C#

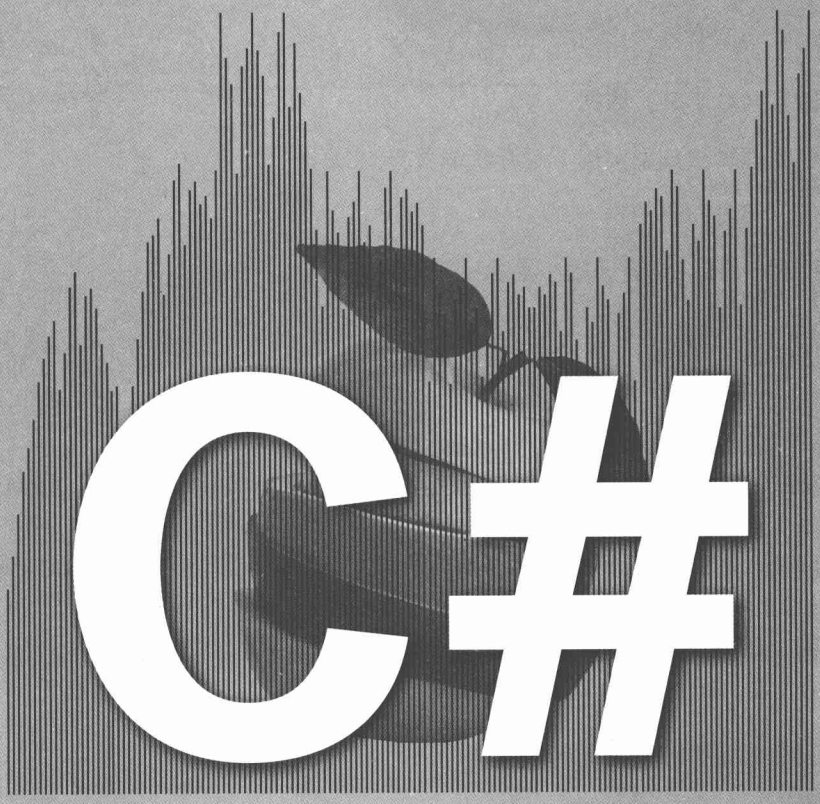
数字图像处理算法典型实例

赵春江 编著

CD-ROM

- **48** 种典型算法，涵盖C#数字图像处理的常用领域
- **50** 个典型实例，详细讲解其实现过程和实现效果
- **附赠** 本书全部源代码，可直接用于工程实践

人民邮电出版社
POSTS & TELECOM PRESS



数字图像处理算法典型实例

赵春江 编著

人民邮电出版社
北京

图书在版编目（CIP）数据

C#数字图像处理算法典型实例 / 赵春江编著. —北京：人民邮电出版社，2009.3
ISBN 978-7-115-19358-2

Ⅰ. C… Ⅱ. 赵… Ⅲ. C语言－数字图象处理－程序设计
Ⅳ. TP391.41 TP312

中国版本图书馆CIP数据核字（2008）第194809号

内 容 提 要

本书精选数字图像处理领域中的一些应用实例，以理论和实践相结合的方式，系统地介绍了如何使用C#进行数字图像处理。

全书共11章，分别讲述了图像的点运算、几何运算、数学形态学图像处理方法、频率变换、图像平滑与去噪、边缘检测、图像分割、图像压缩编码和彩色图像处理等相关技术。本书的光盘中附有相关章节的实现代码，可供广大的读者参考、阅读。

本书内容丰富，叙述详细，实用性强，适合于数字图像处理工作者阅读参考。

C# 数字图像处理算法典型实例

◆ 编　　著　赵春江
责任编辑　屈艳莲
执行编辑　黄　焱

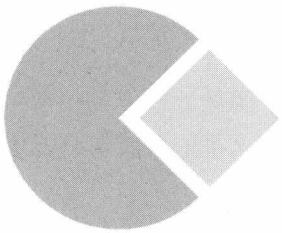
◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京顺义振华印刷厂印刷

◆ 开本：787×1092　1/16
印张：23
字数：565千字　　2009年3月第1版
印数：1－4 000册　　2009年3月北京第1次印刷

ISBN 978-7-115-19358-2/TP

定价：49.00元（附光盘）

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154



前　言

写作背景

图像处理是对图像进行分析、加工和处理，使其满足视觉、心理以及其他要求的技术。目前大多数的图像是以数字形式存储，因而图像处理在很多情况下是指数字图像处理。图像处理是信号处理的子类，另外它与计算机科学、人工智能等领域也有密切的关系。自从 20 世纪 60 年代以来，数字图像处理的理论和方法不断完善，已经在宇宙探测、遥感、生物医学、工农业生产、军事、公安、办公自动化、视频和多媒体系统等领域得到了广泛应用，并显示出广阔的应用前景，它已成为计算机科学、信息科学、生物学、医学等学科研究的热点。

为了实现和开发数字图像处理的算法，目前主流的应用软件是 C++。在过去的二三十年间，C++已经成为在商业软件的开发领域中使用最广泛的语言。数字图像处理可以被看成是二维数组的运算，应用 C++来完成正是利用了 C++的灵活多变、快速高效的运行能力和面向对象的编程思想等优点。

然而 C++在给程序员带来灵活性的同时，也牺牲了开发效率。用 C++开发应用程序往往需要较长的时间，用它来编写数字图像处理算法尤其如此。对于初学者来说，既要精通图像处理的各种算法，又要熟练掌握该种语言的语法结构，似乎是一件很难的事情。尽管有各种相关书籍可以参考，但对于不是很精通 C++的人来说，仍然会感到一头雾水，无从下手。C++（尤其是 Visual C++）指针的运用、各种自定义的数据类型、函数之间的相互调用、程序的流程走向等问题，始终困扰他们。即使熟悉图像处理的算法，有时也无法用 C++顺利地编写出程序来，不能直观地看出图像处理的效果，更进一步地影响了继续研究数字图像处理算法的信心。

C#是由微软公司开发的一种面向对象的新型编程语言，它是从 C 和 C++中派生出来的，保留了 C/C++原有的强大功能，并且继承了 C/C++的灵活性。同时，由于是 Microsoft 公司的产品，它又同 Visual Basic 一样具有简单的语法结构和高效的开发能力，可以使程序员快速地编写出基于.NET 平台的应用程序。

编写同一段代码，C#与 C++相比不仅开发周期短、代码量小，而且可读性好。C#同样适用于数字图像处理的算法开发，对于那些刚刚接触数字图像处理，对编程语言不是很熟悉，而又急需一种编程语言来实现、验证和开发某种算法的读者来说，C#是一个不错的选择。

然而到目前为止，市面上关于用C#开发数字图像处理的书籍少之又少，这不能不说是一个遗憾。但用C#开发数字图像处理的能力是无需质疑的。许多工程师都开发出了基于C#的图像处理软件，主要讲解C#语言的网站都有图像处理的专栏或详细介绍。对于初学者，它的简洁易懂、上手快、开发周期短等优点，具有更大的吸引力，不会因为编程语言上的障碍，而放弃对数字图像处理的研究。

本书系统而全面地介绍如何利用C#这一高级语言来实现数字图像处理中各种常用的算法，使读者把主要精力集中到图像处理算法本身的研究中去，既快又准确地运用高级语言实现算法。

本书主要内容

本书共分11章，分别介绍了数字图像处理中最常用的一些处理方法，如绘制直方图、灰度线性变换、灰度拉伸、直方图均衡化、直方图匹配、图像平移、图像镜像、图像旋转、数学形态学中的腐蚀运算、膨胀运算、开运算与闭运算、击中击不中变换、图像的频域变换、图像的噪声模型、中值滤波、均值滤波、灰度形态学滤波、小波变换滤波、高斯低通滤波、统计方法滤波、一阶导数边缘检测、二阶导数边缘检测、Canny边缘检测、小波变换边缘检测、灰度形态学边缘检测、金字塔边缘检测、阈值分割法、特征空间聚类分割法、松弛迭代分割法、Hough变换、哈夫曼编码、香农编码、香农-弗诺编码、行程编码、LZW编码、预测编码、傅里叶变换编码、小波变换编码、伪彩色处理，彩色图像的直方图均衡化算法、彩色图像平滑处理、彩色图像锐化处理、彩色图像边缘检测和彩色图像分割处理等内容。

读者对象

本书是为那些已经有一定C#编程基础，并掌握了一些基本的数字图像处理原理，而想进一步利用C#来进行数字图像处理的读者而编写的。

致谢

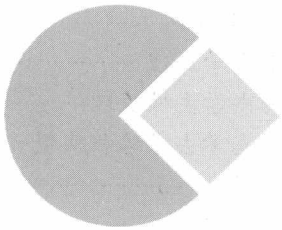
本书由赵春江负责编写并统编全部书稿，上海交通大学施文康教授为该书的出版作出了重要的贡献，在此特别感谢。

同时参与编写的还有谭敏、王俊、顾涓涓、胡学友、吉小军、邓勇、张广金、史贤荣、杨得彦、李积俊、黄楠、祁国军、包杰、赵龙、魏延文、卢广平、杨福财、解立龙、郭应世、唐永祥、柴鑫林、冯亮、任俊杰、李成良、何辰、王生成、高广越、韩晓冬、刘志宇、李松等，在此一并表示感谢。

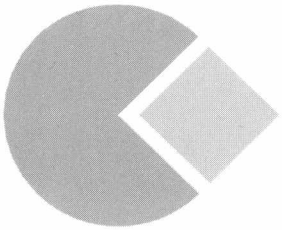
由于笔者水平有限，编写时间仓促，书中难免有疏漏和不足之处，恳请广大读者提出宝贵意见。本书责任编辑的联系方式是huangyan@ptpress.com.cn，欢迎来信交流。

编 者

2009年1月



目　录



第1章 绪　论

1.1 数字图像处理概述

尽管最近十几年来，数字计算机和通信技术并没有十分重大的突破，但人们还是怀着极大的热情关注信息技术这一领域。这是因为越来越便宜的个人电脑及互联网的广泛应用，使人们获得了海量的及时信息。而大多数的这类信息都被设计成更容易理解的可视的形式，如文本、图像和多媒体。

图像处理就是对图像进行分析和处理的一门很有趣也很重要的学科，它无处不在，从电视到 CT，从摄像到印刷，从机器人到遥感，可以说，数字图像处理技术已经从工业领域、实验室走向商业领域、艺术领域及办公室，甚至走向了人们的日常生活。之所以图像信息在我们生活中几乎所有领域都扮演着重要的角色，那是由于图像的直观、易懂、存储方便和信息量大等特点所决定的。

一般来讲，根据对图像处理的不同目的，数字图像处理可以分为 3 类。

- 改善图像质量：如进行图像的亮度和颜色变换，增强和抑制某些成分，对图像进行几何变换等，以提高图像的视觉效果。
- 提取图像特征：被提取的特征可以包括很多方面，如频域特征、灰度或颜色特征、边缘特征、区域特征、纹理特征、形状特征、拓扑特征和关系结构等，从而为分析图像提供便利。
- 存储传输图像信息：对图像数据进行变换、编码和压缩。

1.2 C#概述

1.2.1 C#特点

C#（C Sharp）是由微软公司所开发的一种面向对象，且运行于.NET Framework 之上的高级程序设计语言。C#看似基于 C++写成，但又融入其他语言如 Delphi、Java、VisualBasic 等。

（1）微软公司开发C#的初衷及C#的特点如下。

- C#旨在设计成为一种简单、现代、通用以及面向对象的程序设计语言。
- C#语言的实现，应提供对于以下软件工程要素的支持：强类型检查、数组维数检查、未初始化的变量引用检测、自动垃圾收集（一种自动内存释放技术），软件必须做到强大、持久，并具有较强的编程能力。
- C#语言应在分布式环境中的开发提供适用的组件开发应用。
- 为使程序员容易迁移到C#语言，源代码的可移植性十分重要，尤其是对于那些已熟悉C和C++的程序员而言。
- 对国际化的支持非常重要。
- C#适合为独立和嵌入式的系统编写程序，从使用复杂操作系统的大型系统到特定应用的小型系统均适用。
- 虽然C#程序在存储和操作能力需求方面具备经济性，但此种语言并不能在性能和尺寸方面与C语言或汇编语言相抗衡。

（2）相对于C和C++，C#在许多方面进行了限制和增强。

- 指针只能被用于不安全模式，大多数对象访问通过安全的引用实现，以避免无效的调用，并且有许多算法用于验证溢出，指针只能用于调用值类型以及受垃圾收集控制的托管对象。
- 对象不能被显式释放，而是当不存在被引用时通过垃圾回收器回收。
- 只允许单一继承，但是一个类可以实现多个接口。
- C#比C++更加类型安全，默认的安全转换是隐含转换，例如由短整型转换为长整型和从派生类转换为基类，而接口同整型，及枚举型同整型不允许隐含转换，非空指针（通过引用相似对象）同用户定义类型的隐含转换必需被显式地确定，不同于C++的复制构造函数。
- 数组声明语法不同。
- 枚举位于其所在的命名空间中。
- C#中没有模板，但是在C# 2.0中引入了泛型，并且支持一些C++模板不支持的特性，比如泛型参数中的类型约束，另一方面，表达式不能像C++模板中被用于类型参数。
- 属性支持，使用类似访问成员的方式调用。
- 完整的反射支持。

总之，C#学习起来要更容易，应用起来安全性更高。

C#并不被编译成为能够直接在计算机上执行的二进制本地代码。与Java类似，它被编译成为中间代码，然后通过.NET Framework的虚拟机（被称之为通用语言运行时）执行。所有的. NET编程语言都被编译成这种被称为MSIL的中间代码。因此虽然最终的程序在表面上仍然与传统意义上的可执行文件都具有“.exe”的后缀名。但是实际上，如果计算机上没有安装.NET Framework，那么这些程序将不能被执行。在程序执行时，.NET Framework将中间代码翻译成为二进制机器码，从而使它得到正确的运行。最终的二进制代码被存储在一个缓冲区中。所以一旦程序使用了相同的代码，那么将会调用缓冲区中的版本。这样如果一个.NET程序第二次被运行，那么这种翻译不需要进行第二次，速度明显加快。

1.2.2 WinForm 编程

WinForm 是.NET 开发平台中对 Windows Form 的一种称谓。.NET 为开发 WinForm 的应用程序提供了丰富的类库。这些 WinFrom 类库支持 RAD（快速应用程序开发），它们被封装在一个命名空间之中，这个命名空间就是 System.Windows.Forms。在该命名空间中定义了许多类，在开发基于.NET 的 GUI 应用程序的时候，就是通过继承和扩展这些类才使得程序有着多样的用户界面。本书开发的所有程序都是基于 WinForm 的。

一个典型的 Windows 窗体应用程序至少应该包含一个窗体。WinForm 应用程序中通常有一个窗体作为主窗体，它是该应用程序生命期内可能显示的其他窗体的父窗体或所有者，主菜单、工具栏、状态栏等都显示于该窗体内。当主窗体被关闭时，程序应该随即退出。

1.2.3 GDI+

GDI+是与.NET Framework 中的图形设备接口进行交互的入口。它使程序开发人员可以编写出与设备无关的受控应用程序，它的设计目的是要提供较高的性能、方便的使用以及对多语言的支持。如果要编写与监视器、打印机或文件等图形设备进行交互的.NET 应用程序，那么就必须使用 GDI+。

GDI+使得应用程序开发人员在输出屏幕和打印机信息的时候，无需考虑具体显示设备的细节，他们只需调用 GDI+库输出的类的一些方法即可完成图形操作，真正的绘图工作由这些方法交给特定的设备驱动程序来完成，GDI+使得图形硬件和应用程序相互隔离，从而使开发人员编写与设备无关的应用程序变得非常容易。

GDI+在 GDI 的基础上提供了明显地改进。最主要的特点是在 GDI+中，没有了句柄或设备上下文的概念，它被 Graphics 对象取代。Graphics 类提供了绘制不同图形对象的方法和属性，而且更易于使用。

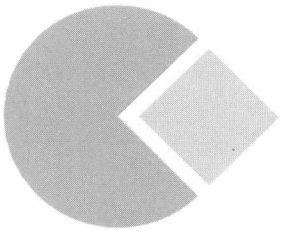
1.3 补充说明

- 本书并没有详细地讲解 C#的基本概念，也没有系统地介绍数字图像处理的基本理论和基本技术，而只是应用 C#这种高级语言实现数字图像处理中的几十种常用的算法。
- 本书侧重于实用性，力争用最简洁明了的语言介绍算法。在程序编写的过程中，既考虑效率，又考虑可读性，但更偏重于后者，从而使程序通俗易懂。
- 本书所用到的图像基本上都是图像处理领域内的标准测试图像。图像的文件格式均为 BMP 格式，大小都为 512×512，而且是 24 位彩色图像或者是 8 位灰度图像，即使是二值黑白图像，也仍然使用 8 位灰度图像代替（灰度值只有 0 和 255 两种，0 表示黑色，255 表示白色）。这样做的目的就是为了简化程序，而把精力放到具体算法的实现上，这更有利于初学者。因此，在本书的所有程序中，都默认为满足条件，而没有对上述参数进行判断。如果要把本书的程序应用到其他场合，只需添加几条判断图像格式和大小的语句即可。
- 上节已经说过，在本书的其他章节中每一章都是一个完整的 WinForm 程序，图像显

示在主窗体内，通过布置在主窗体内的按钮控件来促发各种图像处理的算法。由于很多算法需要人为地设置各种参数，所以还需要创建一个从窗体用来设置相关参数。处理后的图像替代原图像，同样显示在主窗体内。

- 本书程序是在以下电脑配置下完成的。

硬件	CPU	AMD Athlon 4400+
	内存	DDR Ⅱ-800 1G
软件	操作系统	Microsoft Windows XP SP2
	开发环境	Visual Studio 2005



第2章 C#数字图像处理的3种方法

本章通过彩色图像灰度化这一简单的图像处理算法来介绍C#数字图像处理的3种方法。通过编写第一个C#程序，读者不仅可以了解C#的特点，也能够掌握C#图像处理的基本方法，为以后编写更复杂的数字图像处理算法打下基础。

2.1 C#图像处理基础

Bitmap类、BitmapData类和Graphics类是C#图像处理中最重要的3个类，如果要用C#进行图像处理，就一定要掌握它们。

2.1.1 Bitmap类

Bitmap对象封装了GDI+中的一个位图，此位图由图形图像及其属性的像素数据组成。因此Bitmap是用于处理由像素数据定义的图像的对象。该类的主要方法和属性如下。

- GetPixel方法和SetPixel方法：获取和设置一个图像的指定像素的颜色。
- PixelFormat属性：返回图像的像素格式。
- Palette属性：获取或设置图像所使用的颜色调色板。
- Height属性和Width属性：返回图像的高度和宽度。
- LockBits方法和UnlockBits方法：分别锁定和解锁系统内存中的位图像素。在基于像素点的图像处理方法中使用LockBits和UnlockBits是一个很好的方式，这两种方法可以使我们通过指定像素的范围来控制位图的任意一部分，从而消除了通过循环对位图的像素逐个进行处理的需要。每次调用LockBits之后都应该调用一次UnlockBits。

LockBits方法的定义如下：

```
public BitmapData LockBits ( Rectangle rect, ImageLockMode flags, PixelFormat format );
```

LockBits方法使用3个类型，分别为Rectangle、ImageLockMode枚举和PixelFormat枚举的参数，并返回一个类型为BitmapData的对象。其中矩形参数定义了要在系统内存中锁定的位图的一部分；ImageLockMode枚举提供了对数据的访问方式，表2.1所示是它的成员；PixelFormat枚举表示像素的格式，表2.2所示是它的主要成员。

表 2.1 ImageLockMode 的成员

成 员	描 述
ReadOnly	位图的锁定部分只用于读操作
ReadWrite	位图的锁定部分用于读操作和写操作
UserInputBuffer	读取和写入像素数据的缓存由用户分配
WriteOnly	位图的锁定部分只用于写操作

表 2.2 PixelFormat 的主要成员

成 员	描 述
Format1bppIndexed	每个像素 1 位，使用索引颜色，因此颜色表中有两种颜色
Format4bppIndexed	每个像素 4 位，使用索引颜色
Format8bppIndexed	每个像素 8 位，使用索引颜色
Format16bppGrayScale	每个像素 16 位，共指定 65536 种灰色调
Format24bppRgb	每个像素 24 位，红色、绿色、蓝色分量分别使用 8 位，它们的顺序是蓝、绿、红
Format32bppArgb	每个像素 32 位，Alpha、红色、绿色、蓝色分量分别使用 8 位，这是默认的 GDI+颜色组合
Format64bppArgb	每个像素 64 位，Alpha、红色、绿色、蓝色分量分别使用 16 位
Indexed	索引颜色值，这些值是系统颜色表中颜色的索引，而不是单个颜色值

UnlockBits 方法使用一个由 LockBits 返回的类型为 BitmapData 的参数，它定义为：

```
public void UnlockBits ( BitampData bitmapdata );
```

2.1.2 BitmapData 类

BitmapData 对象指定了位图的属性，如下所示。

- Height 属性：被锁定位图的高度。
- Width 属性：被锁定位图的宽度。
- PixelFormat 属性：数据的实际像素格式。
- Scan0 属性：被锁定数组的首字节地址。如果整个图像被锁定，则是图像的第一个字节地址。
- Stride 属性：步幅，也称为扫描宽度。

如图 2.1 所示，数组的宽度并不一定等于图像像素数组的宽度，还有一部分未用区域。这是为了提高效率，系统要确定每行的字节数必须为 4 的倍数。例如一幅 24 位、宽为 17 个像素的图像，它需要每行占有的空间为 51（3×17）个字节，但 51 不是 4 的倍数，因此还需要补充 1 个

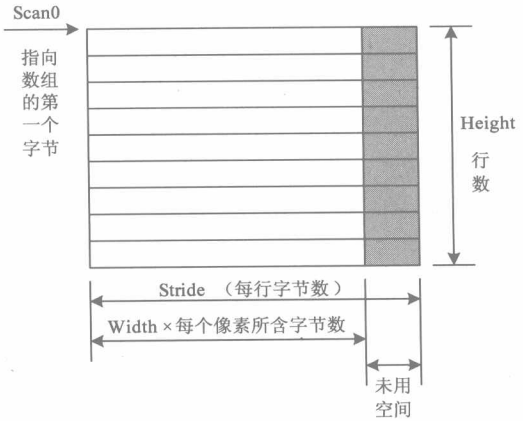


图 2.1 被锁定图像像素数组基本布局

字节，从而使每行的字节数扩展为 52（4×13，即 Stride=52），这样就满足了每行字节数是 4 的倍数的条件。需要扩展多少个字节不仅是由图像的宽度决定，而且还由图像像素的格式决定。

由于本书所选择的图像大小都为 512×512，因此无论是 24 位彩色图像，还是 8 位的灰度图像，都满足是 4 的倍数的条件，无需再扩展。如果处理的是任意宽度的图像，那么在进行下一行扫描的时候，就需要把扩展字节去除掉。

2.1.3 Graphics 类

Graphics 对象是 GDI+的关键所在。许多对象都是由 Graphics 类表示的，该类定义了绘制和填充图形对象的方法和属性。一个应用程序只要需要进行绘制或着色，它就必须使用 Graphics 对象。

Graphics 类的方法和属性很多，我们在后面遇到时再进行详细讲解。

2.2 彩色图像灰度化

为加快处理速度，在图像处理算法中，往往需要把彩色图像转换为灰度图像，本书所介绍的算法大多数都是在灰度图像上进行的。况且，在灰度图像上得到验证的算法，很容易移植到彩色图像上。

24 位彩色图像每个像素用 3 个字节表示，每个字节对应着 R、G、B 分量的亮度（红、绿、蓝）。当 R、G、B 分量值不同时，表现为彩色图像；当 R、G、B 分量值相同时，表现为灰度图像，该值就是我们所求的。

一般来说，转换公式有 3 种。第一种转换公式为：

$$Gray(i,j)=\left[R(i,j)+G(i,j)+B(i,j)\right]\div 3 \tag{2.1}$$

其中，$Gray(i,j)$为转换后的灰度图像在(i,j)点处的灰度值。该方法虽然简单，但人眼对颜色的感应是不同的，因此有了第二种转换公式：

$$Gray(i,j)=0.299\times R(i,j)+0.587\times G(i,j)+0.114\times B(i,j) \tag{2.2}$$

我们观察上式，发现绿色所占的比重最大，所以转换时可以直接使用 G 值作为转换后的灰度：

$$Gray(i,j)=G(i,j) \tag{2.3}$$

在这里，我们应用最常用的公式(2.2)，并且变换后的灰度图像仍然用 24 位图像表示。

2.3 彩色图像灰度化编程实例

下面我们就详细地讲解本书的第一个 C#图像处理程序。

2.3.1 使用图像

首先使用 Visual Studio 创建一个 Windows 应用程序。它的步骤为：打开 Visual Studio，

在主菜单中选择“文件 | 新建 | 项目”选项，然后在“项目类型”下选择“Visual C# Windows”，并在“模板”下选择“Windows 应用程序”，最后为项目选好路径和起好名字后，单击“确定”按钮即可，如图 2.2 所示。

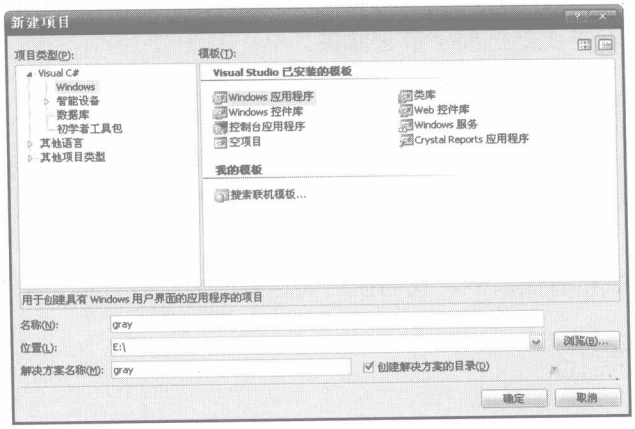


图 2.2 创建项目

在打开的程序主窗体内添加 3 个 Button 控件，如图 2.3 所示，其属性修改如表 2.3 所示。

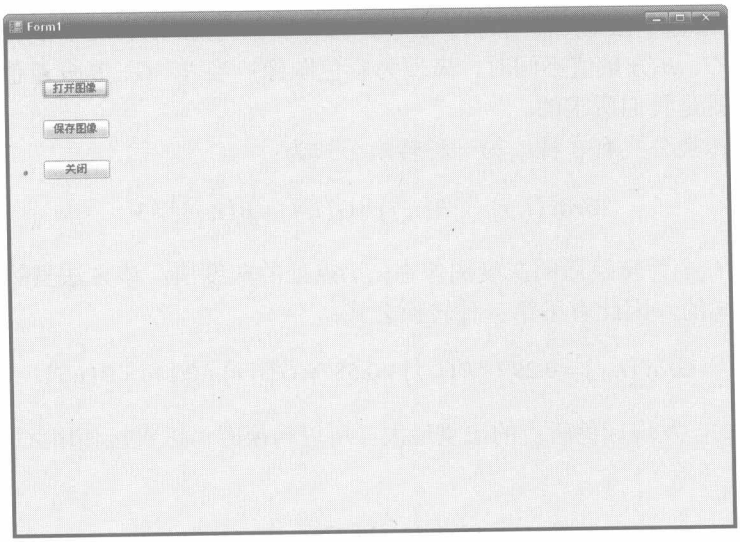


图 2.3 图像处理主窗体

表 2.3 所修改的属性

控 件	属 性	所修改内容
Form1	Size	800，600
Botton1	Name	open
	Text	打开图像
	Location	37，46
Botton2	Name	save

续表

控　件	属　性	所修改内容
Botton2	Text	保存图像
	Location	37，92
Botton3	Name	close
	Text	关闭
	Location	37，138

设置完控件，我们就进入程序的编写工作。在应用程序范围中分别定义一个字符串和一个 Bitmap 类型的数据成员。在 Form1 类中添加以下代码：

```
// 文件名
private string curFileName;
// 图像对象
private System.Drawing.Bitmap curBitmap;
```

分别双击在主窗体内所添加的 3 个 Botton 控件，为它们添加 Click 事件，代码如下：

```
// 打开图像文件
private void open_Click(object sender, EventArgs e)
{
    // 创建 OpenFileDialog
    OpenFileDialog opnDlg = new OpenFileDialog();
    // 为图像选择一个筛选器
    opnDlg.Filter = "所有图像文件 | *.bmp; *.pcx; *.png; *.jpg; *.gif;"+
      "*.tif; *.ico; *.dxf; *.cgm; *.cdr; *.wmf; *.eps; *.emf|" +
      "位图( *.bmp; *.jpg; *.png;...) | *.bmp; *.pcx; *.png; *.jpg; *.gif; *.tif; *.ico|" +
      "矢量图( *.wmf; *.eps; *.emf;...) | *.dxf; *.cgm; *.cdr; *.wmf; *.eps; *.emf";
    // 设置对话框标题
    opnDlg.Title = "打开图像文件";
    // 启用“帮助”按钮
    opnDlg.ShowHelp = true;

    // 如果结果为“打开”，选定文件
    if (opnDlg.ShowDialog() == DialogResult.OK)
    {
        // 读取当前选中的文件名
        curFileName = opnDlg.FileName;
        // 使用 Image.FromFile 创建图像对象
        try
        {
            curBitmap = (Bitmap)Image.FromFile(curFileName);
        }
        catch (Exception exp)
        {
            // 抛出异常
            MessageBox.Show(exp.Message);
        }
    }
    // 对窗体进行重新绘制，这将强制执行 paint 事件处理程序
    Invalidate();
}

// 保存图像文件
```