第 7 章　C#图形图像编程基础

本章主要介绍使用 C#进行图形图像编程基础，其中包括 GDI+绘图基础、C#图像处理基础以及简单的图像处理技术。

7.1　GDI+绘图基础

编写图形程序时需要使用 GDI（Graphics Device Interface，图形设备接口），从程序设计的角度看，GDI 包括两部分：一部分是 GDI 对象，另一部分是 GDI 函数。GDI 对象定义了 GDI 函数使用的工具和环境变量，而 GDI 函数使用 GDI 对象绘制各种图形，在 C#中，进行图形程序编写时用到的是 GDI+（Graphice Device Interface Plus 图形设备接口）版本，GDI+是 GDI 的进一步扩展，它使我们编程更加方便。

7.1.1　GDI+概述

GDI+是微软在 Windows 2000 以后操作系统中提供的新的图形设备接口，其通过一套部署为托管代码的类来展现，这套类被称为 GDI+的“托管类接口”，GDI+主要提供了以下三类服务:

(1) 二维矢量图形：GDI+提供了存储图形基元自身信息的类(或结构体)、存储图形基元绘制方式信息的类以及实际进行绘制的类。

(2) 图像处理：大多数图片都难以划定为直线和曲线的集合，无法使用二维矢量图形方式进行处理。因此，GDI+为我们提供了 Bitmap、Image 等类,它们可用于显示、操作和保存 BMP、JPG、GIF 等图像格式。

(3) 文字显示：GDI+支持使用各种字体、字号和样式来显示文本。

我们要进行图形编程，就必须先讲解 Graphics 类，同时我们还必须掌握 Pen、Brush 和 Rectangle 这几种类。

GDI+比 GDI 优越主要表现在两个方面：第（一）GDI+通过提供新功能（例如：渐变画笔和 alpha 混合）扩展了 GDI 的功能；第（二）修订了编程模型，使图形编程更加简易灵活。

7.1.2　Graphics 类

Graphics 类封装一个 GDI+绘图图面，提供将对象绘制到显示设备的方法，Graphics 与特定的设备上下文关联。画图方法都被包括在 Graphics 类中，在画任何对象（例如：

Circle,Rectangle）时，我们首先要创建一个 Graphics 类实例，这个实例相当于建立了一块画布，有了画布才可以用各种画图方法进行绘图。

绘图程序的设计过程一般分为两个步骤:(一)创建 Graphics 对象;(二)使用 Graphics 对象的方法绘图、显示文本或处理图像。

通常我们使用下述三种方法来创建一个 Graphics 对象。

方法一、利用控件或窗体的 Paint 事件中的 PainEventArgs

在窗体或控件的 Paint 事件中接收对图形对象的引用，作为 PaintEventArgs（PaintEventArgs 指定绘制控件所用的 Graphics）的一部分，在为控件创建绘制代码时，通常会使用此方法来获取对图形对象的引用。

例如：

```
//窗体的 Paint 事件的响应方法
private void form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
}
```

也可以直接重载控件或窗体的 OnPaint 方法，具体代码如下所示：

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;
}
```

Paint 事件在重绘控件时发生。

方法二、调用某控件或窗体的 CreateGraphics 方法

调用某控件或窗体的 CreateGraphics 方法以获取对 Graphics 对象的引用，该对象表示该控件或窗体的绘图图面。如果想在已存在的窗体或控件上绘图，通常会使用此方法。

例如：

```
Graphics g = this.CreateGraphics();
```

方法三、调用 Graphics 类的 FromImage 静态方法

由从 Image 继承的任何对象创建 Graphics 对象。在需要更改已存在的图像时，通常会使用此方法。

例如：

```
//名为“g1.jpg”的图片位于当前路径下
Image img = Image.FromFile("g1.jpg");//建立 Image 对象
Graphics g = Graphics.FromImage(img);//创建 Graphics 对象
```

1．Graphics 类的方法成员

有了一个 Graphics 的对象引用后，就可以利用该对象的成员进行各种各样图形的绘制，表 7.1 列出了 Graphics 类的常用方法成员。

表 7.1 Graphics 类常用方法

名称	说明

DrawArc	画弧。
DrawBezier	画立体的贝尔塞曲线。
DrawBeziers	画连续立体的贝尔塞曲线。
DrawClosedCurve	画闭合曲线。
DrawCurve	画曲线。
DrawEllipse	画椭圆。
DrawImage	画图像。
DrawLine	画线。
DrawPath	通过路径画线和曲线。
DrawPie	画饼形。
DrawPolygon	画多边形。
DrawRectangle	画矩形。
DrawString	绘制文字。
FillEllipse	填充椭圆。
FillPath	填充路径。
FillPie	填充饼图。
FillPolygon	填充多边形。
FillRectangle	填充矩形。
FillRectangles	填充矩形组。
FillRegion	填充区域。

在.NET 中，GDI+的所有绘图功能都包括在 System、System.Drawing、System.Drawing.Imaging、System.Drawing.Darwing2D 和 System.Drawing.Text 等命名空间中，因此在开始用 GDI+类之前，需要先引用相应的命名空间。

2．引用命名空间

在 C#应用程序中使用 using 命令已用给定的命名空间或类，下面是一个 C#应用程序引用命名空间的例子：

```
using System;
using System.Collections.Generic;
using System.Data;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;
```

7.1.3 常用画图对象

在创建了 Graphics 对象后，就可以用它开始绘图了，可以画线、填充图形、显示

文本等等，其中主要用到的对象还有：

- Pen：用来用 patterns、colors 或者 bitmaps 进行填充。
- Color：用来画线和多边形，包括矩形、圆和饼形。
- Font：用来给文字设置字体格式。
- Brush：用来描述颜色。
- Rectangle：矩形结构通常用来在窗体上画矩形。
- Point：描述一对有序的 x,y 两个坐标值。

1．Pen 类

Pen 用来绘制指定宽度和样式的直线。使用 DashStyle 属性绘制几种虚线，可以使用各种填充样式（包括纯色和纹理）来填充 Pen 绘制的直线，填充模式取决于画笔或用作填充对象的纹理。

使用画笔时，需要先实例化一个画笔对象，主要有以下几种方法。

用指定的颜色实例化一只画笔的方法如下：

```
public Pen(Color);
```

用指定的画刷实例化一只画笔的方法如下：

```
public Pen(Brush);
```

用指定的画刷和宽度实例化一只画笔的方法如下：

```
public Pen(Brush, float);
```

用指定的颜色和宽度实例化一只画笔的方法如下：

```
public Pen(Color, float);
```

实例化画笔的语句格式如下：

```
Pen pn=new Pen(Color.Blue);
```

或者 `Pen pn=new Pen(Color.Blue,100);`

Pen 常用的属性有以下几个，如表 7.2 所示：

表 7.2 Pen 常用属性

名称	说明
Alignment	获得或者设置画笔的对齐方式。
Brush	获得或者设置画笔的属性。
Color	获得或者设置画笔的颜色。
Width	获得或者设置画笔的宽度。

2．Color 结构

在自然界中，颜色大都由透明度（A）和三基色（R,G,B）所组成。在 GDI+中，通过 Color 结构封装对颜色的定义，Color 结构中，除了提供（A,R,G,B）以外，还提供许多系统定义的颜色，如 Pink（粉颜色），另外，还提供许多静态成员，用于对颜色进行操作。Color 结构的基本属性如表 7.3 所示。

表 7.3　颜色的基本属性

名称	说明

A	获取此 Color 结构的 alpha 分量值，取值（0～255）。
B	获取此 Color 结构的蓝色分量值，取值（0～255）。
G	获取此 Color 结构的绿色分量值，取值（0～255）。
R	获取此 Color 结构的红色分量值，取值（0～255）。
Name	获取此 Color 结构的名称，这将返回用户定义的颜色的名称或已知颜色的名称（如果该颜色是从某个名称创建的），对于自定义的颜色，将返回 RGB 值。

Color 结构的基本(静态)方法如表 7.4 所示

表 7.4　颜色的基本方法

名称	说明
FromArgb	从四个 8 位 ARGB 分量(alpha、红色、绿色和蓝色)值创建 Color 结构。
FromKnowColor	从指定的预定义颜色创建一个 Color 结构。
FromName	从预定义颜色的指定名称创建一个 Color 结构。

Color 结构变量可以通过已有颜色构造，也可以通过 RGB 建立，例如：

```
Color clr1 = Color.FromArgb(122,25,255);
Color clr2 = Color.FromKnowColor(KnowColor.Brown);//KnownColor 为枚举类型
Color clr3 = Color.FromName("SlateBlue");
```

在图像处理中一般需要获取或设置像素的颜色值，获取一幅图像的某个像素颜色值的具体步骤如下：

（1）定义 Bitmap

```
Bitmap myBitmap = new Bitmap("c:\\MyImages\\TestImage.bmp");
```

（2）定义一个颜色变量把在指定位置所取得的像素值存入颜色变量中

```
Color c = new Color();
c = myBitmap.GetPixel(10,10);//获取此 Bitmap 中指定像素的颜色。
```

（3）将颜色值分解出单色分量值

```
int r,g,b;
r= c.R;
g=c.G;
b=c.B;
```

3．Font 类

Font 类定义特定文本格式，包括字体、字号和字形属性。Font 类的常用构造函数是 public Font(string 字体名,float 字号，FontStyle 字形)，其中字号和字体为可选项和 public Font(string 字体名,float 字号)，其中“字体名”为 Font 的 FontFamily 的字符串表示形式。下面是定义一个 Font 对象的例子代码：

```
FontFamily fontFamily = new FontFamily("Arial");
Font font = new Font(fontFamily,16,FontStyle.Regular,GraphicsUnit.Pixel);
```

字体常用属性如表 7.5 所示。

表 7.5 字体的常用属性

名称	说明
Bold	是否为粗体。
FontFamily	字体成员。
Height	字体高。
Italic	是否为斜体。
Name	字体名称。
Size	字体尺寸。
SizeInPoints	获取此 Font 对象的字号，以磅为单位。
Strikeout	是否有删除线。
Style	字体类型。
Underline	是否有下划线。
Unit	字体尺寸单位。

4．Brush 类

Brush 类是一个抽象的基类，因此它不能被实例化，我们总是用它的派生类进行实例化一个画刷对象，当我们对图形内部进行填充操作时就会用到画刷，关于画刷在 7.1.5 中有详细讲解。

5．Rectangle 结构

存储一组整数，共四个，表示一个矩形的位置和大小。矩形结构通常用来在窗体上画矩形，除了利用它的构造函数构造矩形对象外，还可以使用 Rectangle 结构的属性成员，其属性成员如表 7.6 所示。

表 7.6 Rectangle 结构属性

名称	说明
Bottom	底端坐标
Height	矩形高
IsEmpty	测试矩形宽和高是否为 0
Left	矩形左边坐标
Location	矩形的位置
Right	矩形右边坐标
Size	矩形尺寸.
Top	矩形顶端坐标
Width	矩形宽
X	矩形左上角顶点 X 坐标
Y	矩形左上角顶点 Y 坐标

Retangle 结构的构造函数有以下两个：

//用指定的位置和大小初始化 Rectangle 类的新实例。

public Retangle(Point,Size);//Size 结构存储一个有序整数对，通常为矩形的宽度和高度。

和

```
public Rectangle(int,int,int,int);
```

6．Point 结构

用指定坐标初始化 Point 类的新实例。这个结构很像 C++中的 Point 结构，它描述了一对有序的 x,y 两个坐标值，其构造函数为：public Point(int x,int y)；其中 x 为该点的水平位置；y 为该点的水垂直位置。下面是构造 Point 对象的例子代码：

```
Point pt1=new Point(30,30);
Point pt2=new Point(110,100);
```

7.1.4 基本图形绘制举例

1．画一个矩形

【例 7.1】建立一个项目，在窗体上画一个矩形，通过直接在 Form1 类中重载 OnPaint 函数的方法来实现。

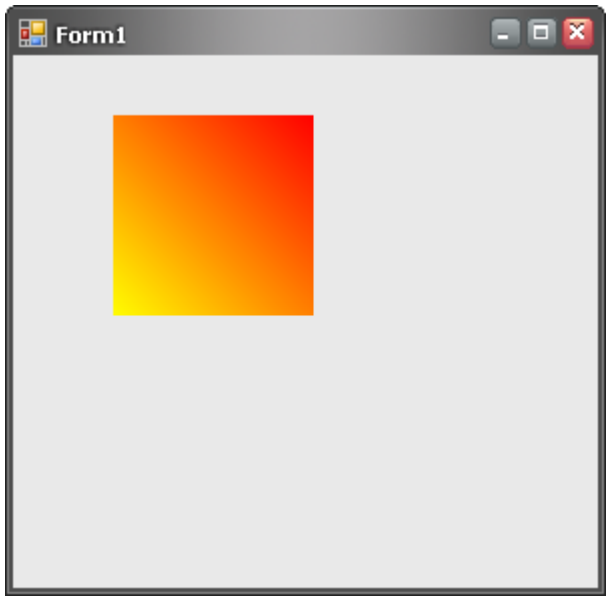

图 7.1 画一个矩形

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics ;
    Rectangle rect = new Rectangle(50, 30, 100, 100);
    LinearGradientBrush lBrush = new LinearGradientBrush(rect,
    Color.Red,Color.Yellow,LinearGradientMode.BackwardDiagonal);
    g.FillRectangle(lBrush, rect);
```

```
}
```

运行结果如图 7.1 所示。

2. 画一个弧

【例 7.2】画一个弧形。

弧形函数格式如下：

```
public void DrawArc(Pen pen,Rectangle rect,Float startArgle,Float sweepAngle);
```

直接在 Form1 类中重载 OnPaint 函数

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics ;
    Pen pn = new Pen( Color.Blue);
    Rectangle rect = new Rectangle(50, 50, 200,100);
    g.DrawArc(pn,rect,12,84);
}
```

运行结果如图 7.2 所示。

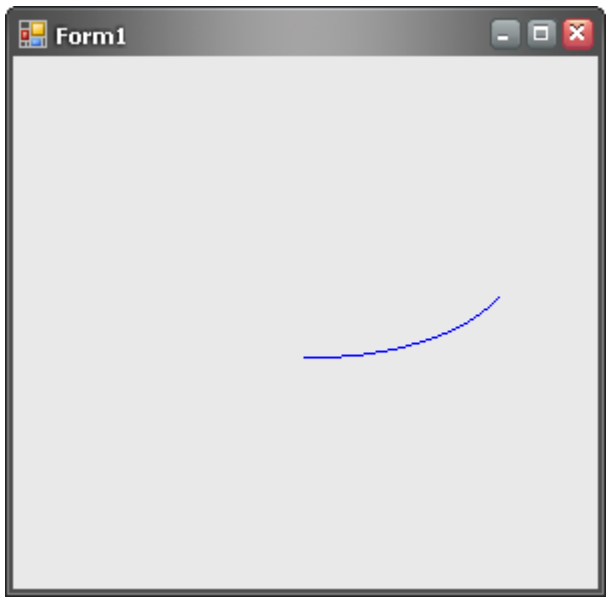

图 7.2 画一个弧

3. 画线

【例 7.3】画一条线。

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics ;
    Pen pn = new Pen(Color.Blue);
```

```
    Point pt1 = new Point(30,30);
    Point pt2 = new Point(110,100);
    g.DrawLine(pn,pt1,pt2);
}
```

运行结果如图 7.3 所示。

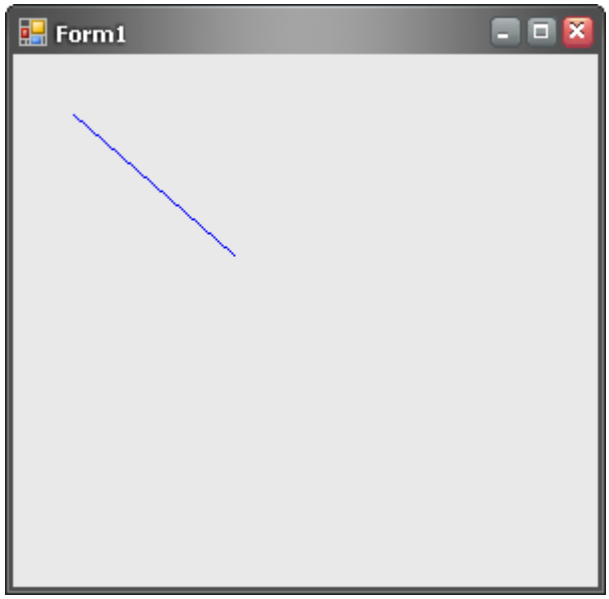

图 7.3 画一条线

4．画椭圆

【例 7. 4】画一个椭圆。

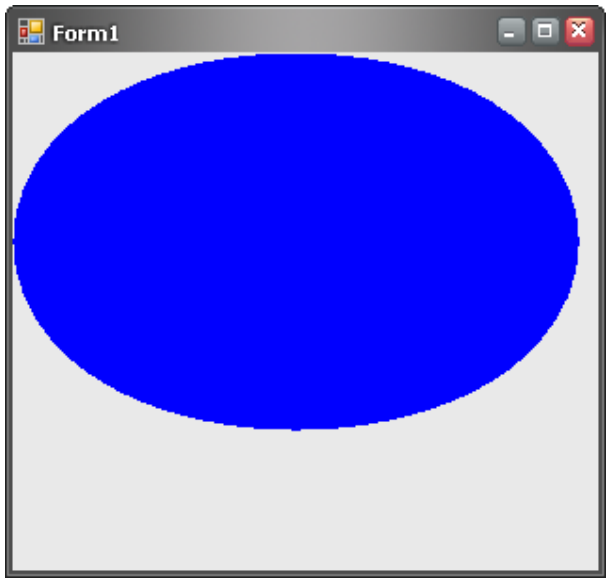

图 7.4 画一个椭圆

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics ;
    Pen pn = new Pen( Color.Blue, 100 );
    Rectangle rect = new Rectangle(50, 50, 200, 100);
    g.DrawEllipse( pn, rect );
}
```

运行结果如图 7.4 所示。

5. 输出文本

【例 7.5】输出文本。

```
protected override void OnPaint(PaintEventArgs e)
{
     Font fnt = new Font("Verdana", 16);
     Graphics g = e.Graphics;
     g.DrawString("GDI+ World", fnt, new SolidBrush(Color.Red), 14,10);
}
```

运行结果如图 7.5 所示。

图 7.5 输出文本

6. 填充路径

【例 7.6】填充路径。

```
protected override void OnPaint(PaintEventArgs e)
{
  Graphics g = e.Graphics;
```

```
    g.FillRectangle(new SolidBrush(Color.White), ClientRectangle);
    GraphicsPath path = new GraphicsPath(new Point[] {
    new Point(40, 140), new Point(275, 200),
    new Point(105, 225), new Point(190, 300),
    new Point(50, 350), new Point(20, 180), },
    new byte[] {
    (byte)PathPointType.Start,
    (byte)PathPointType.Bezier,
    (byte)PathPointType.Bezier,
    (byte)PathPointType.Bezier,
    (byte)PathPointType.Line,
    (byte)PathPointType.Line,
    });
    PathGradientBrush pgb = new PathGradientBrush(path);
    pgb.SurroundColors = new Color[]
      {
      Color.Green,Color.Yellow,Color.Red, Color.Blue,
      Color.Orange, Color.White,
      };
      g.FillPath(pgb, path);
}
```

运行结果如图 7.6 所示。

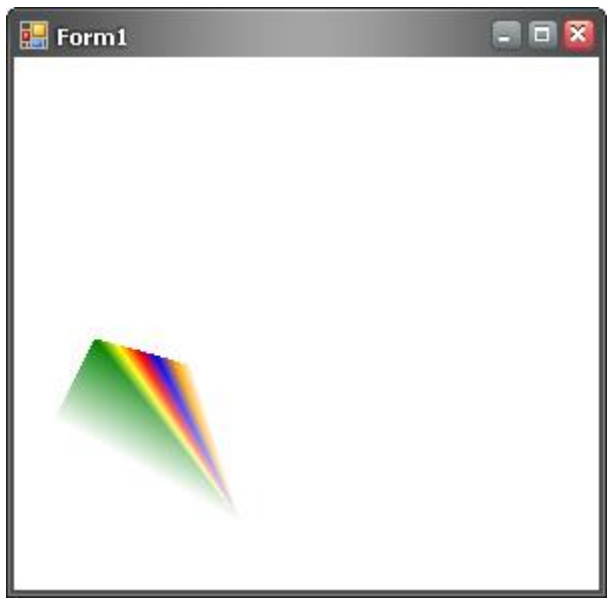

图 7.6 填充路径

注意：GraphicsPath 类位于命名空间 System.Drawing.Drawing2D 中，表示一系列相互连接的直线和曲线。

7.1.5 画刷和画刷类型

Brush 类型是一个抽象类，所以它不能被实例化，也就是不能直接应用，但是我们可以利用它的派生类，如：HatchBrush、SolidBrush 和 TextureBrush 等。画刷类型一般在 System.Drawing 命名空间中，如果应用 HatchBrush 和 GradientBrush 画刷，需要在程序中引入 System.Drawing.Drawing2D 命名空间。

1. SolidBrush（单色画刷）

它是一种一般的画刷，通常只用一种颜色去填充 GDI+图形，例如：

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;
    SolidBrush sdBrush1 = new SolidBrush(Color.Red);
    SolidBrush sdBrush2 = new SolidBrush(Color.Green);
    SolidBrush sdBrush3 = new SolidBrush(Color.Blue);
    g.FillEllipse(sdBrush2, 20, 40, 60, 70);
    Rectangle rect = new Rectangle(0, 0, 200, 100);
    g.FillPie(sdBrush3, 0, 0, 200, 40, 0.0f, 30.0f );
    PointF point1 = new PointF(50.0f, 250.0f);
    PointF point2 = new PointF(100.0f, 25.0f);
    PointF point3 = new PointF(150.0f, 40.0f);
    PointF point4 = new PointF(250.0f, 50.0f);
    PointF point5 = new PointF(300.0f, 100.0f);
    PointF[] curvePoints = {point1, point2, point3, point4,point5 };
    g.FillPolygon(sdBrush1, curvePoints);
}
```

运行结果如图 7.7 所示。

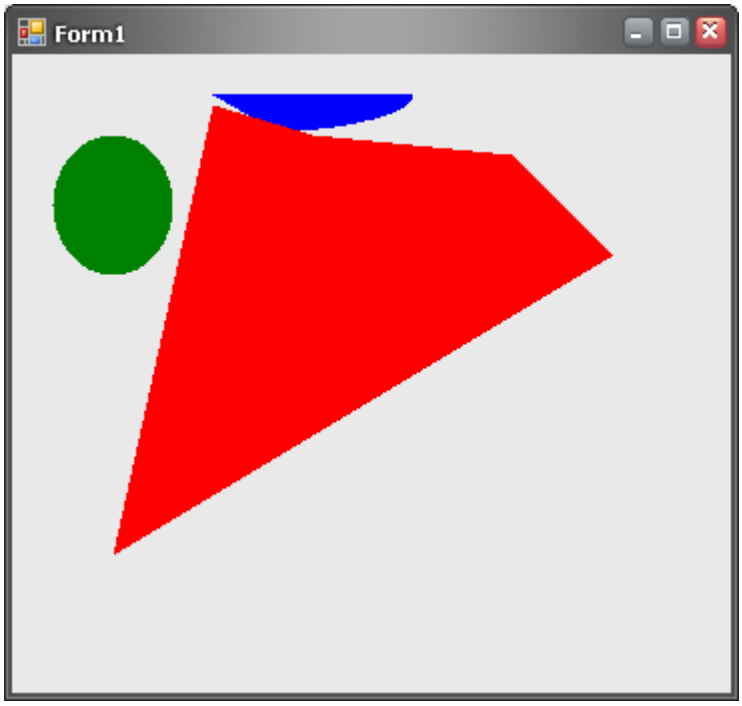

图 7.7 SolidBrush 应用

2．HatchBrush（阴影画刷）

HatchBrush 类位于 System.Drawing.Drawing2D 命名空间中。阴影画刷有两种颜色：前景色和背景色，以及 6 种阴影。前景色定义线条的颜色，背景色定各线条之间间隙的颜色。HatchBrush 类有两个构造函数：

- `public HatchBrush(HatchStyle,Color forecolor);`
- `public HatchBrush(HatchStyle,Color forecolor,Color backcolor);`

HatchStyle 枚举值指定可用于 HatchBrush 对象的不同图案。

HatchStyle 的主要成员如表 7.7 所示。

表 7.7 HatchStyle 主要成员

名称	说明
BackwardDiagonal	从右上到左下的对角线的线条图案。
Cross	指定交叉的水平线和垂直线。
DarkDownwardDiagonal	指定从顶点到底点向右倾斜的对角线，其两边夹角比 ForwardDiagonal 小 50%，宽度是其两倍。此阴影图案不是锯齿消除的。
DarkHorizontal	指定水平线的两边夹角比 Horizontal 小 50%并且宽度是 Horizontal 的两倍。
DarkUpwardDiagonal	指定从顶点到底点向左倾斜的对角线，其两边夹角比 BackwardDiagonal 小 50%，宽度是其两倍，但这

	些直线不是锯齿消除的。
DarkVertical	指定垂直线的两边夹角比 Vertical 小 50%并且宽度是其两倍。
DashedDownwardDiagonal	指定虚线对角线，这些对角线从顶点到底点向右倾斜。
DashedHorizontal	指定虚线水平线。
DashedUpwardDiagonal	指定虚线对角线，这些对角线从顶点到底点向左倾斜。
DashedVertical	指定虚线垂直线。
DiagonalBrick	指定具有分层砖块外观的阴影，它从顶点到底点向左倾斜。
DiagonalCross	交叉对角线的图案。
Divot	指定具有草皮层外观的阴影。
ForwardDiagonal	从左上到右下的对角线的线条图案。
Horizontal	水平线的图案。
HorizontalBrick	指定具有水平分层砖块外观的阴影。
LargeGrid	指定阴影样式 Cross。
LightHorizontal	指定水平线，其两边夹角比 Horizontal 小 50%。
LightVertical	指定垂直线的两边夹角比 Vertical 小 50%。
Max	指定阴影样式 SolidDiamond。
Min	指定阴影样式 Horizontal。
NarrowHorizontal	指定水平线的两边夹角比阴影样式 Horizontal 小 75%（或者比 LightHorizontal 小 25%）。
NarrowVertical	指定垂直线的两边夹角比阴影样式 Vertical 小 75%（或者比 LightVertica 小 25%）。
OutlinedDiamond	指定互相交叉的正向对角线和反向对角线，但这些对角线不是锯齿消除的。
Percent05	指定 5%阴影。前景色与背景色的比例为 5:100。
Percent90	指定 90%阴影。前景色与背景色的比例为 90:100。
Plaid	指定具有格子花呢材料外观的阴影。
Shingle	指定带有对角分层鹅卵石外观的阴影，它从顶点到底点向右倾斜。
SmallCheckerBoard	指定带有棋盘外观的阴影。
SmallConfetti	指定带有五彩纸屑外观的阴影。
SolidDiamond	指定具有对角放置的棋盘外观的阴影。
Sphere	指定具有球体彼此相邻放置的外观的阴影。
Trellis	指定具有格架外观的阴影。
Vertical	垂直线的图案。

Wave	指定由代字号“~”构成的水平线。
Weave	指定具有织物外观的阴影。

下面代码显示了 HatchBrush 画刷的使用。

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;
    HatchBrush hBrush1 = new HatchBrush(HatchStyle.DiagonalCross,
    Color.Chocolate, Color.Red);
    HatchBrush hBrush2 = new HatchBrush(HatchStyle.DashedHorizontal,
Color.Green, Color.Black);
    HatchBrush hBrush3 = new HatchBrush(HatchStyle.Weave,
Color.BlueViolet, Color.Blue);
    g.FillEllipse(hBrush1, 20, 80, 60, 20);
    Rectangle rect = new Rectangle(0, 0, 200, 100);
    g.FillPie(hBrush3, 0, 0, 200, 40, 0.0f, 30.0f );
    PointF point1 = new PointF(50.0f, 250.0f);
    PointF point2 = new PointF(100.0f, 25.0f);
    PointF point3 = new PointF(150.0f, 40.0f);
    PointF point4 = new PointF(250.0f, 50.0f);
    PointF point5 = new PointF(300.0f, 100.0f);
    PointF[] curvePoints = {point1, point2, point3, point4, point5 };
    g.FillPolygon(hBrush2, curvePoints);
}
```

运行结果如图 7.8 所示。

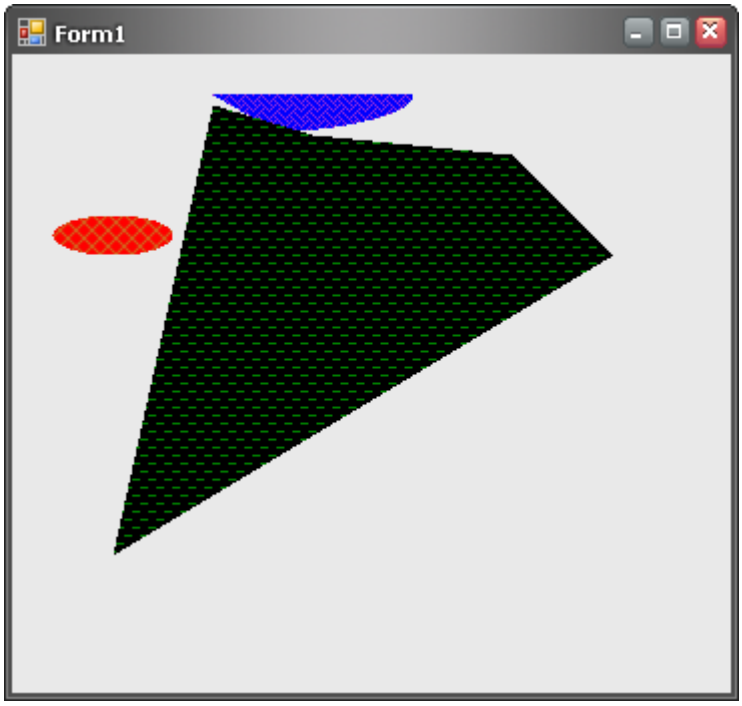

图 7.8 HatchBrush 应用

3．TextureBrush（纹理画刷）

纹理画刷拥有图案，并且通常使用它来填充封闭的图形。为了对它初始化，可以使用一个已经存在的别人设计好了的图案，或使用常用的设计程序设计的自己的图案，同时应该使图案存储为常用图形文件格式，如 BMP 格式文件。这里有一个设计好的位图，被存储为 Papers.bmp 文件。

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    //根据文件名创建原始大小的 bitmap 对象
    Bitmap bitmap = new Bitmap("D:\\mm.jpg");
    //将其缩放到当前窗体大小
    bitmap = new Bitmap(bitmap, this.ClientRectangle.Size);
    TextureBrush myBrush = new TextureBrush(bitmap);
    g.FillEllipse(myBrush, this.ClientRectangle);
}
```

运行结果如图 7.9 所示。

图 7.9 TextTureBursh 应用

4. LinearGradientBrush 和 PathGradientBrush（渐变画刷）

渐变画刷类似与实心画刷，因为它也是基于颜色的，与实心画刷不同的是：渐变画刷使用两种颜色；它的主要特点是：在使用过程中，一种颜色在一端，而另外一种颜色在另一端，在中间，两种颜色融合产生过渡或衰减的效果。

渐变画刷有两种：线性画刷和路径画刷（LinearGradientBrush 和 PathGradientBrush）。

其中 LinearGradientBrush 可以显示线性渐变效果，而 PathGradientBrush 是路径渐变的可以显示比较具有弹性的渐变效果。

（1）LinearGradientBrush 类

LinearGradientBrush 类构造函数如下：

public LinearGradientBrush(Point point1,Point point2,Color color1,Color color2)

参数说明：

point1：表示线性渐变起始点的 Point 结构。

point2：表示线性渐变终结点的 Point 结构。

color1：表示线性渐变起始色的 Color 结构。

color2：表示线性渐变结束色的 Color 结构。

代码如下：

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
```

```
    LinearGradientBrush myBrush = new
    LinearGradientBrush(this.ClientRectangle, Color.White, Color.Blue,
    LinearGradientMode.Vertical);
    g.FillRectangle(myBrush, this.ClientRectangle);
}
```

运行结果如图 7.10 所示。

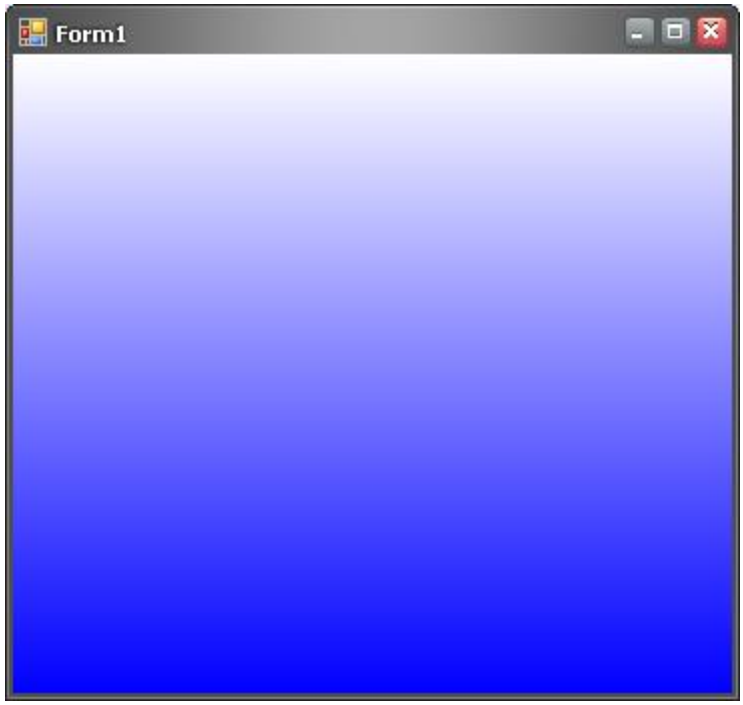

图 7.10 LinearGradientBrush 的应用

（2）PathGradientBrush 类

PathGradientBrush 类的构造函数如下：

```
public PathGradientBrush (GraphicsPath path);
```

参数说明：

path：GraphicsPath，定义此 PathGradientBrush 填充的区域。

例子代码如下：

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Point centerPoint = new Point(150, 100);
    int R = 60;
    GraphicsPath path = new GraphicsPath();
    path.AddEllipse(centerPoint.X-R,centerPoint.Y-R,2*R,2*R);
    PathGradientBrush brush = new PathGradientBrush(path);
```

```
        //指定路径中心点
        brush.CenterPoint = centerPoint;
        //指定路径中心的颜色
        brush.CenterColor = Color.Red;
        //Color 类型的数组指定与路径上每个顶点的颜色
        brush.SurroundColors = new Color[] { Color.Plum };
        g.FillEllipse(brush,centerPoint.X-R,centerPoint.Y-R,2*R,2* R);
        centerPoint = new Point(350, 100);
        R = 20;
        path = new GraphicsPath();
        path.AddEllipse(centerPoint.X-R,centerPoint.Y-R,2*R,2*R);
        path.AddEllipse(centerPoint.X-2*R,centerPoint.Y-2*R,4*R,4* R);
        path.AddEllipse(centerPoint.X-3*R,centerPoint.Y-3*R,6*R,6* R);
        brush = new PathGradientBrush(path);
        brush.CenterPoint = centerPoint;
        brush.CenterColor = Color.Red;
        brush.SurroundColors = new Color[] { Color.Black, Color.Blue,
Color.Green };
        g.FillPath(brush, path);
}
```

运行结果如图 7.11 所示。

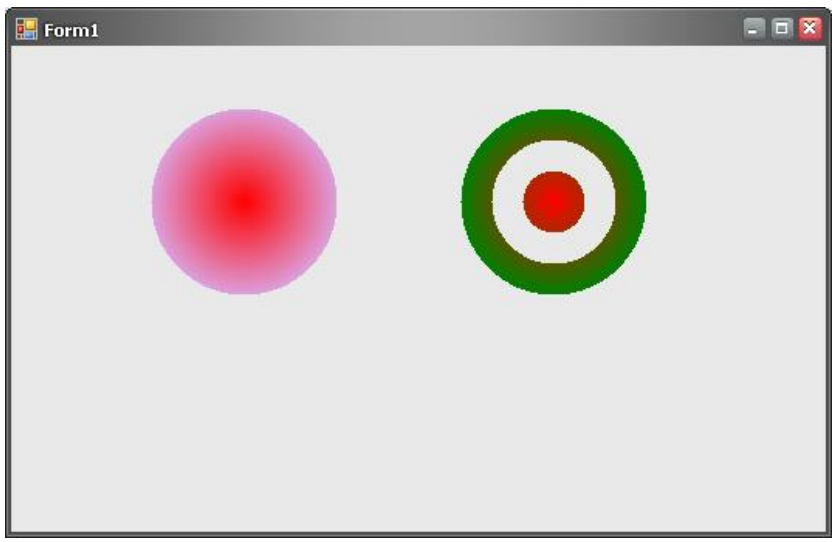

图 7.11 PathGradientBrush 应用

7.2 C#图像处理基础

本节主要介绍C#图像处理基础知识以及对图像的基本处理方法和技巧，主要包括图像的加载、变换和保存等操作。

7.2.1 C#图像处理概述

1. 图像文件的类型

GDI+支持的图像格式有BMP、GIF、JPEG、EXIF、PNG、TIFF、ICON、WMF、EMF等，几乎涵盖了所有的常用图像格式，使用GDI+可以显示和处理多种格式的图像文件。

2. 图像类

GDI+提供了Image、Bitmap和Metafile等类用于图像处理，为用户进行图像格式的加载、变换和保存等操作提供了方便。

（1）Image类

Image类是为Bitmap和Metafile的类提供功能的抽象基类。

（2）Metafile类

定义图形图元文件，图元文件包含描述一系列图形操作的记录，这些操作可以被记录（构造）和被回放（显示）。

（3）Bitmap类

封装GDI+位图，此位图由图形图像及其属性的像素数据组成，Bitmap是用于处理由像素数据定义的图像的对象，它属于System.Drawing命名空间，该命名空间提供了对GDI+基本图形功能的访问。Bitmap类常用方法和属性如表7.8所示。

表7.8 Bitmap常用属性和方法

名称	说明
公共属性	
Height	获取此Image对象的高度。
RawFormat	获取此Image对象的格式。
Size	获取此Image对象的宽度和高度。
Width	获取此Image对象的宽度。
公共方法	
GetPixel	获取此Bitmap中指定像素的颜色。
MakeTransparent	使默认的透明颜色对此Bitmap透明。
RotateFlip	旋转、翻转或者同事旋转和翻转Image对象。
Save	将Image对象以指定的格式保存到指定的Stream对象。
SetPixel	设置Bitmap对象中指定像素的颜色。

SetPropertyItem	将指定的属性项设置为指定的值。
SetResolution	设置此 Bitmap 的分辨率。

Bitmap 类有多种构造函数，因此可以通过多种形式建立 Bitmap 对象，例如：

从指定的现有图像建立 Bitmap 对象

```
Bitmap box1 =new Bitmap(pictureBox1.Image);
```

从指定的图像文件建立 Bitmap 对象，其中“C:\MyImages\TestImage.bmp”已存在的图像文件

```
Bitmap box2 =new Bitmap("C:\\MyImages\\TestImage.bmp");
```

从现有的 Bitmap 对象建立新的 Bitmap 对象

```
Bitmap box3 = new Bitmap(box1);
```

7.2.2 图像的输入和保存

1. 图像的输入

在窗体或图形框内输入图像有两种方式：（一）在窗体设计时使用图形框对象的 Image 属性输入；（二）在程序中通过打开文件对话框输入。

方法（一）、窗体设计时使用图形框对象的 Image 属性输入

窗体设计时使用对象的 Image 属性输入图像的操作如下：

（1)在窗体上，建立一个图形框对象(pictureBox1)，选择图形框对象属性中的 Image 属性，如图 7.12 所示。

（2）单击 Image 属性右侧的【…】，弹出一个“选择资源”窗口，在该窗口中选择“本地资源”，单击【导入(M)...】将弹出一个“打开”对话框，如图 7.13 所示。

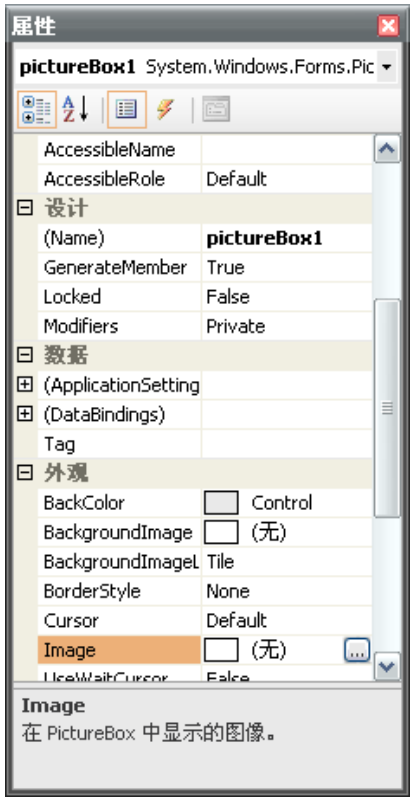

图 7.12 Image 属性

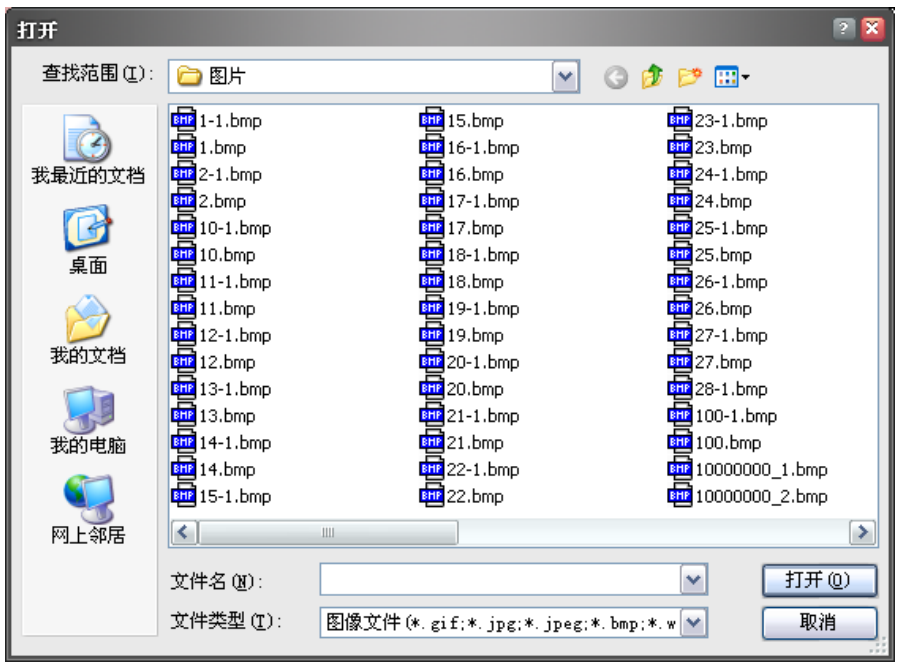

图 7.13“打开”对话框

（3）选择图像文件后，单击【打开】按钮。

方法（二）、使用“打开文件”对话框输入图像

在窗体上添加一个命令按钮（button1）和一个图形框对象（pictureBox1），双击命令按钮，在响应方法中输入如下代码：

```
private void button1_Click(object sender, EventArgs e)
{
    OpenFileDialog ofdlg = new OpenFileDialog();
    ofdlg.Filter = "BMP File(*.bmp)|*.bmp";
    if (ofdlg.ShowDialog() == DialogResult.OK)
    {
        Bitmap image = new Bitmap(ofdlg.FileName);
        pictureBox1.Image = image;
    }
}
```

执行该程序时，使用“打开文件”对话框，选择图像文件，该图像将会被打开，并显示在 pictureBox1 图像框中。

【例 7.7】图像输入。

采用方法（二）来实现图像的输入。

设计步骤如下：

（1）建立如图 7.14 所示的项目界面，在窗体上加入【打开图像】命令按钮和一个 PictureBox 控件。

（2）双击【打开图像】命令按钮，编辑按钮的单击事件响应函数，其代码同方法（二）中所写代码，在此不再重复。

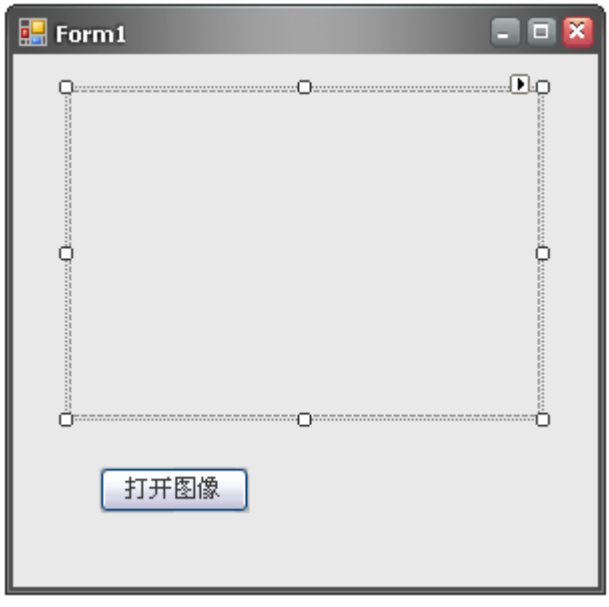

图 7.14 图像输入

（3）运行后单击【打开图像】按钮，弹出一个“打开文件”对话框，选择图象文件名，运行结果如图 7.15 所示。

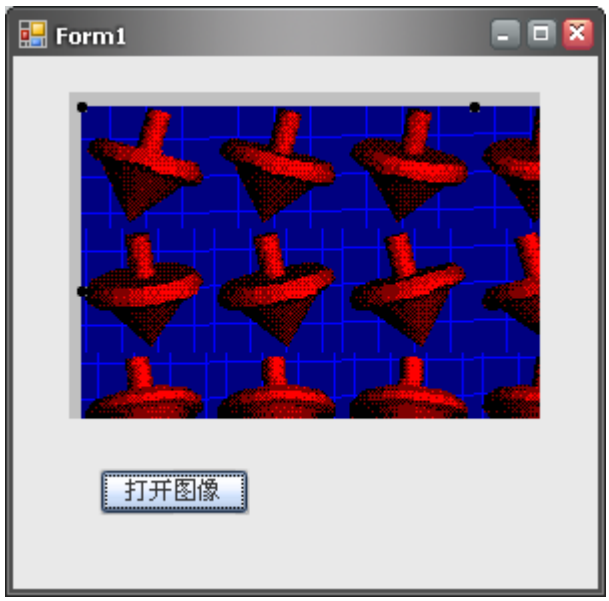

图 7.15 运行结果

2．图像的保存

保存图像的步骤如下：

（1）当使用按钮和保存对话框保存文件时，加入保存按钮和 PictureBox 控件，窗体设计如图 7.16 所示。

（2）保存命令钮的单击事件的响应函数代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
 string str;
 Bitmap box1 = new Bitmap(pictureBox1.Image);
 SaveFileDialog sfdlg = new SaveFileDialog();
 sfdlg.Filter = "bmp文件(*.BMP)|*.BMP|All File(*.*)|*.*";
 sfdlg.ShowDialog();
 str = sfdlg.FileName;
 box1.Save(str);
}
```

执行该过程时，将打开“另存为”对话框，如图 7.17 所示。

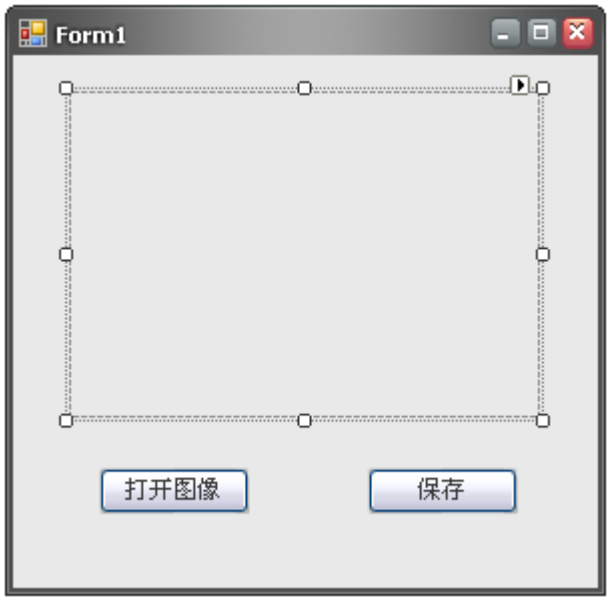

图 7.16 保存图像

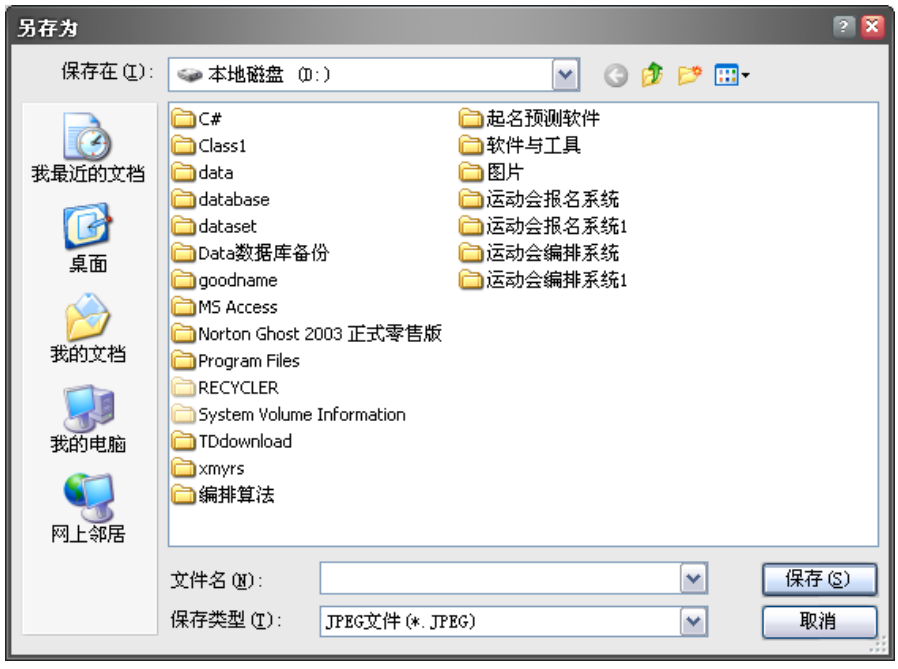

图 7.17 “另存为”对话框

选择图像文件的保存路径。

3. 图像格式的转换

使用 Bitmap 对象的 Save 方法，可以把打开的图像保存为不同的文件格式，从而实现图像格式的转换。在上述例子中添加一个命令按钮，双击该命令按钮，编辑其相应代码如下：

```
private void button3_Click(object sender, EventArgs e)
```

```
{
    string str;
    Bitmap box1 = new Bitmap(pictureBox1.Image);
    SaveFileDialog sfdlg = new SaveFileDialog();
    sfdlg.Filter = "bmp文件(*.jpeg)|*.jpeg|All File(*.*)|*.*";
    sfdlg.ShowDialog();
    str = sfdlg.FileName;
    box1.Save(str,System.Drawing.Imaging.ImageFormat.Jpeg);
}
```

Bitmap 对象的 Save 方法中的第二个参数指定了图像保存的格式。

Imaging.ImageFormat 支持的格式如表 7.9 所示：

表 7.9 Imaging.ImageFormat 支持的格式

名称	说明
Bmp	获取位图图像格式(BMP)。
Emf	获取增强型 Windows 图元文件图像格式(EMF)。
Exif	获取可交换图像文件(Exif)格式。
Gif	获取图形交换格式(GIF)图像格式。
Guid	获取表示此 ImageForma 对象的 Guid 结构。
Icon	获取 Windows 图标图像格式。
Jpeg	获取联合图像专家组(JPEG)图像格式。
MemoryBmp	获取内存位图图像格式。
Png	获取 W3C 可移植网络图形(PNG)图像格式。
Tiff	获取标签图像文件格式(TIFF)图像格式。
Wmf	获取 Windows 图元文件(WMF)图像格式。

7.2.3 图像的拷贝和粘贴

图像拷贝和粘贴是图像处理的基本操作之一，通常有两种方法来完成图像的拷贝和粘贴：一种可以使用剪贴板拷贝和粘贴图像，一种使用 AxPictureClip 控件拷贝和粘贴图像。

1．使用剪贴板拷贝和粘贴图像

剪贴板是在 Windwos 系统中单独预留出来的一块内存，它用来暂时存放在 Windwos 应用程序间要交换的数据，使用剪贴板对象可以轻松实现应用程序间的数据交换，这些数据包括图像或文本。在 C#中，剪贴板通过 Clipboard 类来实现，Clipboard 类的常用方法如表 7.10 所示。

表 7.10 Clipboard 类常用方法

名称	说明
Clear	从剪贴板中移除所有数据。
ContainsData	指示剪贴板中是否存在指定格式的数据，或可转换成此格式的数据。
ContainsImage	指示剪贴板中是否存在 Bitmap 格式或可转换成此格式的数据。
ContainsText	已重载。指示剪贴板中是否存在文本数据。
GetData	从剪贴板中检索指定格式的数据。
GetDataObject	检索当前位于系统剪贴板中的数据。
GetFileDropList	从剪贴板中检索文件名的集合。
GetImage	检索剪贴板上的图像。
GetText	已重载。从剪贴板中检索文本数据。
SetAudio	已重载。将 WaveAudio 格式的数据添加到剪贴板中。
SetData	将指定格式的数据添加到剪贴板中。
SetDataObject	已重载。将数据置于系统剪贴板中。
SetImage	将 Bitmap 格式的 Image 添加到剪贴板中。
SetText	已重载。将文本数据添加到剪贴板中。

剪贴板的使用主要有一下两个步骤：

- 将数据置于剪贴板中。
- 从剪贴板中检索数据。

下面简要介绍剪贴板的使用。

（1）将数据置于剪贴板中

可以通过 SetDataObject 方法将数据置于剪贴板中，SetDataObject 方法有以下三种形式的定义：

- Clipboard.SetDataObject(Object)：将非持久性数据置于系统剪贴板中。由.NET Compact Framework 支持。
- Clipboard.SetDataObject(Object,Boolean)：将数据置于系统剪贴板中，并指定在退出应用程序后是否将数据保留在剪贴板中。
- Clboard.SetDataObject(Object,Boolean,Int32,Int32)：尝试指定的次数，以将数据置于系统剪贴板中，且两次尝试之间具有指定的延迟，可以选择在退出应用程序后将数据保留在剪贴板中。

将字符串置于剪贴板中的语句如下所示：

```
string str = "Mahesh writing data to the Clipboard";
Clipboard.SetDataObject(str)
```

（2）从剪贴板中检索数据

可以通过 GetDataObject 方法从剪贴板中检索数据，它将返回 IdataObject，其定义如下：

```
public static IDataObject GetDataObject();
```

首先使用 IdataObject 对象的 GetDataPresent 方法检测剪贴板上存放的是什么类型的数据，然后是使用 IdataObject 对象的 GetData 方法获取剪贴板上相应的数据类型的数据。下面使用 GetDataObject 方法从剪贴板中检索出字符串数据。

例如：

```
IDataObject iData = Clipboard.GetDataObject();
if (iData.GetDataPresent(DataFormats.Text))
{
    string str =(String)iData.GetData(DataFormats.Text);
}
```

【例 7.8】使用剪贴板拷贝和粘贴图像。

（1）建立如图 7.18 所示的窗体。在窗体上天加两个图片框控件和两个命令按钮控件。利用第一个图片框的属性窗口为其输入图像。

（2）双击【复制】命令按钮，输入如下代码，将图像置于剪贴板中。

```
private void button1_Click(object sender, EventArgs e)
{
   Clipboard.SetDataObject(pictureBox1.Image);
}
```

（3）双击【粘贴】命令按钮，输入如下代码，从剪贴板中检索出图像，并显示于第二个图片框中。

```
private void button2_Click(object sender, EventArgs e)
{
   IDataObject iData = Clipboard.GetDataObject();
   if (iData.GetDataPresent(DataFormats.Bitmap))
   {
        pictureBox2.Image = (Bitmap)iData.GetData
(DataFormats.Bitmap);
    }
}
```

（4）运行程序，首先单击【复制】命令按钮，然后单击【粘贴】命令按钮，运行结果如图 7.19 所示。

图 7.18 剪贴板窗体设计

图 7.19 剪贴板图像复制

2. 使用 AxPictureClip 控件拷贝和粘贴图像

AxPictureClip 控件不是常规控件，而是一个 ActiveX 控件。因此，工具箱中没有该控件，要想使用该控件，必须把该控件添加到工具箱中，具体步骤如下：

（1）右键单击工具箱的空白处，在弹出的快捷菜单中选择【选择项】菜单项，则弹出“选择工具箱项”对话框，如图 7.20 所示。

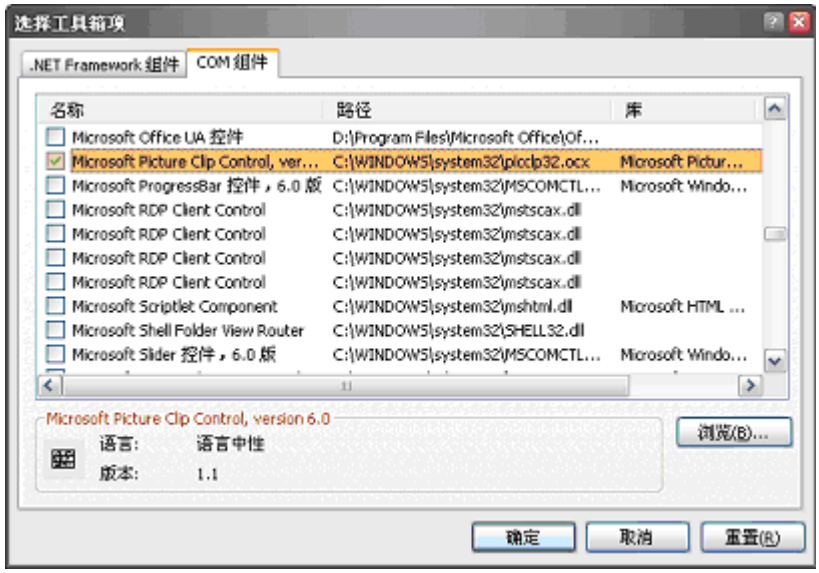

图 7.20 添加 AxPictureClip 控件

（2）在该对话框中的【COM 组件】选项卡中选择【Microsoft Picture Clip Control,version6】项，并单击【确定】按钮，该控件就添加到工具箱中了。

AxPictureClip 控件可用于随机访问方法或者枚举访问方法指定源位图中剪切区域如下：

- 使用随机访问方法来作为剪切区域选择源位图的任何部分。通过使用ClipX和ClipY属性指定的剪切区域左上角，ClipHeight 和 ClipWidth 属性确定剪切区域的区域。当想要查看位图的一部分此方法很有用。
- 使用枚举访问方法可以分成的行和列数指定源位图。结果是图片的统一矩阵单元编号 0、1、2 和等等，通过使用 GraphicCell 属性来访问单个单元。当源位图图像与要访问单独的调色板包含这种方法非常有用。

【例 7.9】使用 AxPictureClip 控件剪切和粘贴图像。

(1)建立如图 7.21 示的窗体。在窗体上天加两个图片框控件和两个命令按钮控件。

（2）双击【打开】命令按钮，输入如下代码，将图像打开。

```
private void button1_Click(object sender, EventArgs e)
{
    OpenFileDialog ofdlg = new OpenFileDialog();
    ofdlg.Filter = "BMP File(*.bmp)|*.bmp";
    if (ofdlg.ShowDialog() == DialogResult.OK)
    {
        Bitmap image = new Bitmap(ofdlg.FileName);
        pictureBox1.Image = image;
    }
}
```

（3）双击【复制与粘贴】命令按钮，输入如下代码，将图像复制到第二个图片框中。

```
private void button2_Click(object sender, EventArgs e)
{
  //使用枚举访问方法
  axPictureClip1.Picture = pictureBox1.Image;
  axPictureClip1.Cols = 6;//将图片分成6列
  axPictureClip1.Rows = 3;//将图片分成3行
  try
  {
     pictureBox2.Image = axPictureClip1.get_GraphicCell(0);//取出第一个图像块
  }
  catch (Exception ex)
  {
     MessageBox.Show(ex.ToString());
  }
}
```

（4）运行程序，单击【打开】命令按钮，选择一个图像文件打开，如图 7.21 所示，然后单击【复制与粘贴】命令按钮，运行结果如图 7.22 所示。

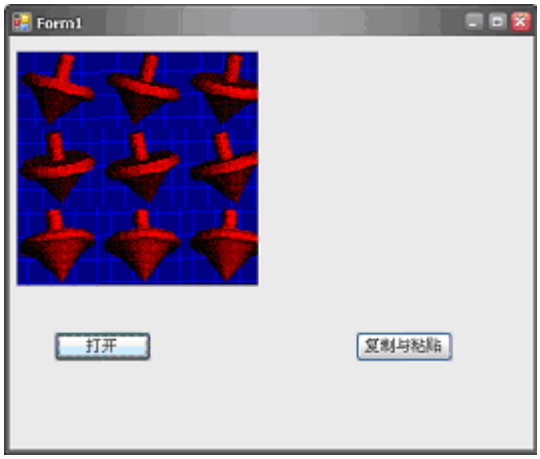

图 7.21 打开图片后效果

（5）使用随机访问方法。

也可以使用随机访问方法，只需将上述例子中的【复制与粘贴】命令按钮的响应方法改为如下代码即可：

```
private void button2_Click(object sender, EventArgs e)
{
   //使用随机访问方法
   axPictureClip1.Picture = pictureBox1.Image;
```

```
    axPictureClip1.ClipX = 15;
    axPictureClip1.ClipY = 15;
    axPictureClip1.ClipHeight = 50;
    axPictureClip1.ClipWidth = 50;
    pictureBox2.Image = axPictureClip1.Clip;//
}
```

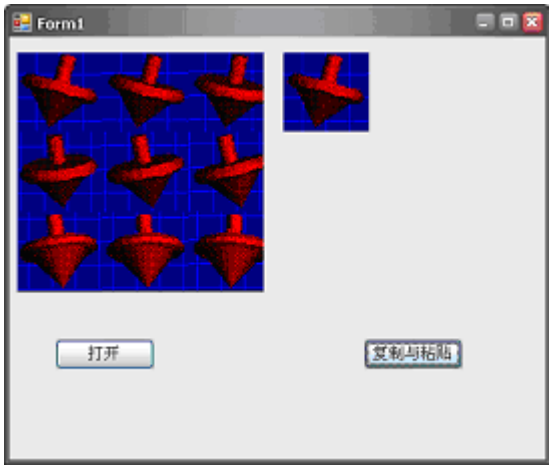

图 7.22 粘贴后效果

注意：如果没有 PICCLP32.OCX 控件，需要自行下载或复制 PICCLP32.OCX 到本机，并通过注册程序 REGSVR32 注册该组件。例如，如果该文件在 C:\WINDOWS\system32\路径下，可以通过如下命令行语句实现注册：REGSVR32　C:\WINDOWS\system32\PICCLP32.OCX

7.2.4　彩色图像处理

1．图像的分辨率

所谓分辨率就是指画面的解析度，由多少像素构成，数值越大，图像也就越清晰。我们通常所看到的分辨率都以乘法形式表现的，例如 800*600，其中“800”表示屏幕上水平方向显示的点数，“600”表示垂直方向的点数。图像分辨率越大，越能表现更丰富的细节。图像的分辨率决定了图像与原物的逼进程度，对同一大小的图像，其像素数越多，即将图像分割的越细，图像越清晰，称之为分辨率高，反之为分辨率低，分辨率的高低取决于采样操作。例如，对于一幅 256*256 分辨率的图像，采用变换的方法可以实现不同分辨率显示。

【例 7.10】将 256*256 分辨率的图像变换为 64*64 分辨率。

算法说明：将 256*256 分辨率的图像变换为 64*64 分辨率方法是将源图像分成 4*4 的子图像块，然后将该 4*4 子图像块的所有像素的颜色按 F(i,j)的颜色值进行设定，达到降低分辨率的目的。

建立一个如图 7.23 所示界面的项目，【分辨率】命令按钮的响应方法的代码设计如

下：

```
private void button3_Click(object sender, EventArgs e)
{
    Color c = new Color();
    //把图片框中的图片给一个 Bitmap 类型
    Bitmap box1 = new Bitmap(pictureBox1.Image);
    Bitmap box2 = new Bitmap(pictureBox1.Image);
    int r, g, b, size, k1, k2, xres, yres, i, j;
    xres = pictureBox1.Image.Width;
    yres = pictureBox1.Image.Height;
    size = 4;
    for (i = 0; i <= xres - 1; i += size)
    {
        for (j = 0; j <= yres - 1; j += size)
         {
           c = box1.GetPixel(i, j);
           r = c.R;
           g = c.G;
           b = c.B;
           //用 FromArgb 把整型转换成颜色值
           Color cc = Color.FromArgb(r, g, b);
           for (k1 = 0; k1 <= size - 1; k1++)
             {
               for (k2 = 0; k2 <= size - 1; k2++)
               {
                   if (i + k1 < pictureBox1.Image.Width)
                       box2.SetPixel(i + k1, j + k2, cc);
               }
           }
       }
     }
    pictureBox2.Refresh();//刷新
    pictureBox2.Image = box2;//图片赋到图片框中
}
```

输入图像分辨率为 256*256 像素，转换为 64*64 分辨率图像如图 7.23 所示。

图 7.23 分辨率

2．彩色图像变换灰度图像

（1）彩色位图图像的颜色

图像像素的颜色是由三种基本色颜色，即红（R）、绿（G）、蓝（B）有机组合而成的，称为三基色。每种基色可取 0～255 的值，因此由三基色可组合成（256*256*256）1677 万种颜色，每种颜色都有其对应的 R、G、B 值。例如，常见的 7 种颜色及其对应的 R、G、B 值如表 7.11 所示。

表 7.11 常见的 7 种颜色及其对应的 R、G、B 值

颜色名	R 值	G 值	B 值
红	255	0	0
蓝	0	0	255
绿	0	255	0
白	255	255	255
黄	255	255	0
黑	0	0	0
青	0	255	255
品红	255	0	255

（2）彩色图像颜色值的获取

在使用 C#系统处理彩色图像时，使用 Bitmap 类的 GetPixel 方法获取图像上指定像素的颜色值，格式为：

```
Color c = new Color();
c = box1.GetPixel(i,j);
```

其中，(i,j)为获得颜色的坐标位置。GetPixel 方法取得指定位置的颜色值并返回

一个长整型的整数。例如，求图片框 1 中图像在位置(i,j)的像素颜色值 c 时，可写为：

```
Color c=new Color();
c = box1.GetPixel(i,j);
```

（3）彩色位图颜色值分解

像素颜色值 c 是一个长整型的数值，占 4 个字节，最上位字节的值为“0”，其它 3 个下位字节依次为 B、G、R，值为 0～255。

从从值分解出 R、G、B 值可直接使用：

```
Color c =new Color();
c= box1.GetPixel(i,j);
r=c.R;
g=c.G;
b=c.B;
```

（4）图像像素颜色的设定

设置像素可使用 SetPixel 方法。用法如下：

```
Color c1=Color.FromArgb(rr,gg,bb);
Box2.SetPixel(i+k1,j+k2,c1);
```

【例 7.11】彩色图像生成灰度图像。

算法说明：将彩色图像像素的颜色值分解为三基色 R、G、B，求其和的平均值，然后使用 SetPixel 方法以该平均值参数生成图像。

其相应的代码设计如下：

```
c = b.GetPixel(i,j);
r = c.b;
g = c.G;
b = c.B;
cc = (int)((r+g+b)/3);
if (cc<0)cc=0;
f (cc>255)cc=255;
Color c1 = Color.FromArgb(cc,cc,cc);
B1.SetPixel(i,j,c1);
```

（1）在上例中增加一个【灰度图像】命令按钮。

（2）双击该按钮，编辑其响应方法的代码如下：

```
private void button4_Click(object sender, EventArgs e)
{
     Color c = new Color();
     //把图片框1中的图片给一个Bitmap类型
    Bitmap b = new Bitmap(pictureBox1.Image);
   Bitmap b1 = new Bitmap(pictureBox1.Image);
   int rr, gg, bb, cc;
   for (int i = 0; i < pictureBox1.Width; i++)
   {
      for (int j = 0; j < pictureBox1.Height; j++)
```

```
        {
            c = b.GetPixel(i,j);
            rr = c.R;
            gg = c.G;
            bb = c.B;
            cc = (int)((rr + gg + bb) / 3);
            if (cc < 0) cc = 0;
            if (cc > 255) cc = 255;
              //用FromArgb把整型转换成颜色值
              Color c1 = Color.FromArgb(cc, cc, cc);
              b1.SetPixel(i,j,c1);
        }
        pictureBox2.Refresh();//刷新
        pictureBox2.Image = b1;//图片赋给图片框2
    }
}
```

（3）运行程序，程序运行结果如图 7.24 所示。

图 7.24 灰度图像

3．灰度图像处理

【例 7.12】改善对比度。

算法说明：本例根据特定的输入输出灰度转换关系，增强了图像灰度，处理后图像的中等灰度值增大，图像变亮。

注意：本例中描述对比度改善的输入、输出灰度值对应关系的程序段为：

```
lev= 80;
wid =100;
for (x=0;x<256;x+=1)
{
    lut[x]=255;
}
```

```
for (x=lev;x<(lev+wid);x++)
{
   dm =((double)(x-lev)/(double)wid)*255f;
   lut [x] =(int)(dm);
}
```

（1）在窗体上添加一个“对比度”命令按钮。

（2）双击“对比度”命令按钮，编辑代码如下：

```
private void button5_Click(object sender, EventArgs e)
{
     Color c = new Color();
     Bitmap box1 = new Bitmap(pictureBox1.Image);
     Bitmap box2 = new Bitmap(pictureBox1.Image);
     int rr, x, m, lev, wid;
     int[] lut = new int[256];
     int [,,]pic=new int[600,600,3];
     double dm;
     lev = 80;
     wid = 100;
     for (x = 0; x < 256; x += 1)
      {
        lut[x] = 255;
     }
     for (x = lev; x < (lev + wid); x++)
     {
        dm = ((double)(x - lev) / (double)wid) * 255f;
        lut[x] = (int)dm;
      }
      for (int i = 0; i < pictureBox1.Image.Width - 1; i++)
      {
        for (int j = 0; j < pictureBox1.Image.Height; j++)
        {
          c = box1.GetPixel(i, j);
          pic[i, j, 0] = c.R;
          pic[i, j, 1] = c.G;
          pic[i, j, 2] = c.B;
        }
     }
     for (int i = 0; i < pictureBox1.Image.Width - 1; i++)
     {
         for (int j = 0; j < pictureBox1.Image.Height; j++)
        {
          m = pic[i, j, 0];
          rr=lut[m];
          Color c1 = Color.FromArgb(rr, rr, rr);
          box2.SetPixel(i,j,c1);
        }
        pictureBox2.Refresh();
        pictureBox2.Image = box2;
     }
  }
```

（3）运行程序，运行结果如图 7.25 所示。

图 7.25 改善对比度

本章小结

本章主要讲述了 C#下的图形图像基础知识，对图形的绘制，图像的处理和音频视频等多媒体的使用方法；在图片处理方面.NET 提供了一个 GDI+，功能十分强大，能完成对图像的全方位处理。

思考与练习（习题）

1. 绘制一个图形需要哪些基本步骤？
2. 在窗体上绘制图形有哪些方法？
3. 如何构造一个颜色对象？
4. 打开图像有哪些方法？
5. 如何转换图像格式？