые изгибы, а менее гибкий не так легко огибает ключевые точки.

Сплайн создаётся с помощью метода DrawCurve. При этом определяется перо, ключевые точки сплайна и его упругость. Если упругость сплайна равно 0, то это значит, что он абсолютно упругий, как резиновая лента. Сплайн с упругостью 0 представляет собой ломанную линию и фактически не является кривой. Чем выше упругость, тем более гладкой получается кривая. Значение упругости задавать не обязательно, по умолчанию она равна 1.

Гибкость сплайна имеет тип Single и может иметь дробное значение. Пример использования метода DrawCurve:

**Visual Basic .Net:**

Dim g As Graphics

g = Me.CreateGraphics

Dim redpen As Pen

redpen = New Pen(Color.Red, 3)

Dim points As Point() = {New Point(10, 10), New Point(10, 100), \_

New Point(200, 50), New Point(250, 300)}

g.DrawCurve(redpen, points, 0.5)

**Visual C#:**

Graphics g;

g = this.CreateGraphics();

Pen redpen;

redpen = new Pen(Color.Red, 3);

Point[] points = { new Point(10, 10), new Point(10, 100), new Point(200, 50), new Point(250, 300) };

g.DrawCurve(redpen, points, 0.5f);

Можно строить сплайн не по всему массиву точек, а по его части. В этом случае задается сам массив, номер начальной точки и количество точек. Массив точек должен содержать не менее 4 элементов, иначе применение этого метода не имеет смысла, например:

**Visual Basic .Net:**

Dim p1(6) As Point

p1(0) = New Point(10, 10)

p1(1) = New Point(15, 25)

p1(2) = New Point(35, 50)

p1(3) = New Point(45, 80)

p1(4) = New Point(87, 90)

p1(5) = New Point(100, 100)

g.DrawCurve(redpen, p1, 1, 3, 2)

g.DrawCurve(Pens.Blue, p1, 1, 3, 3)

**Visual C#:**

Point[] p1=new Point[6];

p1[0] = new Point(10, 10);

p1[1] = new Point(15, 25);

p1[2] = new Point(35, 60);

p1[3] = new Point(45, 80);

p1[4] = new Point(87, 90);

p1[5] = new Point(100, 100);

g.DrawCurve(redpen, p1, 1, 3, 2);

g.DrawCurve(Pens.Blue, p1, 1, 3, 3);

Следует иметь в виду, что математический сплайн может иметь любую упругость, хотя при большом её значении форма сплайна получается довольно неожиданной. Физический сплайн при такой упругости может просто сломаться.

Работа с мышью

Система VisualStudio .Netпозволяет отслеживать события мыши. Для этого используются события MouseClick и MouseDoubleClick. Эти события имеют аргумент e, который, в свою очередь, имеет свойства и методы. Например, свойство e.Button позволяет определить, какая кнопка мыши нажата – правая или левая, а свойства e.X и e.Y позволяют определить координаты мыши в момент нажатия кнопки. Это можно использовать для нанесения точек на форму. Сами точки могут быть представлены в виде прямоугольников или кругов размером 1 пиксель. Эти точки можно поместить в динамический массив точек и в дальнейшем использовать для рисования кривых и многоугольников.

Рассмотрим пример. Пусть при щелчке левой кнопкой мыши по форме на форме отражается точка синего цвета, при этом её координаты запоминаются в массиве точек, размерность которого автоматически увеличивается с каждой новой точкой. Код обработчика события MouseClick будет следующим:

**Visual Basic .Net:**

If e.Button = Windows.Forms.MouseButtons.Left Then

Dim p AsNew Rectangle(e.X, e.Y, 1, 1)

Dim mouse\_point As New Point(e.X, e.Y)

g.DrawEllipse(Pens.Blue, p)

num\_of\_points += 1

ReDim Preserve point\_array(num\_of\_points)

point\_array(num\_of\_points) = mouse\_point

EndIf

**Visual C#:**

if (e.Button==MouseButtons.Left)

{

Rectangle p = new Rectangle(e.X, e.Y, 1, 1);

Point mouse\_point = new Point(e.X, e.Y);

g.DrawEllipse(Pens.Blue, p);

num\_of\_points += 1;

Array.Resize(ref point\_array, num\_of\_points);

point\_array[num\_of\_points-1] = mouse\_point;

}

Массив и переменную num\_of\_points, в которую записывается число нанесенных точек, необходимо определить в области описаний модуля, чтобы он был доступен всем процедурам формы и в дальнейшем по нанесённым точкам можно было бы что-нибудь нарисовать:

**Visual Basic .Net:**

Dim point\_array() As Point

Dim num\_of\_point as integer

**Visual C#:**

Point[] point\_array;

int num\_of\_points;

Переменная g также здесь определена на уровне формы, а поверхность для рисования создана в обработчике события загрузки формы. Поскольку в языке C# отсутствует оператор Redim, то для изменения размера массива используются методы класса Math. В программе на Visual Basic .Net. их можно также использовать.

Кривые Безье

Кривые Безье отличаются от фундаментальных сплайнов большей гладкостью. Кривы́е Безье́ были разработаны в 60-х годах XX века независимо друг от друга Пьером Безье из автомобилестроительной компании «[Рено](http://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BD%D0%BE)» и Полем де Кастелье из компании«Ситроен», где применялись для проектирования кузовов автомобилей.

Кривая Безье задается двумя главными опорными точками (начало и конец кривой), а также рядом промежуточных опорных точек, через которые кривая не проходит. Промежуточные точки играют роль магнитов. Кривая начинается в одной начальной точке и направляется ко второй, по мере движения притягиваясь к контрольным точкам. Сначала преобладает влияние первой точки, но по мере приближения ко второй точке более сильным становится влияние контрольных точек.

Кривая Безье — параметрическая кривая, задаваемая выражением

 0<t<1

где Pi - функция компонент векторов опорных вершин, а bi,n(t) - базисные функции кривой Безье, называемые также [полиномами Бернштейна](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B8%D0%BD%D0%BE%D0%BC_%D0%91%D0%B5%D1%80%D0%BD%D1%88%D1%82%D0%B5%D0%B9%D0%BD%D0%B0).





где n — степень полинома, i - порядковый номер опорной точки.

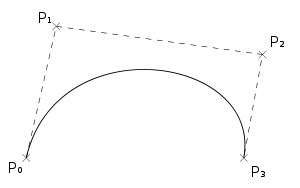
Степень полинома определяет необходимое количество промежуточных вершин, и, соответственно, порядок кривой. Кривая первого порядка (линейная) (n=1) определяется только двумя основными вершинами и не требует промежуточных вершин. Фактически она представляет собой прямую от начальной до конечной точки. Наибольшее применение имеют квадратичные (n=2) и кубические (n=3) кривые. Квадратичные кривые задаются двумя конечными точками (P0 и P2) и одной промежуточной точкой (P1).



Квадратичные кривые Безье в составе используются для описания формы символов в шрифтах [TrueType](http://ru.wikipedia.org/wiki/TrueType).

Кубические кривые (n=3)задаются двумя опорными вершинами (P0 и P3) и двумя промежуточными (P1 и P2). Они описываются следующим уравнением:



[](http://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Bezier_curve.svg)

Линия берёт начало из точки P0 направляясь к P1 и заканчивается в точке P3 подходя к ней со стороны P2. То есть кривая не проходит через точки P1 и P2, они используются для указания её направления. Длина отрезка между P0 и P1 определяет, как скоро кривая повернёт к P3.

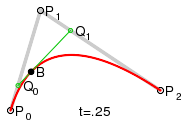
В матричной форме кубическая кривая Безье записывается следующим образом:



где MBназывается базисной матрицей Безье:



Принцип построения квадратичной кривой иллюстрирует следующий рисунок:

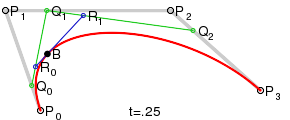
[](http://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Bezier_quadratic.png)

Точка Q0 изменяется от P0 до P1 и описывает линейную кривую Безье.

Точка Q1 изменяется от P1 до P2 и также описывает линейную кривую Безье.

Точка B0 изменяется от Q0 до Q1 и описывает квадратичную кривую Безье.

Принцип построения кубической кривой иллюстрирует следующий рисунок:

[](http://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Bezier_cubic.png)

Точки Q0, Q1 и Q2, описывают линейные кривые, а точки R0 и R1 описывают квадратичные кривые:

Visual Studio .Net содержит методы, позволяющие строить кубические кривые Безье. Для этого используется метод DrawBezier, аргументами которого являются объект Pen и координаты четырёх точек (двух опорных и двух промежуточных) или четыре объекта Point, например:

**Visual Basic .Net:**

Dim g As Graphics

g = Me.CreateGraphics

Dim redpen As Pen

redpen = New Pen(Color.Red, 3)

g.DrawBezier(redpen, 10, 10, 40, 40, 100, 100, 200, 300)

**Visual C#:**

Graphics g;

g = this.CreateGraphics();

Pen redpen;

redpen = new Pen(Color.Red, 3);

g.DrawBezier(redpen, 10, 10, 40, 40, 100, 100, 200, 300);

Сами контрольные точки можно нарисовать с помощью метода FillRectangle с аргументами, соответствующими одной точке.

Замкнутые кривые

Замкнутые кривые (замкнутые сплайны) рисуются по тому же принципу, что и многоугольники, т.е. задается перо или кисть, а также набор точек, через которые должна пройти кривая. Для этого используются методы DrawClosedCurve и FillClosedCurve, например:

**Visual Basic .Net:**

Dim g As Graphics

g = Me.CreateGraphics

Dim redpen As Pen

redpen = New Pen(Color.Red, 3)

Dim points As Point() = {New Point(10, 10), New Point(10, 100),New Point(200, 50), New Point(250, 300)}

g.DrawClosedCurve(redpen, points)

**Visual C#:**

Graphics g;

g = this.CreateGraphics();

Pen redpen;

redpen = new Pen(Color.Red, 3);

Point[] points = { new Point(10, 10), new Point(10, 100), new Point(200, 50), new Point(250, 300) };

g.DrawClosedCurve(redpen, points);

Изображения

Изображения выводятся в заданной области объекта Graphics с помощью метода DrawImage. Само изображение задаётся с помощью объекта Image также, как и при создании объекта PictureBox. Метод DrawImage имеет несколько форм. Самой простой формой прорисовки изображения является задание координат его левого верхнего угла и размеров, например:

**Visual Basic .Net:**

Dim img As Image

img = Image.FromFile("C\pic1.jpg")

Dim g As Graphics

g = Me.CreateGraphics

g.DrawImage(img, 0, 0, 200, 200)

**Visual C#:**

Image img;

img = Image.FromFile("C:\\pic1.jpg");

Graphics g;

g = this.CreateGraphics();

g.DrawImage(img, 0, 0, 200, 200);

Этот код рисует заданное изображение размером 200 х 200 пикселей, начиная с левого верхнего угла формы. Если размеры изображения в объекте Imageне соответствуют третьему и четвёртому аргументам объекта DrawImage, то изображение увеличивается или уменьшается. Вообще третий и четвёртый аргументы данного метода не являются обязательными, если они не указаны, то изображение создается в своих реальных размерах, а в случае, если размеры объекта, на котором оно рисуется, меньше, то изображение обрезается.

Можно рисовать изображение в заданном прямоугольнике, например:

**Visual Basic .Net:**

Dim img As Image

img = Image.FromFile("C\pic1.jpg")

Dim g As Graphics

g = Me.CreateGraphics

Dim r1 AsNew Rectangle(100, 100, 100, 100)

g.DrawImage(img, r1)

**Visual C#:**

Image img;

img = Image.FromFile("C:\\pic1.jpg ");

Graphics g;

g = this.CreateGraphics();

Rectangle r1 = new Rectangle(100, 100, 100, 100);

g.DrawImage(img, r1);

Чтобы при этом не произошло искажение изображения, пропорции прямоугольника должны быть такими же, как пропорции изображения. Если размеры прямоугольника больше или меньше размеров изображения, то, как в предыдущем случае, оно увеличивается или уменьшается.

Третья форма метода DrawImage позволяет создать изображение, изменив его форму в виде параллелограмма. В качестве аргумента задаётся массив из трёх точек, соответствующих левому верхнему, правому верхнему и левому нижнему углам параллелограмма. Координаты правого нижнего угла вычисляются автоматически, например:

**Visual Basic .Net:**

Dim img AsImage

img = Image.FromFile("C\pic1.jpg")

Dim g As Graphics

g = Me.CreateGraphics

Dim points() As Point = {New Point(100, 100), New Point(550, 100), New Point(150, 250)}

g.DrawImage(img, points)

**Visual C#:**

Image img;

img = Image.FromFile("C:\\ pic1.jpg");

Graphics g;

g = this.CreateGraphics();

Point[] points = { new Point(100, 100), new Point(550, 100), new Point(150, 250) };

g.DrawImage(img, points);

Можно указать также часть рисунка, которую нужно вырезать и отобразить в заданном параллелограмме. Для этого задаются дополнительные аргументы, определяющие вырезаемый прямоугольник и единицы его измерения, например:

**Visual Basic .Net:**

Dim img As Image

img = Image.FromFile("C\pic1.jpg")

Dim g As Graphics

g = Me.CreateGraphics

Dim points() As Point = {New Point(100, 100), New Point(325, 100), New Point(150, 250)}

Dim r1 AsNew Rectangle(10, 10, 50, 50)

g.DrawImage(img, points,r1, GraphicsUnit.Pixel)

**Visual C#:**

Image img;

img = Image.FromFile("C:\\ pic1.jpg");

Graphics g;

g = this.CreateGraphics();

Point[] points = { new Point(100, 100), new Point(550, 100), new Point(150, 250) };

Rectangle r1 = new Rectangle(10, 10, 50, 50);

g.DrawImage(img, points, r1, GraphicsUnit.Pixel);

Этот код вырезает из изображения квадрат размером 50 х 50 пикселей, начиная с точки (10,10) и отображает его в параллелограмме, задаваемом точками массива points. Следует отметить, что параметры прямоугольника в данном случае относятся к изображению (т.е. к объекту Image), а не к объекту Graphics. Если задать размеры прямоугольника, равные размеру изображения, а координаты точки – (0,0), то изображение выведется таким же образом, как в предыдущем случае.

С помощью данного метода можно также управлять гаммой цветов изображения. Для этого используется аргумент атрибутов изображения, который определяется как объект ImageAttributes. У этого объекта имеется метод SerGamma, который и устанавливает гамму цветов. Гамма определяется как значение типа Single. По умолчанию значение гаммы равно 1, увеличение гаммы делает цвета более тёмными, уменьшение – более светлыми. Следует отметить, что понятия гаммы и яркости являются различными. При изменении гаммы учитывается весь диапазон цветов изображения, но при этом на разные цвета изменение гаммы влияет по разному, а также изменяется контрастность. Рассмотрим пример:

**Visual Basic .Net:**

Dim img As Image

img = Image.FromFile("C\pic1.jpg")

Dim g As Graphics

g = Me.CreateGraphics

Dim points() As Point = {New Point(100, 100), New Point(325, 100), New Point(150, 250)}

Dim r1 AsNew Rectangle(10, 10, 50, 50)

Dim atr AsNew ImageAttributes

atr.SetGamma(0.1)

g.DrawImage(img, points,r1, GraphicsUnit.Pixel, atr)

**Visual C#:**

Image img;

img = Image.FromFile("C:\\ pic1.jpg");

Graphics g;

g = this.CreateGraphics();

Point[] points = { new Point(100, 100), new Point(550, 100), new Point(150, 250) };

Rectangle r1 = new Rectangle(10, 10, 50, 50);

ImageAttributes atr = new ImageAttributes();

atr.SetGamma(0.1f);

g.DrawImage(img, points, r1, GraphicsUnit.Pixel, atr);

Этот код рисует фрагмент изображения также, как и в предыдущем случае, а также изменяет гамму цветов до значения 0.1, т.е. делает изображение достаточно бледным.

Использование метода DrawImage для рисования сложнее создания изображений на элементе PictureBox. Кроме этого, при его использовании, как и при рисовании с помощью других графических методов, необходимо учитывать возможность изменения размеров формы, многократного её вызова на экран и т.д. Однако и возможности данного метода значительно шире. Например, изображение можно выводить по частям, или постепенно изменяя его гамму цветов – от самого бледного до исходного.

Комбинация графических изображений

Такая комбинация создаётся с помощью объекта GraphicsPath. Можно создать любое количество прямоугольников, линий, эллипсов и других элементов и поместить их в объект GraphicsPath. Все элементы объекта GraphicsPath обведены по контуру одним способом, который определяется при выводе объекта GraphicsPath. Использовать разную окраску разных элементов объекта GraphicsPath нельзя.

Добавить новые элементы в объект Path можно с помощью методов:

AddLine – добавляет прямую линию;

AddRectangle – добавляет прямоугольник (добавление прямоугольника возможно только с использованием нового объекта Rectangle, по точкам добавлять нельзя);

AddEllipse – добавляет эллипс;

AddPolygon – добавляет многоугольник (замкнутую ломанную);

AddCurve – добавляет сплайн (разомкнутую кривую);

AddBezier – добавляет новую кривую Безье;

AddPie– добавляет новый сегмент эллипса и др.

Синтаксис данных методов аналогичен синтаксису методов для вызова соответствующих геометрических фигур, за исключением того, что не задаётся первый аргумент – объект Pen.

При рисовании с помощью объекта GraphicsPathиспользуется метод DrawPath, у которого имеются два аргумента – перо и выводимый объект GraphicsPath.

Следующий код иллюстрирует возможность вывода нового изображения с помощью объекта GraphicsPath, состоящего из эллипса, прямоугольника, линии, многоугольника, замкнутой и разомкнутой кривой.

**Visual Basic .Net:**

Dim g As Graphics

g = Me.CreateGraphics

Dim redpen As Pen

redpen = New Pen(Color.Red, 3)

Dim path1 AsNew GraphicsPath

path1.AddEllipse(10, 10, 200, 100)

Dim r1 AsNew Rectangle(25, 25, 100, 100)

path1.AddRectangle(r1)

path1.AddLine(10, 10, 400, 400)

Dim points As Point() = {New Point(10, 10), New Point(10, 100), New Point(200, 50), New Point(250, 300)}

path1.AddPolygon(points)

path1.AddCurve(points, 0.5)

path1.AddClosedCurve(points, 0.5)

g.DrawPath(redpen, path1)

**Visual C#:**

Graphics g;

g = this.CreateGraphics();

Pen redpen;

redpen = new Pen(Color.Red, 3);

GraphicsPath path1 = new GraphicsPath();

path1.AddEllipse(10, 10, 200, 100);

Rectangle r1 = new Rectangle(25, 25, 100, 100);

path1.AddRectangle(r1);

path1.AddLine(10, 10, 400, 400);

Point[] points = { new Point(10, 10), new Point(10, 100), new Point(200, 50), new Point(250, 300) };

path1.AddPolygon(points);

path1.AddCurve(points, 0.5f);

path1.AddClosedCurve(points, 0.5f);

g.DrawPath(redpen, path1);

Можно в один объект GraphicsPath добавлять другой объект с помощь метода Addpath. Данный метод имеет два аргумента – добавляемый объект GraphicsPath и логическое значение, равное True, если первая фигура в добавленном объекте является частью последней фигуры в данном объекте и False, если первая фигура в добавленном контуре отделена от последней фигуры в добавляемом объекте.

Сформировав один раз объект GraphicsPath, можно нарисовать его несколько раз в разных местах экрана или разными перьями), можно объявить его вне кода процедур с атрибутом Public и добавлять в него фигуры с помощью разных кнопок, а затем отслеживать его изменение при закрытии формы или её изменении.

Создание фигурных изображений

Фигурные изображения создаются с помощью технологии вырезания и создания масок. Маской называется любой замкнутый контур, ограничивающий область, на которой можно рисовать. Если создать маску в объекте Graphics, то, какое бы изображение не было создано, на экране останется только та его часть, которая ограничена этой маской. Маска создаётся с помощью метода SelClip объекта Graphics. Этот метод имеет два аргумента: обязательный – форма маски и необязательный – определяющий, как новая вырезаемая область будет объединена с предыдущей. Форма маски может быть в виде прямоугольника – Rectangle и в виде любого произвольного контура – GraphicPath. Второй аргумент – CombineMode может принимать пять вариантов значений из предлагаемого системой списка. Этот аргумент используется редко.

**Пример.** Отобразить на форме круглое изображение, созданное из заданного графического файла, в круге, вписанном в прямоугольник с координатами левого верхнего угла (50,50) и размерами 150 х 150 пикселей. Поскольку методы вырезания круга отсутствуют, то такой круг необходимо создать на основе объекта Path:

**Visual Basic .Net:**

Dim img As Image

img = Image.FromFile("C\pic1.jpg")

Dim g As Graphics

g = Me.CreateGraphics

Dim r1 AsNew Rectangle(50, 50, 150, 150)

Dim p1 AsNew GraphicsPath

p1.AddEllipse(r1)

g.SetClip(p1)

g.DrawImage(img, 50, 50, 150, 150)

**Visual C#:**

Image img;

img = Image.FromFile("C:\\ pic1.jpg");

Graphics g;

g = this.CreateGraphics();

Rectangle r1 = new Rectangle(50, 50, 150, 150);

GraphicsPath p1 = new GraphicsPath();

p1.AddEllipse(r1);

g.SetClip(p1);

g.DrawImage(img, 50, 50, 150, 150);

Очистка объекта Graphics

Содержимое объекта Graphics можно удалить с помощью метода Clear. При этом задаётся цвет, которым закрашивается объект, например:

**Visual Basic .Net:**

Dim g As Graphics

g = Me.CreateGraphics

g.Clear(Me.BackColor)

**Visual C#:**

Graphics g;

g = this.CreateGraphics();

g.Clear(this.BackColor);

Данный пример очищает объект Graphics, связанный с формой и закрашивает его фоновым цветом формы.

Аффинные преобразования графических изображений

Аффинные преобразования - это преобразования (плоскости или пространства), при которых прямые переходят в прямые, и сохраняется их параллельность (в частности, преобразования подобия), например:

В компьютерной графике используются три типа преобразования графики – масштабирование, перенос и поворот. Эти преобразования 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.

Поворот вокруг заданной точки осуществляется с использованием метода RotateAt объекта Matrix. Аргументами данного метода являются угол поворота, выраженный в градусах и координаты точки, вокруг которой осуществляется поворот. При этом сам код рисования остается неизменным.

**Пример.** Нарисовать на форме прямоугольник размером 100 х 100 пикселей с координатами верхнего левого угла (10,10) коричневым пером толщиной в 3 пикселя а затем повернуть его на 15 градусов. Код, рисующий прямоугольник в форме при загрузке необходимо записать в обработчик события Form\_Paint:

**Visual Basic .Net:**

Dim path As New GraphicsPath

Dim rect As Rectangle = New Rectangle(10, 10, 100, 100)

Dim g As Graphics

g = Me.CreateGraphics

path.AddRectangle(rect)

g.DrawPath(New Pen(Color.Brown, 3), path)

**Visual C#:**

GraphicsPath path = new GraphicsPath();

Rectangle rect = new Rectangle(10, 10, 100, 100);

Graphics g;

g = this.CreateGraphics();

path.AddRectangle(rect);

g.DrawPath(new Pen(Color.Brown, 3), path);

Код, обеспечивающий поворот, необходимо написать, например, в обработчике события Click кнопки с надпистью «Поворот» (координаты центра прямоугольника находятся в точке (60,60).

**Visual Basic .Net:**

Dim path As New GraphicsPath

Dim rect As Rectangle = New Rectangle(10, 10, 100, 100)

Dim g As Graphics

g = Me.CreateGraphics

path.AddRectangle(rect)

Dim matr1 As New Matrix

matr1.RotateAt(15, New Point(60, 60))

path.Transform(matr1)

g.DrawPath(New Pen(Color.Brown, 3), path)

**Visual C#:**

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.RotateAt(15, new Point(60, 60));

path.Transform(matr1);

g.DrawPath(new Pen(Color.Brown, 3), path);

При этом на форме будет отражены и начальное и повёрнутое изображение. Чтобы начальное изображение исчезало, объект Graphics необходимо очищать. С помощью данного метода и с использованием таймера можно, например, обеспечить динамическое вращение рисунка.

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

**Пример.** Переместить прямоугольник разметром 100 х 100 пикселей, с левым верхним углом с координатами (10,10) в точку с координатами (20,20).

**Visual Basic .Net:**

Dim path As New GraphicsPath

Dim rect As Rectangle = New Rectangle(10, 10, 100, 100)

Dim g As Graphics

g = Me.CreateGraphics

path.AddRectangle(rect)

Dim matr1 As New Matrix

matr1.Translate(20, 20)

path.Transform(matr1)

g.DrawPath(New Pen(Color.Brown, 3), path)

**Visual C#:**

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);

Метод Translate может быть также использован для преобразования прямоугольников в параллелограммы. Для этого в качестве аргументов объекта Matrix передаются объект, определяющий прямоугольник и массив из трёх точек, определяющих левый верхний, правый верхний и левый нижний углы параллелограмма, например:

**Visual Basic .Net:**

Dim path As New GraphicsPath

Dim rect As Rectangle = New Rectangle(10, 10, 100, 100)

Dim g As Graphics

g = Me.CreateGraphics

path.AddRectangle(rect)

Dim point() As Point = {New Point(110, 10), New Point(210, 10), New Point(170, 110)}

Dim matr1 AsNew Matrix(rect, point)

path.Transform(matr1)

g.DrawPath(New Pen(Color.Brown, 3), path)

**Visual C#:**

GraphicsPath path = new GraphicsPath();

Rectangle rect = new Rectangle(10, 10, 100, 100);

Graphics g;

g = this.CreateGraphics();

path.AddRectangle(rect);

Point[] point = { new Point(110, 10), new Point(210, 10), new Point(170, 110) };

Matrix matr1 = new Matrix(rect, point);

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.

**Пример.** 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:

**Visual Basic .Net:**

Dim path As New GraphicsPath

Dim rect As Rectangle = New Rectangle(10, 10, 100, 100)

Dim g As Graphics

g = Me.CreateGraphics

path.AddRectangle(rect)

Dim matr1 As New Matrix

matr1.Scale(0.5, 0.5)

path.Transform(matr1)

g.DrawPath(New Pen(Color.Brown, 3), path)

**Visual C#:**

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.Scale(0.5f, 0.5f);

path.Transform(matr1);

g.DrawPath(new Pen(Color.Brown, 3), path);

Все методы преобразования объекта Matrix могут использоваться комбинированно. В этом случае у них можно задать дополнительный аргумент – элементы списка MatrixOrder. Значение Prepend производит указанное преобразование до начала других преобразований, значение Append – после. Это позволяет производить преобразования не одновременно, а в определённой последовательности.

Объект Graphics имеет собственные методы для аффинных преобразований: ScaleTransform для сжатия, TranslateTransform для перемещения и RotateTransform для поворота. Синтаксис этих методов аналогичен синтаксису соответствующих методов объекта Matrix, за исключением метода для поворота, где задаётся только угол поворота, т.к. поворот осуществляется только относительно левого верхнего угла объекта Graphics (например, если пользователь рисует на форме, то поворот осуществляется относительно левого верхнего угла формы). Преимуществом данных методов является то, что с их помощью преобразовываются все нарисованные пользователем фигуры, в том числе и изображения.

Градиенты

Градиентом называется плавный переход цвета из одного оттенка в другой. В Visual Studio .Net чаще всего используются линейные градиенты и градиенты на основе объекта Path.

Линейные градиенты

Данные градиенты реализованы как кисти. Для использования градиента нужно создать кисть соответствующего типа с помощью объекта LinearGradientBrush. При создании данного объекта задаются четыре аргумента: прямоугольник, заполненный градиентом, стартовый цвет с левой стороны прямоугольника, конечный цвет с правой стороны прямоугольника и режим перехода цвета, который имеет следующие значения:

BackwardDiagonal – градиент заполняет прямоугольник по диагонали, от правого верхнего угла к левому нижнему;

ForwardDiagonal – градиент заполняет прямоугольник по диагонали от левого верхнего угла к правому нижнему;

Horizontal – градиент заполняет прямоугольник слева направо;

Vertical – градиент заполняет прямоугольник сверху вниз.

После этого создаются необходимые замкнутые графические фигуры, которые окрашиваются созданной кистью.

**Пример.** Создать градиентную кисть размером 10 х 10 пикселей с горизонтальной раскраской, переходящей от цвета морской волны к шоколадному цвету и закрасить ею прямоугольник размером 200 х 100 пикселей с координатами левого верхнего угла (10,10):

**Visual Basic .Net:**

Dim lb As LinearGradientBrush

Dim r1 As New Rectangle(10, 10, 10, 10)

lb = New LinearGradientBrush(r1, Color.Aqua, Color.Chocolate, LinearGradientMode.Horizontal)

Dim g As Graphics

g = Me.CreateGraphics

g.FillRectangle(lb, 10, 10, 200, 100)

**Visual C#:**

LinearGradientBrush lb;

Rectangle r1 = new Rectangle(10, 10, 10, 10);

lb = new LinearGradientBrush(r1, Color.Aqua, Color.Chocolate, LinearGradientMode.Horizontal);

Graphics g;

g = this.CreateGraphics();

g.FillRectangle(lb, 10, 10, 200, 200);

Координаты начала прямоугольника для создания градиента значения не имеют, но без них нельзя создать объект Rectangle. Имеют значение только его размеры, т.к. на их основе рассчитывается градиент. Если реальная фигура будет меньше этого прямоугольника, то она будет заполнена частью градиента. Если же фигура будет больше, то градиент повторится столько раз, сколько нужно для её заполнения.

Градиенты можно использовать при печати для создания эффектных надписей, для этого их нужно применить к объекту DrawString.

Градиент на основе объекта GraphicsPath

Данный градиент создается на основе объекта GraphicsPath. Цвета данного градиента начинаются в центре и расходятся по сторонам. Для использования градиента нужно создать кисть соответствующего типа с помощью объекта PathGradientBrush. Данная кисть создаётся на основе объекта GraphicsPath, в который, в принципе, можно помещать любые геометрические фигуры, но на практике рекомендуется подходить к этому разумно. Объект GraphicsPath для создания кисти должен ограничиваться замкнутой ломанной линией.

Если кисть создана на основе прямоугольника, то ею можно окрашивать сам этот прямоугольник и вписанный в него эллипс. Если кисть создана на основе многоугольника, то ею можно окрашивать сам этот многоугольник и созданный по этим же точкам замкнутый сплайн.

Цвета градиента задаются центральным цветом (определяется свойством CenterColor) и массивом цветов, расходящимся по углам фигуры (определяется свойством SurroundColors). Число элементов в массиве соответствует числу углов объекта GraphicsPath.

**Пример.** Создать градиентную кисть на основе квадрата размером 100 х 100 пикселей, с координатами левого верхнего угла (10,10). Прямоугольник помещается в объект GraphicsPath с именем p1. Градиент состоит из центрального бежевого цвета и четырёх плавно расходящихся к углам цветов (морской волны, кораллового, красного и лимонно-жёлтого). С помощью этого градиента закрасить круг, вписанный в квадрат:

**Visual Basic .Net:**

Dim p1 As New GraphicsPath

Dim r1 As Rectangle

r1 = New Rectangle(10, 10, 100, 100)

p1.AddRectangle(r1)

Dim pg AsNew PathGradientBrush(p1)

pg.CenterColor = Color.Beige

Dim scolors() As Color = {Color.Aqua, Color.Coral, Color.Red, Color.LemonChiffon}

pg.SurroundColors = scolors

Dim g As Graphics

g = Me.CreateGraphics

g.FillEllipse(pg, r1)

**Visual C#:**

GraphicsPath p1=new GraphicsPath();

Rectangle r1;

r1 = new Rectangle(10, 10, 100, 100);

p1.AddRectangle(r1);

PathGradientBrush pg = new PathGradientBrush(p1);

pg.CenterColor = Color.Beige;

Color[] scolors = { Color.Aqua, Color.Coral, Color.Red, Color.LemonChiffon };

pg.SurroundColors = scolors;

Graphics g;

g = this.CreateGraphics();

g.FillEllipse(pg, r1);

Особенностью данного градиента является то, что практически для каждой фигуры необходимо создавать свой градиент (например, только что созданным градиентом можно окрасить только круг и сам квадрат), но создаваемая окраска получается очень красивой.

Изменение единиц измерения

По умолчанию все размеры при программировании графики измеряются в пикселях. Однако размер самого пикселя на различных мониторах может быть различным, поэтому и размеры изображения могут быть для одной и той же программы разными.

Можно устанавливать свои единицы измерения, присвоив соответствующее значение свойству PageUnit объекта Graphics, например:

**Visual Basic .Net:**

Dim g As Graphics

g = Me.CreateGraphics

g.PageUnit = GraphicsUnit.Inch

**Visual C#:**

Graphics g;

g = this.CreateGraphics();

g.PageUnit = GraphicsUnit.Inch;

В этом случае размеры всех фигур будут одинаковыми на всех дисплеях всех машин, на которых запускается программа (в данном случае они будут измеряться в дюймах). Однако и толщина линий также будет измеряться в дюймах, т.е., например, инструкция

**Visual Basic .Net:**

Dim redpen As Pen

redpen = New Pen(Color.Red, 3)

**Visual C#:**

Pen redpen;

redpen = new Pen(Color.Red, 3);

создаёт перо толщиной в 3 дюйма. Если толщину линий необходимо выражать в пикселях, то это значение необходимо разделить на количество пикселей на дюйм по горизонтали и вертикали. Это количество вычисляется с помощью свойств DpiX и DpiY объекта Graphics. Данные свойства доступны только для чтения. Например, для того, чтобы создать перо, рисующее линию толщиной в три пикселя, когда единицей измерения является дюйм, необходимо выполнить оператор:

**Visual Basic .Net:**

Dim pen1 = New Pen(Color.Red, 3 \* (1 / g.DpiX))

**Visual C#:**

Pen pen1 = new Pen(Color.Red, 3 \* (1 / g.DpiX));