精通

Visual C++

数字图像处理典型算法及实现 第2版

求是科技　张宏林　编著

详细讲述了Visual C++数字图像处理典型算法及实现

对每种常用的数字图像处理方法，本书都提供了完整的源代码

本书内容丰富，叙述详细，实用性强

人民邮电出版社
POSTS & TELECOM PRESS

CD-ROM

精通

Visual C++

数字图像处理典型算法及实现 第2版

求是科技　张宏林　编著

人民邮电出版社

北京

图书在版编目（CIP）数据

精通 Visual C++数字图像处理典型算法及实现 / 张宏林编著. —2 版. —北京：人民邮电出版社，2008.7
ISBN 978-7-115-18049-0

Ⅰ. 精… Ⅱ. 张… 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 061856 号

内 容 提 要

本书主要讲述了 Visual C++数字图像处理典型算法及实现。全书共 12 章，分别介绍了数字图像编程基础，图像感知与获取，图像的点运算，图像的几何变换，图像的正交变换，图像的增强，数字图像腐蚀算法、膨胀算法、细化算法，图像边缘检测与提取、轮廓跟踪，图像分割，图像配准，图像复原和图像的压缩编码技术等内容。对每种常用的数字图像处理方法，本书都提供了完整的源代码。

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。

精通 Visual C++数字图像处理典型算法及实现（第 2 版）

◆ 编　　著　求是科技　张宏林
责任编辑　屈艳莲

◆ 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：36
字数：888 千字　　2008 年 7 月第 2 版
印数：7 001–11 000 册　　2008 年 7 月北京第 1 次印刷

ISBN 978-7-115-18049-0/TP

定价：75.00 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

第2版前言

随着人工智能和多媒体技术的发展，数字图像处理技术的应用越来越广泛。数字图像典型算法在数字图像处理技术中占有举足轻重的地位。由于 Visual C++本身就是图形开发界面，它提供了丰富的关于位图操作的函数，对开发图像处理系统提供了极大的方便。目前 Visual C++已经成为开发图像处理程序的主要开发工具。本书第 1 版主要讲述了 Visual C++数字图像处理典型算法及实现，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。

本书第 1 版出版以来，受到了广大读者的一致好评，很多读者都提出了很好的建议与意见。为了弥补前一版书的不足，我们经过精心策划与改编后推出了第 2 版图书。

第 2 版图书的特点主要表现在以下几个方面。

- 知识构架：在不影响知识构架的完整性的前提下，进一步优化了知识构架。
- 内容讲解：在内容讲解与表达上综合了读者、作者、编审的意见，做到字斟句酌。
- 图片清晰：更换了第 1 版图书中的一些比较模糊图片。
- 技术问题：修正了第 1 版图书中存在的部分容易引起读者误解的技术问题，使得第 2 版内容更加严谨。
- 细节调整：投入了大量的精力和时间，对容易引起读者阅读困难的细节进行了全面调整。

编　者

2008 年 5 月

前　言

数字图像处理技术从广义上可看做是各种图像加工技术的总称。它包括利用计算机和其他电子设备完成的一系列工作，如图像的采集、获取、编码、存储和传输，图像的合成和产生，图像的显示、绘制和输出，图像变化、增强、恢复和重建，特征的提取和测量，目标的检测、表达和描述，序列图像的校正，图像数据库的建立、索引、查询和抽取，图像的分类、表示和识别，3D景物的重建复原，图像模型的建立，图像知识的利用和匹配，图像和场景的解释和理解，以及基于它们的推理、判断、决策和行为规划等。

Visual C++是 Microsoft 公司推出的开发 Win32 环境程序，面向对象的可视化集成编译系统。它不仅具有程序框架自动生成、灵活方便的类管理、代码编写和界面设计集成交互操作、可开发多种程序等优点，而且通过简单的设置就可使其生成程序框架支持的数据库接口、OLE2、WinSock网络、3D 控制界面。由于 Visual C++本身就是一个图形的开发界面，提供了丰富的关于位图操作的函数，对开发图像处理系统提供了极大的方便。因此，它现在已经成为开发 Win32 程序，包括图像处理程序的主要开发工具。

本书分为 12 章，简要介绍如下。

第 1 章：主要介绍 Windows 下 Visual C++数字图像编程的基础知识。

第 2 章：主要介绍数字图像的采集、量化和表示。

第 3 章：主要介绍数字图像的点运算。

第 4 章：主要介绍数字图像的几何变换，通常包括图像的平移、镜像变换、转置、缩放和旋转等。

第 5 章：主要介绍数字图像处理的一些常见的频域处理方法。

第 6 章：主要介绍一些常用的数字图像增强技术。

第 7 章：主要介绍数字图像腐蚀、膨胀和细化算法。

第 8 章：主要介绍数字图像边缘检测与提取及轮廓跟踪。

第 9 章：主要介绍数字图像分割技术。

第 10 章：主要介绍图像配准中所用到的理论基础、配准方法，并给出了一个利用 Visual C++实现的图像配准示例。

第 11 章：主要介绍数字图像复原相关技术。

第 12 章：主要介绍数字图像压缩编码相关技术。

由于时间仓促和编写的水平有限，书中疏漏之处在所难免，敬请广大读者批评指正，同时欢迎广大读者提出宝贵意见和建议。

编　者

2006 年 5 月

目　录

第 1 章 Visual C++数字图像编程基础

本章将介绍在 Windows 下使用 Visual C++数字图像编程的基础知识，它是后面章节编程学习的基础。主要介绍的内容有 Windows 位图的结构和调色板的概念，GDI 位图与设备无关位图的概念，如何构造自己的 DIB 函数库，以及如何用 Visual C++编程来实现 Windows 位图的读写等。

1.1 数字图像处理概述

数字图像处理（Digital Image Processing）是指用计算机对图像信息进行的处理，因此也称为计算机图像处理（Computer Image Processing）。数字图像处理技术处理精度比较高，而且还可以通过改进处理软件来优化处理效果。总地来说，数字图像处理包括以下内容。

（1）点运算

点运算主要是针对图像的像素进行加、减、乘、除等运算。图像的点运算可以有效地改变图像的直方图分布，这对提高图像的分辨率以及图像的均衡都是非常有益的。

（2）几何处理

几何处理主要包括图像的坐标转换，图像的移动、缩小、放大和旋转，多个图像的配准和图像扭曲校正等。几何处理是最常见的图像处理手段，几乎任何图像处理软件都提供了最基本的图像缩放功能。图像的扭曲校正功能可以对变形的图像进行几何校正，从而得出准确的图像。

（3）图像增强

图像增强主要是突出图像中重要的信息，同时减弱或去除不需要的信息。常用方法有直方图增强和伪彩色增强等。

（4）图像复原

图像复原的主要目的是去除干扰和模糊，从而恢复图像的本来面目。例如去噪声复原处理。

（5）图像形态学处理

图像形态学是数学形态学的延伸，是一门独立的研究学科。利用图像形态学处理技术，可以实现图像的腐蚀、细化和分割等效果。

（6）图像编码

图像编码研究属于信息论中信源编码的范畴，主要是利用图像信号的统计特性和人类视觉特性对图像进行高效编码，从而达到压缩图像的目的。图像编码是数字图像处理中一个经典的研究范畴，有 60 多年的研究历史，目前已经制定了多种编码标准，如 H.261、JEPG 和 MEPG 等。

（7）图像重建

图像的重建起源于 CT 技术的发展，是一门新兴的数字图像处理技术，主要是利用采集的数据重建出图像。图像重建的主要算法有代数法、迭代法、傅立叶反投影法和使用最广泛的卷积反投影法等。

（8）模式识别

模式识别也是数字图像处理的一个新兴的研究方向，目前模式识别方法有 3 种，即统计识别法、句法结构模式识别法和模糊识别法。应用广泛的文字识别（OCR）技术就是应用模式识别技术开发出来的。

1.2 图像和调色板

目前，Windows 系列操作系统已经成为主流操作系统，它成功的一个重要原因是因为 Windows 有良好的人机交互界面和简便的操作。如果离开图形和图像，那么所有的 Windows 应用程序就会变得单调乏味。下面首先介绍一下图像是如何被显示出来的。

1.2.1 图像

普通的显示器屏幕是由许多的点构成的，这些点称为像素。显示时采用扫描的方式：电子枪每次从左到右扫描一行，为每个像素着色，然后再从上到下扫描整个屏幕，利用人眼的视觉暂留效应就可以显示出一屏完整的图像。为了防止闪烁，每秒要重复几十次扫描过程。我们常说的屏幕分辨率为 1024 × 768 像素，刷新频率为 85Hz，意思是每行扫描 1024 像素，一共要扫描 768 行，每秒重复扫描屏幕 85 次。一般刷新频率大于 80Hz 时，人眼感受不到因屏幕刷新而产生的闪烁，这种显示器被称为位映像设备。所谓位映像，就是指一个二维的像素矩阵，位图就是采用位映像方法显示和存储的图像。

图 1-1 所示是一幅普通的黑白位图，图 1-2 所示是被放大后的图，图中每个方格代表了一个像素，可以看到整个图像就是由这样一些黑点和白点组成的。

图 1-1 人脸

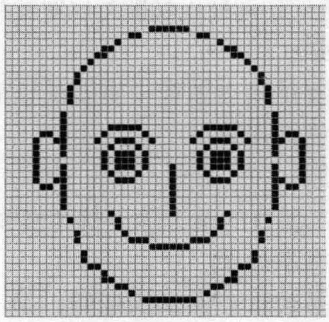

图 1-2 放大后的人脸位图

对于彩色图像，它的显示必须从三原色 RGB 概念说起。众所周知，自然界中的所有颜色都可以由红、绿、蓝（R、G、B）三原色组合而成。有的颜色含有红色成分多一些，其他成分少一些。针对含有红色成分的多少，可以人为地分成 0～255 共 256 个等级，0 级表示不含红色成分，255 级表示含有 100%的红色成分。同样，绿色和蓝色也可以被分成 256 级。这样，根据红、绿、蓝各种不同的组合可以表示出 256 × 256 × 256（约 1 600 万）种颜色。

表 1-1 所示为常见的一些颜色的 RGB 组合值。

表 1-1　　常见颜色的 RGB 组合

颜　色	红 色 成分	绿 色 成 分	蓝 色 成 分
黑色	0	0	0
白色	255	255	255
红色	255	0	0
绿色	0	255	0
蓝色	0	0	255
青色	0	255	255
紫色	255	0	255
黄色	255	255	0
灰色	128	128	128
橄榄色	128	128	0
深青色	0	128	128
银色	192	192	192

当一幅图中每个像素被赋予不同的 RGB 值时，就能呈现出五彩缤纷的颜色了，这就形成了彩色图像。

1.2.2　调色板

如果一幅图像的每个像素都用其 RGB 分量来表示，那么图像文件将变得非常庞大，实际上的做法不完全是这样的，可以先来看看一个简单的计算。

对一幅 200 × 200 的 16 色图像，它共有 40 000 个像素，如果每一个像素都用 *R*、*G*、*B* 3 个分量表示，则一个像素需要 3 个字节（因为每个分量有 256 个级别，要用 8 位，即 1 个字节来表示，所以 3 个分量需要用 3 个字节）。这样保存整个图像要用 200 × 200 × 3，即 120 000 字节。但是如果采用下面的方法，就能省很多字节。

对于 16 色图像，图中最多只有 16 种颜色，如果采用一个颜色表，表中的每一行记录一种颜色的 *R*、*G*、*B* 值，这样当表示一个像素的颜色时，只需要指出该颜色是在第几行，即指出的该颜色在表中的索引值便可以。例如，如果表的第 0 行为（255，0，0），表示红色，那么当某个像素为红色时，只需要标明 0 即可。通过颜色索引表来表示图像，16 种状态可以用 4 位（bit）表示，所以一个像素要用半个字节。整个图像要用 200 × 200 × 0.5，即 20 000 字节，再加上颜色表占用 3 × 16 = 48 字节，也不过 20 048 字节。这样一幅图像整个占用的字节数只是用前面方法表示的 1/6。

其实这张 RGB 表就是通常所说的调色板（Palette），它还有另外一种更确切的名称“颜色查找表 LUT（Look Up Table）”。在 Windows 位图中便用到了调色板技术，其实不仅仅是 Windows 位图，其他许多图像文件格式，例如“.pcx”、“.tif”、“.gif”等都用到了调色板。所以，很好地掌握调色板的概念是十分重要的。

还有一种情况，即真彩色图像（又叫做 24 位图像）的颜色种类高达 $256 \times 256 \times 256 = 2^{24} = 16\,777\,216$ 种，也就是包含上述提到的 *R*、*G*、*B* 颜色表示方法中所有的颜色。真彩色图像是说它具有显示所有颜色的能力，即可以包含所有的颜色。通常，在表示真彩色图时，每个像素直接用 *R*、*G*、*B* 这 3 个分量字节表示，而不采用调色板技术。这是因为如果使用调色板，表示一个像素颜色在调色板中的索引要用 24 位（因为共有 2^{24} 种颜色，即调色板有 2^{24} 行），这和直接用 *R*、*G*、*B* 这 3 个分量表示用的字节数一样，不但没有节省任何空间，还要加上一个 $256 \times 256 \times 256 \times 3$ 个字节的大调色板。所以真彩色图直接用 *R*、*G*、*B* 这 3 个分量表示。

1.2.3 色彩系统

前面介绍的 RGB 色彩系统是最常用的颜色系统，但在很多时候也会用到其他的色彩系统，常见的有以下几种。

1. CMY 色彩系统

CMY（Cyan、Magenta、Yellow）色彩系统也是一种常用的表示颜色的方式。计算机屏幕的显示通常用 RGB 色彩系统，它是通过颜色的相加来产生其他颜色，这种做法通常称为加色合成法（Additive Color Synthesis）。而在印刷工业上则通常用 CMY 色彩系统（一般所称的四色印刷 CMYK 则是再加上黑色），它是通过颜色相减来产生其他颜色的，所以称这种方式为减色合成法（Subtractive Color Synthesis）。图 1-3 所示为 RGB 与 CMY 两个色彩系统的关系图。

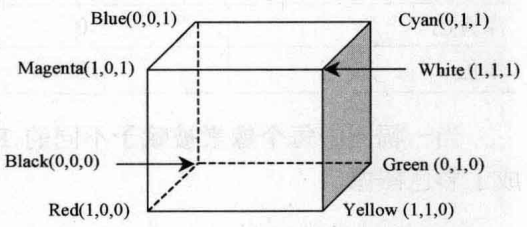

图 1-3 RGB 与 CMY 色彩系统关系图

2. YIQ 色彩系统

YIQ 色彩系统通常被北美的电视系统所采用（属于 NTSC 系统），这里 *Y* 不是指黄色，而是指颜色的明视度（Luminance），即亮度（Brightness）。其实 *Y* 就是图像的灰度值（Gray value）；而 *I* 和 *Q* 则是指色调（Chrominance），即描述图像色彩及饱和度的属性。RGB 与 YIQ 之间的对应关系如下。

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.956 & 0.621 \\ 1 & -0.272 & -0.647 \\ 1 & -1.106 & -1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

3. YUV 色彩系统

YUV 色彩系统被欧洲的电视系统所采用（属于 PAL 系统），其中 *Y* 和上面的 YIQ 色彩

系统中的 Y 相同，都是指明视度；U 和 V 虽然也是指色调，但是和 I 与 Q 的表达方式不完全相同。RGB 与 YUV 之间的对应关系如下。

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.148 & -0.289 & 0.437 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.140 \\ 1 & -0.395 & -0.581 \\ 1 & 2.032 & 0 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

4. YCbCr 色彩系统

YCbCr 色彩系统也是一种常见的色彩系统，JPEG 采用的色彩系统就是该系统。它是从 YUV 色彩系统衍生出来的（有人称 JPEG 采用的色彩系统是 YUV 系统，其实是错误的）。其中 Y 还是指明视度，而 Cb 和 Cr 则是将 U 和 V 做少量调整而得到的。RGB 色彩系统和 YCbCr 色彩系统之间的对应关系如下。

$$\begin{bmatrix} Y \\ Cb \\ Cr \\ 1 \end{bmatrix} = \begin{bmatrix} 0.2990 & 0.5870 & 0.1140 & 0 \\ -0.1687 & -0.3313 & 0.5000 & 128 \\ 0.5000 & -0.4187 & -0.0813 & 128 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \\ 1 \end{bmatrix} \quad \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 1.40200 & 0 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.77200 & 0 \end{bmatrix} \begin{bmatrix} Y \\ Cb-128 \\ Cr-128 \end{bmatrix}$$

1.2.4 灰度图

灰度图（Grayscale）是指只含亮度信息不含色彩信息的图像，就像我们平时看到亮度由暗到明的黑白照片，亮度变化是连续的。因此，要表示灰度图，就需要把亮度值进行量化。通常划分成 0～255 共 256 个级别，0 最暗（全黑），255 最亮（全白）。

BMP 格式的文件中并没有灰度图这个概念，但是可以很容易地用 BMP 文件来表示灰度图。方法是用 256 色的调色板，只不过这个调色板有点特殊，每一项的 RGB 值都是相同的，即 RGB 值从（0，0，0），（1，1，1）一直到（255，255，255），（0，0，0）是全黑色，（255，255，255）是全白色，中间的是灰色。这样，灰度图就可以用 256 色图来表示了。对于 $R=G=B$ 的色彩，带入 YIQ 或 YUV 色彩系统转换公式中可以看到其颜色分量都是 0，即没有色彩信息。

灰度图使用比较方便。首先 RGB 的值都一样；其次，图像数据即调色板索引值，也就是实际的 RGB 的亮度值；另外因为是 256 色的调色板，所以图像数据中一个字节代表一个像素；如果是彩色的 256 色图，图像处理后有可能会产生不属于这 256 种颜色的新颜色，所以，图像处理一般采用灰度图。为了将重点放在算法上，本书给出的程序如不作特殊说明，都是针对 256 级灰度图的。

1.3 GDI 位图

前面介绍了一些关于图像颜色和调色板等的基本概念，下面介绍如何在 Visual C++中使用图像。首先介绍一下如何使用 GDI 位图。

GDI 是图形设备接口（Graphics Device Interface）的缩写。GDI 位图是一种 GDI 对象，在 Microsoft 基本类（MFC）库中用 CBitmap 类来表示。在 CBitmap 类对象中，包

含一种和 Windows 的 GDI 模块有关的 Windows 数据结构，该数据结构是与设备相关的。应用程序可以得到 GDI 位图数据的一个备份，但是其中位图的安排则完全依赖于显示设备。我们可以将 GDI 位图数据在同一台计算机内的不同应用程序间任意传递，但是由于其对设备的依赖性，在不同类型计算机间的传递是没有任何意义的。

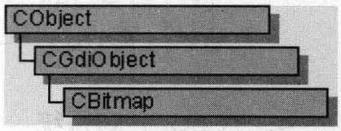

图 1-4　CBitmap 类的继承关系图

图 1-4 所示是 CBitmap 类的继承关系图。CBitmap 类封装了 Windows GDI 位图，同时提供了一些操作位图的成员函数。在使用 CBitmap 对象时，首先要创建一个 CBitmap 对象，然后把它选进设备环境中，再调用其成员函数进行处理，使用完毕后，把它从设备环境中选出并删除。

1.3.1　从资源中装入 GDI 位图

为了加载位图，可以使用 CBitmap 类的 LoadBitmap()成员函数。LoadBitmap 函数有两种调用方式：

一种是通过资源名称（由参数 lpszResourceName 指定）来加载指定的 GDI 位图，另外一种是通过资源 ID（由参数 nIDResource 指定）来加载指定的 GDI 位图。代码设置如下：

```
BOOL LoadBitmap( LPCTSTR lpszResourceName );
BOOL LoadBitmap( UINT nIDResource );
```

假设工程中已经添加了一个 ID 为 IDB_BITMAP1 的位图资源，则用下面的代码就可以显示该位图：

```
void CMyView::OnDraw(CDC* pDC)
{

    // CBitmap 对象
    CBitmap bitmap;

    // CDC 对象
    CDC dcMemory;

    // 加载资源
    bitmap.LoadBitmap(IDB_BITMAP1);

    // 创建内存设备环境
    dcMemory.CreateCompatibleDC(pDC);

    // 将位图选入内存设备环境中
    dcMemory.SelectObject(&bitmap);

    // 将内存设备环境复制到真正的设备环境中
    pDC->BitBlt(0,0,699,919,&dcMemory,0,0,SRCCOPY);

    // CDC 析构函数退出前将删除 dcMemory，位图选出

    // CBitmap 析构函数删除位图

}
```

程序中 BitBlt()函数将位图的像素从内存显示环境复制到显示器(或打印机)设备环境中，该函数十分有用，下面是它的函数原型：

```
BOOL BitBlt( int x, int y, int nWidth, int nHeight, CDC* pSrcDC, int xSrc, int ySrc, DWORD
dwRop )
```

上述代码中各个参数含义如下。

- x: 指定绘制区域的左上角 *x* 坐标（逻辑单位）。
- y: 指定绘制区域的左上角 *y* 坐标（逻辑单位）。
- nWidth: 指定绘制区域的宽度。
- nHeight: 指定绘制区域的高度。
- pSrcDC: 指向要复制位图所在的 CDC 对象的指针。
- xSrc: 指定原位图要绘制区域的左上角 *x* 坐标（逻辑单位）。
- ySrc: 指定原位图要绘制区域的左上角 *y* 坐标（逻辑单位）。
- dwRop: 指定绘制方式，取值如表 1-2 所示。

表 1-2　　dwRop 参数的取值表

参　数	含　义
BLACKNESS	将所有输出变成黑色
DSTINVERT	反转目标位图
MERGECOPY	合并模式和原位图
MERGEPAINT	用或（or）运算合并反转的原位图和目标位图
NOTSRCCOPY	将反转的原位图复制到目标
NOTSRCERASE	用或（or）运算合并原位图和目标位图，然后反转
PATCOPY	将模式复制到目标位图
PATINVERT	用异或（xor）运算合并目标位图与模式
PATPAINT	用或（or）运算合并反转的原位图与模式，然后用或（or）运算合并上述结果与目标位图
SRCAND	用与（and）运算合并目标像素与原位图
SRCCOPY	将原位图复制到目标位图
SRCERASE	反转目标位图并用与（and）运算合并所得结果与原位图
SRCINVERT	用异或（xor）运算合并目标像素和原位图
SRCPAINT	用或（or）运算合并目标像素和原位图
WHITENESS	将所有输出变成白色

程序运行的结果如图 1-5 所示。

注意　上面的代码产生的显示效果不错，打印预览也能看见图像，但在真正打印时就会发现什么也没打出来。这是因为代码是专门为了显示而构造的，无法将它选进与打印机兼容的内存设备环境中。如果需要打印位图，可以使用 DIB 函数，具体内容将在后面相关章节进行介绍。

图 1-5　从资源中装入 GDI 位图示例

1.3.2　伸缩位图

有时，我们想对位图进行放大或缩小的操作，这时就可以使用 StretchBlt()函数来显示位图。下面是该函数原型：

```
    BOOL StretchBlt( int x, int y, int nWidth, int nHeight, CDC* pSrcDC, int xSrc, int ySrc,
int nSrcWidth, int nSrcHeight, DWORD dwRop )
```

该函数和 BitBlt()基本上一致，只是多了两个参数 nSrcWidth 和 nSrcHeight，用来指定要复制原图像的宽度和高度。

下面更改第 1.3.1 小节的代码，调用该函数显示出两个缩小的图像。

```
    void CMyView::OnDraw(CDC* pDC)
    {

        // CBitmap 对象
        CBitmap bitmap;

        // CDC 对象
        CDC dcMemory;
```