第5章

图像存储

图像数据被采集至内存缓冲区后，机器视觉软件即可对其施加各种图像预处理、图像分析、图像处理和机器视觉算法。算法的设计是一个逐渐明晰的过程，有可能需要经过多次循环往复才能最终满足需求。因此，以图像文件的形式保存中间过程中的图像分析结论或最终处理结果对后续工作都极为重要。

图像数据可以多种文件格式保存至存储设备，常见的标准图像文件格式有 BMP (Bitmap)、TIFF (Tagged Image File Format)、PNG (Portable Network Graphics)、JPEG (Joint Photographic Experts Group)、JPEG2000 (Joint Photographic Experts Group 2000) 等。如果需要，也可开发保存浮点数、复数或 HSL 类型图像的专用图像文件格式或将连续多帧图像数据保存在 AVI (Audio Video Interleaved) 视频格式的文件中。不同格式的图像文件数据组织方式和压缩率各不相同，这也使得它们各自适用的场合差异较大，本章对机器视觉系统常用的图像文件读写方法进行介绍。

5.1 图像文件读写

无论是刚采集到的图像数据，还是经过机器视觉算法处理后得到的图像数据，都可以通过不同的方式组织，再以文件的形式保存至各种存储介质，如硬盘、U 盘或服务器等。保存图像的文件通常由文件头和紧随其后的图像数据构成。文件头包含了文件中像素数据组织方式的信息，如图像的水平和垂直像素分辨率、调色板等信息。而图像数据则包含了图像各个像素点的灰度或色彩信息等。多数情况下，图像数据都会以某种特定的方式组织存放，以节省存储空间或提高数据访问效率。不同格式的图像文件所使用的数据组织方式各不相同，图像文件的特性也因此各异。

LabVIEW 自身集成了对 PNG、JPEG 和 BMP 图像文件的支持，这意味着在没有安装 NI Vision 的情况下也可以使用 LabVIEW 自带的 VI 对这 3 种格式的图像文件直接进行读写操作。这些函数位于 LabVIEW 的 Graphics & Sound→Graphics Formats 函数选板中，如图 5-1 所示。

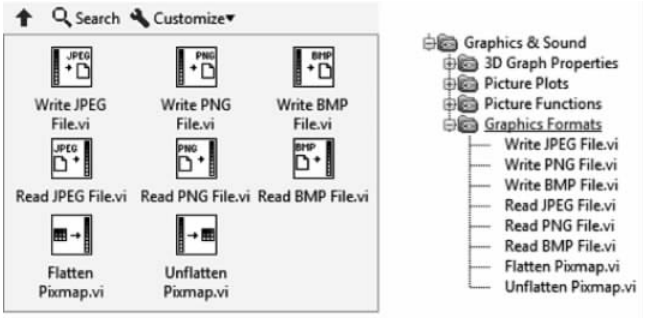

图 5-1 LabVIEW 自带的图像文件读写函数

LabVIEW 自带的这几个图像文件读写函数可以将图像文件中的数据读入内存,并用簇来组织这些数据。图 5-2 显示了使用 LabVIEW 自带的图像文件读写函数的实例。程序先用 Read JPEG File. vi 将数据读入内存,并用 Image Data 簇进行组织,其中图像数据被保存在一位数组中。用簇组织的图像数据被称为扁平数据(Flattened Data)。使用 LabVIEW 的 Draw Flattened Pixmap. vi 可以直接将扁平图像数据绘制到图片控件中进行显示(注意,图片控件并不能自动按图像尺寸进行显示)。也可以使用 Unflatten Pixmap. vi 将扁平图像数据转换为用二维矩阵表示的非扁平(Unflatten)图像数据,前提是要事先知道图像数据的编码方式(24 位、8 位、4 位或二值)。LabVIEW 为非扁平图像数据的显示也提供了 Draw Unflattened Pixmap. vi。

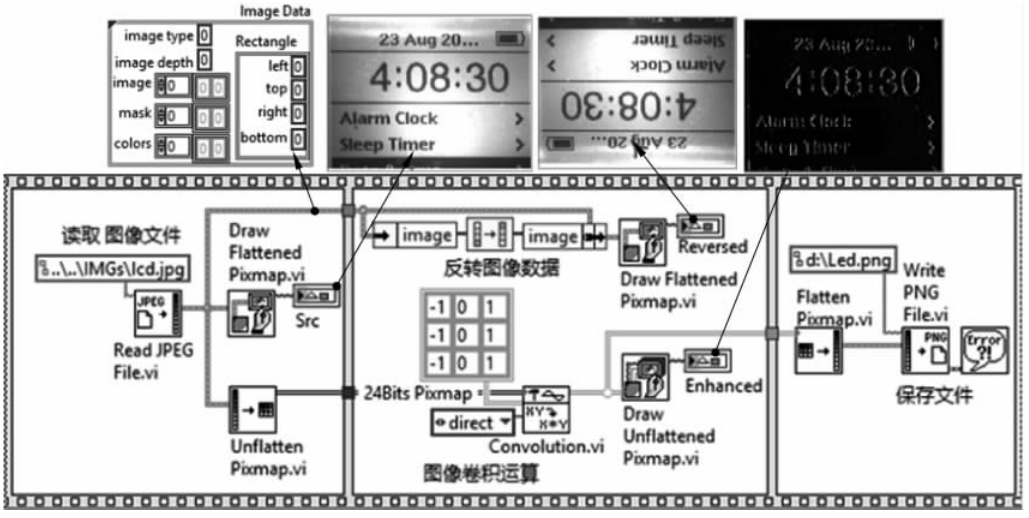

图 5-2 LabVIEW 自带的图像文件读写函数实例

程序将图像读入内存后,一方面使用 Draw Flattened Pixmap. vi 进行显示,另一方面用 Unflatten Pixmap. vi 将扁平的数据转换为非扁平的二维图像数据。紧接着,程序从扁平数据中提取一维像素数组,并将其顺序进行反转,经过重新更新内存中的图像数据并用 Draw Flattened Pixmap. vi 显示后,就会得到一幅对源图像沿水平方向进行反转的图像。Convolution. vi 可以对非扁平的二维矩阵图像数据进行卷积操作,若使用图中所示的尺寸

为 3×3 的卷积核对其处理，并用 Draw Unflattened Pixmap. vi 绘制处理结果，可以发现该卷积操作可提取图像中垂直的边缘。程序最后使用 Flatten Pixmal. vi 将卷积计算结果转换为扁平数据，并用 Write PNG File. vi 将其保存为 PNG 格式的文件。

上述例子给出了没有安装 NI Vision 时使用 LabVIEW 自带函数进行图像读写和处理的例子。使用这种方法时，图像数据被读入到数组中后才进行处理。由于数组在内存的栈区进行分配，而图像处理过程一般都要处理大量数据，因此使用 LabVIEW 自带的文件读写函数只能进行较简单的图像处理。对于机器视觉应用开发来说，通常需要进行大量的图像数据运算，这就需要使用 NI Vision 提供的文件读写和处理函数在内存的堆区进行各种操作。

NI Vision 不仅可以支持常见的 BMP、TIFF、PNG、JPEG、JPEG 2000 等标准图像文件格式读写，还支持可存放连续多帧图像的 AVI 视频文件格式，并且可保存浮点数、复数或 HSL 类型图像的 NI 专用 AIPD 图像文件格式。在通用文件格式中，PNG 文件具有保存机器视觉系统空间校准信息、模板匹配信息、图层以及其他用户自定义信息的能力。JPEG 和 JPEG 2000 具有较高的数据压缩率、而 AVI 文件可以在单独文件保存多帧图像。

不同格式的图像文件组织图像数据的方式和保存的图像类型各不相同。而不同类型的图像在像素位深度、色彩等方面的特点也千差万别。例如 BMP 文件只能保存 8 位灰度或 32 位 RGB 类型的图像，而 PNG 文件中则可以保存 8 位、16 位有符号、16 位无符号的灰度图像或 32 位和 64 位 RGB 类型的图像。AVI 文件可以将多个图像帧和语音同步交错组合在一起存储。AIPD 文件则支持任何类型的图像。表 5-1 显示了 NI Vision 支持的图像文件格式与图像类型之间的关系。

表 5-1 NI Vision 支持的常见图像文件格式与图像类型的关系

图像类型	位深度	数据类型	BMP	JPEG	AVI	TIFF	PNG	JPEG 2000	AIPD
灰度图像	8 位	Unsigned	√	√	√	√	√	√	√
	16 位	Unsigned				√	√	√	√
	16 位	Signed				√	√	√	√
	32 位	Floating							√
彩色图像	32 位	RGB	√	√	√	√	√	√	√
	64 位	RGB				√	√	√	√
	32 位	HSL							√
复数图像	64 位	Complex							√

图像经数字化后生成的数据量较大，因此在存储图像文件时通常都会对图像数据进行压缩，以最大限度地减少存储或传输所需的空间和带宽。根据压缩时是否损失信息，图像数据的压缩方式可分为无损压缩(lossless compression)和有损压缩(lossy compression)。

无损压缩方式通常会对图像数据进行扫描，并在不损失图像信息的前提下，按照某种更有效的方法重新组织并存储图像数据。经重新组织后的数据文件尺寸会明显减小。例如，使用无损压缩方式存储的图像文件可以将图像中重复的图案转换为一个样本和多个索引，或者根据像素强度的变化规律来存储图像数据，以缩小图像文件的尺寸。按照这种压缩方式存放的图像文件经解码后仍可以恢复所有原图像中的信息。

有损压缩算法通过丢弃不必要的信息，仅保留关键的有用信息来减小文件的大小。在

可接受的范围内,使用有损压缩可以有效地提高系统的处理能力和响应速度。例如,可以将图 5-3 所示的条形码图像尺寸缩小到仍足以分辨的程度保存,以减少机器视觉系统处理的数据量。再比如,若要提取图 5-4 中待测元件的边沿,可以先对图像进行二值化,去除不必要的色彩、灰度等冗余信息,仅保留必要的物体形状,以提高边沿提取的速度。

图 5-3 对条形码图像进行有损压缩

图 5-4 通过二值化方法去除不必要的信息

AIPD 文件是 NI 专门为机器视觉系统开发而设计的文件格式,它支持包括 32 位浮点数类型、32 位 HSL 类型以及 64 位复数类型在内的所有图像类型,其文件后缀名为“.apd”。AIPD 文件中的数据紧随在文件标识 AIPD 之后,不对数据进行任何压缩。NI 目前仍尚未公开 AIPD 文件的格式,但可以通过 LabVIEW 等 NI 的软件进行操作,并可将其转换为 BMP、TIFF、JPEG/JPEG 2000、PNG 等其他标准的图像格式。

使用传统文本语言进行机器视觉和图像处理分析系统开发时,工程人员往往要花费大量的时间和精力研究各种图像文件的结构并编写代码,这种不必要的重复劳动不仅增加了软件开发的成本,还降低了整个系统的稳定性并加长了项目周期。NI Vision 针对这一情况,对图像文件的读写等操作进行了封装和测试,提供了一套丰富的图像文件操作 VI。使用这些 VI,开发者无须关心文件格式等细节,而能够专注于机器视觉应用本身的开发。

NI Vision 提供的图像文件操作 VI 包括图像文件信息获取 VI、图像和视觉系统信息获取 VI、图像文件读写 VI 以及 AVI 文件操作 VI 等。这些 VI 位于 LabVIEW 的 Vision and Motion→Vision Utilities→Files 函数选板中,如图 5-5 所示。在 LabVIEW 中调用这些 VI 不仅可以快速读写 BMP、TIFF、JPEG/JPEG 2000、PNG、AIPD 等标准图像文件,还可以读取结构已知的非标准图像文件。当然也可以只获取文件中包含的与图像和机器视觉系统相关的信息,或使图像在不同的文件格式之间进行转换。

图 5-6 显示了在 LabVIEW 使用 NI Vision 读写 BMP 图像文件的程序代码,其他格式图像文件的读写方法与之类似。一开始,IMAQ Load Image Dialog 会显示文件选择对话框,提示操作人员选择要打开的图像文件。该 VI 与 LabvIEW 的标准对话框 VI 相似,但提供了对所选图像文件的预览功能。

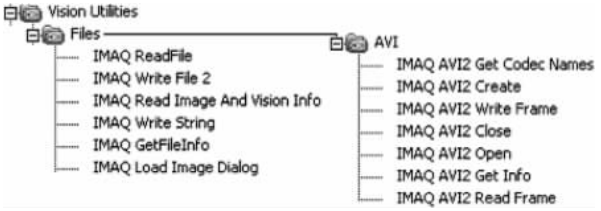

图 5-5 NI Vision 的图像文件操作函数

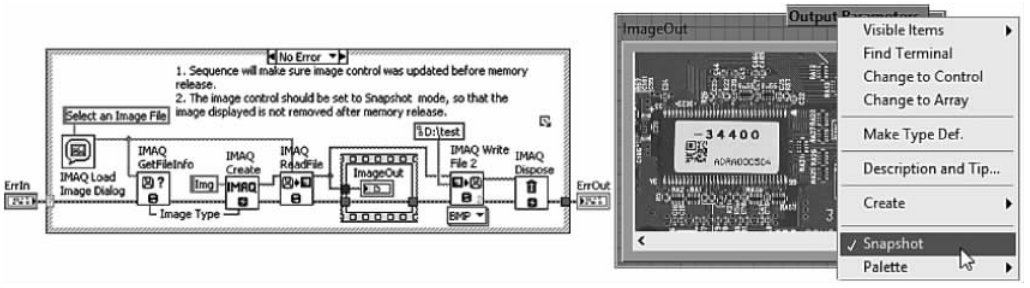

图 5-6 图像文件读写

IMAQ GetFileInfo 可以获取所选图像文件的信息，包括图像文件的类型、文件中所保存图像的类型(Image Type)、图像的分辨率(XY Resolution)以及文件数据类型(File Data Type)等参数。文件类型代表图像数据以何种格式的图像文件存放。如果文件为标准图像文件，则返回 BMP、TIFF、JPEG、JPEG 2000、PNG 或 AIPD 等标识字符串；若为非标文件，则返回 xxx 字符串。图像数据类型(File Data Type)代表了文件中所保存图像的位深度、颜色空间、数据的组织方式等信息，其返回值如表 5-2 所示。

表 5-2 图像类型的值

值	图像类型	说明
0	**Grayscale (U8)**	每像素 8 位、无符号类型的标准灰度图像
1	**Grayscale (16)**	每像素 16 位、带符号类型的灰度图像
2	**Grayscale (SGL)**	每像素 32 位、浮点型的灰度图像
3	**Complex (CSG)**	每像素 2×32 位、复数图像
4	**RGB (U32)**	每像素 32 位、无符号类型的 RGB+Alpha 彩色图像
5	**HSL (U32)**	每像素 32 位、无符号类型的 HSL+Alpha 彩色图像
6	**RGB (U64)**	每像素 64 位、无符号类型的 RGB+Alpha 彩色图像
7	**Grayscale (U16)**	每像素 16 位、无符号类型的标准灰度图像

File Data Type 的返回值是图像文件头中声明的像素数据类型，如 8 位灰度图像的返回值为 3。图 5-7 显示了 IMAQ GetFileInfo 的输入输出参数及其返回的文件数据类型。其中文件数据类型的返回值从 0 开始顺序递增。例如，返回值等于 5 时，代表文件中图像的像素使用 16 位无符号类型。

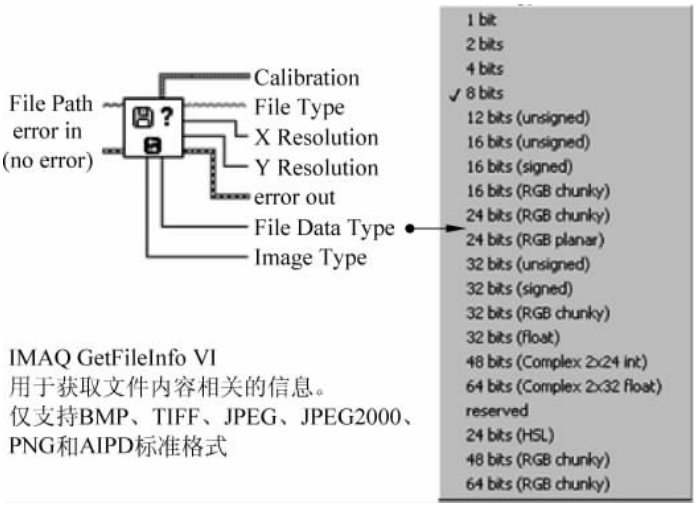

图 5-7 IMAQ GetFileInfo 及其返回的文件数据类型

IMAQ Create 根据图像的类型等信息为要读取的文件在内存的堆中分配空间。当调用该函数时,NI Vision 会在内存堆上先创建一个保存图像名和边界等属性信息的数据结构,但并不立刻为像素数据分配空间。只有等到图像尺寸改变时,才会自动根据实际情况为像素数据分配所需要的空间。图像尺寸改变通常在图像文件读取、图像采集开始或图像被重新采样时发生。IMAQ Create 在内存的堆区域为图像分配空间后,就可以返回指向该区域的引用。使用这一引用,就可以访问堆上的图像数据。如果图像处理代码会改变堆上的图像数据,一般需要重新在堆上重新分配空间,复制图像数据。

一旦为要读取的图像分配了足够的空间,IMAQ ReadFile 就可以将所选图像文件中包含的图像数据读入内存进行各种处理。IMAQ ReadFile 不仅可以将 BMP、TIFF、JPEG/JPEG 2000、PNG 及 AIPD 等标准图像文件中的数据读入内存,也可以将结构已知的非标准图像文件数据读入内存。而数据在内存中以何种类型存放取决于其输入参数 Image 引用所传递的图像类型。也就是说,无论所读取的图像文件为何种格式以及其中包含的图像为何种类型,IMAQ ReadFile 函数都会将其自动转换为 Image 引用所传递的图像类型。

例如,若在图 5-8 显示的代码中将 IMAQ Create 的输入参数 Image Type 人为指定为 7=Grayscale(U16),则无论所选择的图像文件包含何种类型的图像,IMAQ ReadFile 都先将解码后的数据自动转换为 16 位无符号的灰度图像,再输入到内存中。利用这一特点,开发人员始终可以直接从图像文件中得到算法需要的图像类型。当然,如果需要在内存中实现不同图像类型之间的转换,则可以使用 IMAQ Cast Image。

如果要使用 IMAQ ReadFile 读取结构已知的非标准图像文件,则需要事先通过 File Options 输入参数告诉该 VI 如何读取文件中的图像数据,包括要读取文件中数据的类型,文件中实际图像数据相对于文件头的偏移字节数,是否使用用户自定义的最大最小值对越界数据进行处理,以及 8 位以上的像素数据采用由低到高(Little Endian, Intel)还是由高到低的字节顺序(Big Endian, Motorola)存放等信息。图 5-9 显示了 IMAQ ReadFile 及其 File Options 参数的结构。关于各参数的详细意义及用法,读者可参考 NI Vision 的帮助文件。

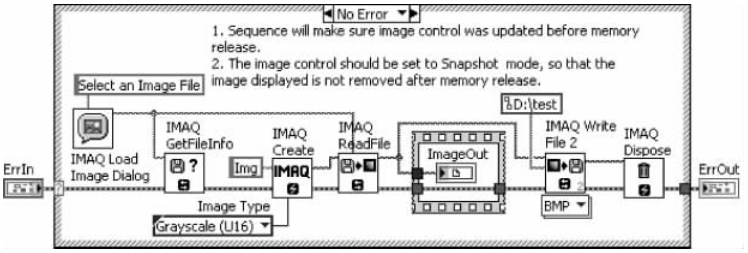

图 5-8 将文件中的图像读取为 16 位无符号灰度图像

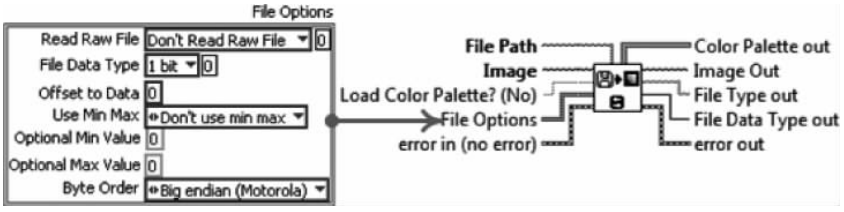

图 5-9 IMAQ ReadFile 及其 File Options 参数的结构

在内存完成对图像数据的分析处理后，就可以使用 IMAQ Write File 2 把处理后的图像及额外的与机器视觉系统相关的信息（如图层、模式匹配模板、系统校准以及文字说明等）保存到图像文件中。目前该 VI 共有 6 种实例可供选择，分别用于将数据保存为 BMP、TIFF、JPEG、JPEG 2000、PNG 图像文件或在 PNG 文件中保存机器视觉系统信息的情况。使用这些实例，开发人员就可以在各种图像文件格式之间自由转换。需要注意的是，图像文件格式的转换可能导致图像信息损失。例如，将采用无损压缩方式的 PNG 文件转换为采用有损压缩方式的 JPEG 文件时，部分图像的细节信息可能会被丢弃。

最后，将处理结果保存到文件并退出程序之前，必须释放在堆上为图像读取所分配的内存。另外，图 5-6 所示的程序使用了放置在 Sequence 中的 ImageOut 控件自动显示图像数据。由于 LabVIEW 以数据流驱动的方式运行，连接到 Sequence 上的错误警告簇可以有效避免图像在控件中还未完全显示时内存空间就已经被释放的情况。此外，还需要将图像显示控件的属性设置为 Snapshot，以确保释放内存中的数据后，控件中仍可显示之前的图像。

在进行机器视觉系统开发时，可针对不同应用场合，根据各种图像文件格式的特点择优选用。鉴于 NI Vision 支持的文件类型中 BMP、TIFF、JPEG/JPEG 2000、PNG 等标准文件在机器视觉系统中的通用性以及可保存多帧图像的 AVI 文件格式的特殊性，以下几节将对这些图像文件格式逐一进行简要介绍。

5.2 BMP 文件

BMP（全称 Bitmap）是 Windows 操作系统标准图像文件格式，使用较为广泛，其文件后缀名为“.bmp”。BMP 文件采用位映射存储格式，除了图像深度可选（1 位、4 位、8 位及 24 位）外，通常不采用其他任何压缩，因此它所占用的空间很大。

位图常分为与设备相关位图(Device Dependent Bitmap,DDB)和与设备无关位图(Device Independent Bitmap,DIB)两大类。DDB位图常见于早期的Windows系统(Windows 3.0以前)中。随着显示设备的多样化,DDB位图因其先天缺陷逐渐被淘汰。例如,由于它不能携带创建图像的原始设备的色彩信息,因此其他显示设备就不能真实准确地显示图像的色彩信息。DIB位图携带图像创建时的颜色和尺寸等信息,可以在不同的显示设备上按照创建时的情况重现,因此目前DIB已经基本取代了DDB。

DIB位图文件由文件头(Bitmap File Header)、位图信息头(DIB Header)、调色板(Color Table)和图像数据(Image Data)4部分组成,图5-10显示了一种DIB位图文件的结构。位图文件头包含BMP图像文件的类型、文件大小及图像数据起始位置相对于位图文件头的偏移(以字节数形式体现)。若用C语言来描述,位图文件头数据结构可定义为

```
typedef struct tagBITMAPFILEHEADER
{
    WORD bfType;                    // 文件的类型, 必须为 BMP
    DWORD bfSize;                   // 文件的大小, 以字节为单位
    WORD bfReserved1;               // 预留, 必须为 0S
    WORD bfReserved2;               // 预留, 必须为 0
    DWORD bfOffBits;                // 图像数据起始位置相对于位图文件头的偏移字节数
} BITMAPFILEHEADER;
```

位图信息头部分包含图像的分辨率、宽、高、压缩方式以及显示颜色等信息。位图信息头的数据结构可以用C语言定义为

```
typedef struct tagBITMAPINFOHEADER
{
    DWORD biSize;               // 本结构所占用字节数
    LONG biWidth;               // 位图的宽度,单位为像素
    LONG biHeight;              // 位图的高度,单位为像素
    WORD biPlanes;              // 目标设备的级别, 必须为 1
    WORD biBitCount;            // 像素位深度,只能是 1 (双色)、4 (16 色)、8 (256 色) 或 24 (真
                                // 彩色) 其中之一
    DWORD biCompression;        // 压缩类型,只能是 0 (不压缩)、1 (BI RLE8 压缩类型) 或 2 (BI
                                // RLE4 压缩类型) 其中之一
    DWORD biSizeImage;          // 文件所存储图像的大小, 以字节为单位
    LONG biXPelsPerMeter;       // 图像水平物理分辨率, 以每米像素数表示
    LONG biYPelsPerMeter;       // 图像垂直物理分辨率, 以每米像素数表示
    DWORD biClrUsed;            // 位图实际使用的颜色数量
    DWORD biClrImportant;       // 位图显示过程中的重要颜色数量
} BITMAPINFOHEADER;
```

调色板只在像素位深度为1位、4位和8位时存在。在这种情况下,它以表格形式包含了各种用于显示的图像颜色,而图像数据中只存放调色板中相应颜色的索引。调色板中每一个色彩都用一个RGBQUAD类型的数据来描述,该数据类型可以用C语言描述如下:

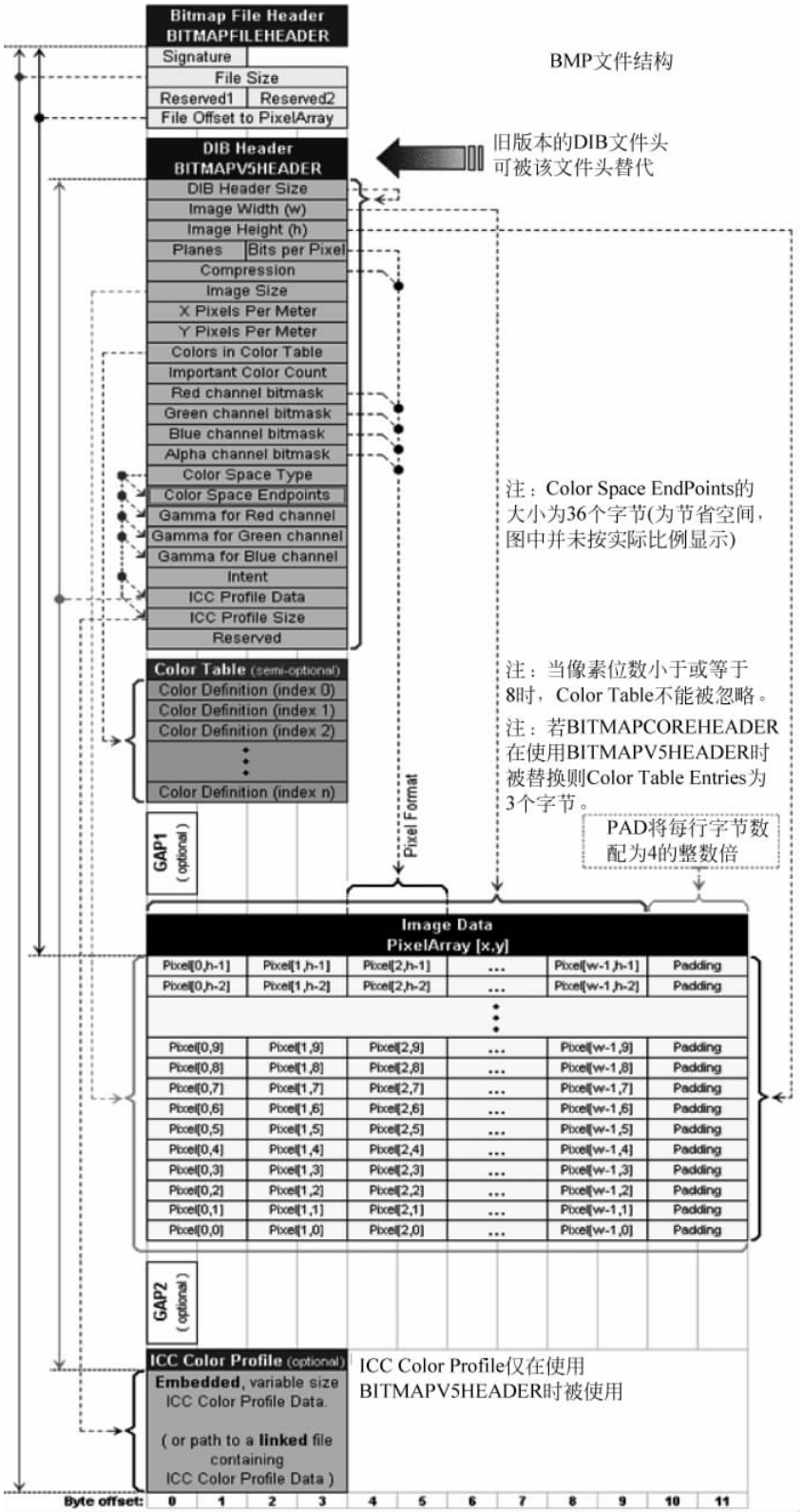

图 5-10 BMP 文件结构

```
typedef struct tagRGBQUAD
{
    BYTE rgbBlue;                    //蓝色的亮度 (取值范围为 0～255)
    BYTE rgbGreen;                   //绿色的亮度 (取值范围为 0～255)
    BYTE rgbRed;                     //红色的亮度 (取值范围为 0～255)
    BYTE rgbReserved;                //保留, 必须为 0
} RGBQUAD;
```

调色板中 RGBQUAD 数据的个数由位图信息头中表示位深度的 biBitCount 的值来确定。当 biBitCount 为 1、4、8 时，调色板中分别有 2、16 或 256 种颜色，且一般将最重要的颜色排在前面。由于 24 位真彩色位图直接使用 3 个字节代表像素的 RGB 值，因此当 biBitCount＝24 时，并不需要调色板。

位图信息头和调色板组合在一起统称为位图信息 BITMAPINFO，其数据结构可用 C 语言描述为

```
typedef struct tagBITMAPINFO
{
    BITMAPINFOHEADER bmiHeader;      //位图信息头
    RGBQUAD bmiColors[1] ;           //调色板
} BITMAPINFO;
```

图像数据部分按照从左到右、从下到上的顺序记录位图的每一个像素值。也就是说位图数据行的存储次序是颠倒的，即位图中的第一行对应的是位图文件中的最底行。像素数据以行为单位存储，且每行数据所占的存储长度总是被调整到 4B(32b)的整数倍。如果某行数据不足 4B 的整倍数，会将不足的位用 0 填充。

位图中单个像素的颜色值或其在调色板中的索引所占的字节数与像素的位深度相关。例如，biBitCount＝1 时，每字节可以包含 8 个像素颜色值索引，此时字节的最高位对应于最左边的像素；biBitCount＝4 时，每字节可包含 2 个像素的索引；biBitCount＝8 时，单个像素颜色索引就占 1 个字节；biBitCount＝24 时，每个像素颜色则直接用 3 个字节表示，从左到右的每一字节分别存储蓝、绿、红的颜色值。

BITMAPINFOHEADER 中的 biClrUsed 字段表示位图实际用到的颜色数量。如果它的值不为 0，则代表调色板中实际包含的颜色个数；而如果其值为 0，则代表调色板中颜色数为位深度的满量程(如位深度为 4 位时，表示包含 16 种颜色)或调色板不存在(对应 24 位真彩色图像)。DIB 位图数据可以压缩也可以不压缩。4 位和 8 位位图，可以采用游程长度编码(Run-Length Encoded，RLE)进行压缩。编码后的位图相应称为 RLE4 和 RLE8 位图。

NI Vision 提供的图像文件操作函数封装了大多数文件保存操作的细节，但仍保留了调色板(Color Palette)和压缩(Compress)参数让开发人员对图像文件保存过程进行控制。在 LabVIEW 中读写位图文件时，可以无须考虑保存操作的细节，就能轻松进行文件读写(参见 5.2 节程序代码)。需要注意的是 NI Vision 会将 8 位以下位深度的图像转换为 8 位位深度进行来操作，在计算机处理和存储能力飞速发展的今天，以这种方式处理已经不会对程序运行效率带来太大影响。此外，BMP 位图并不支持 1 位、4 位、8 位及 24 位以外的位深度，若要以其他位深度存储图像数据，则需要选择诸如 PNG 之类的图像文件类型。

5.3 TIFF 文件

标记图像文件格式(Tagged Image File Format,TIFF)最初是桌面扫描仪厂商为了达成一个公用的扫描图像文件格式在 20 世纪 80 年代中期制定的,其后缀名为".tif"或".tiff"。由于当时的桌面扫描仪处理能力有限,TIFF 最初只是一个二值图像格式。随着扫描仪的功能越来越强大,并且桌面计算机的磁盘空间越来越大,TIFF 才逐渐开始支持灰度图像和彩色图像。目前 TIFF 文件共有 4 种类型:二值图像类型 TIFF-B、黑白灰度图像 TIFF-G、带调色板的彩色图像 TIFF-P 以及 RGB 彩色图像 TIFF-R。

TIFF 通常由图像文件头(Image File Header,IFH)、图像文件目录(Image File Directory,IFD)、目录项(Directory Entry,DE)和图像数据 4 部分组成。TIFF 文件中的 IFH、IFD 和 DE 构建了一种逐级索引数据的方式。各个部分靠指针进行连接来组织数据,图 5-11 显示了 TIFF 文件的逻辑结构。

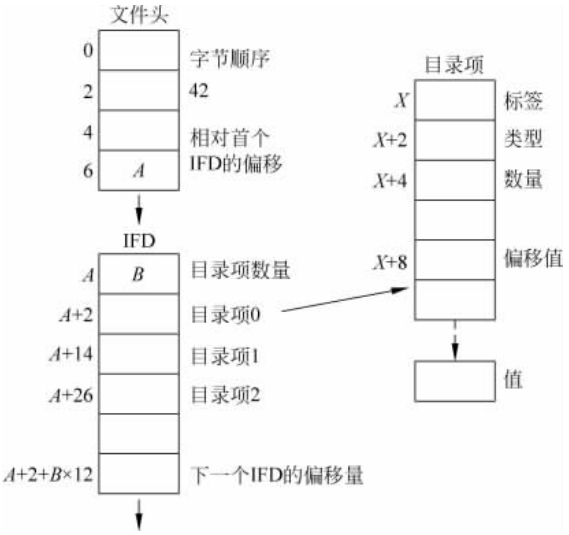

图 5-11 TIFF 文件逻辑结构

TIFF 文件以指向第一个 IFD 的 8 字节的 IFH 开始。IFH 在 TIFF 文件中是唯一的,它是正确解释文件其他部分的开始,其结构如表 5-3 所示。

表 5-3 IFH 的结构

域	字节	目 的
IFH	**0、1**	说明字节顺序: • 0x4949 表示字节顺序由低到高; • 0x4D4D 表示字节顺序由高到低
	2、3	TIFF 标志,目前为 0xZA
	4～7	第一个 IFD 的偏移量。可以在任意位置,但必须位于一个字的边界处,也就是说必须是 2B 的整数倍

IFD包含了指向图像的各种信息,同时也包含了指向实际图像数据的多个指针结构,如表5-4所示。IFD由一个2B的整数和其后的一系列12B标记目录项(DE)构成。这些DE分别标记了文件中各个实际图像数据块的位置及保存方式,因此DE也常被看作是数据标记(Tag)。IFD最后以一个长整型数结束,若其值为0,表示文件的DE项到此为止,否则该长整数为指向下一个IFD的偏移。

表5-4 IFD的结构

域	字 节	目 的
IFD	**0、1**	表示此IFD包含了多少个DE
	2～13	第一个DE
	14～25	第二个DE
	⋮	⋮
	2+n×12～n×12+6	长整型结束标志: • 若为0则表示文件指针块到此结束; • 若不为0,则指向下一个指针表的偏移

IFD中各个DE的结构如表5-5所示,其中直接存储实际图像数据,或者存储指向实际图像数据的指针。

表5-5 DE的结构

域	字 节	目 的
DE	**0、1**	DE的唯一标识,为254～321的整数
	2、3	数据类型:1=BYTE,2=CHAR,3=SHORT,4=LONG,5=RATIONAL(分子和分母各4B)
	4～7	数据长度或数据项个数。通过类型和数量可以确定存储此DE的数据所需要占据的字节数
	8～11	数据或指向数据的指针。如果数据的字节数少于4,则数据直接存于此,否则此处存放指向实际数据的指针

TIFF文件提供存储各种信息的完备的手段,是目前较流行的图像数据交换标准之一。TIFF文件自身携带多种有损和无损图像压缩算法,如JPEG压缩算法、游程编码、Zip和LZW(Lempel-Ziv-Welch)无损压缩算法等。TIFF格式设计时充分考虑了扩展性、方便性和可维护性,因此较为复杂,这不仅增加了程序设计的复杂度,也使得文件的读写速度较慢。

使用NI Vision读写TIFF文件与其他图像文件操作方法类似,图5-12显示了读写TIFF文件的程序代码。程序在图5-6所示的代码基础上增加了检测TIFF文件格式的功能,只有所读取的图像文件为TIFF文件时,才将读取的图像数据再次保存为test.tif文件(相当于文件重命名)。IMAQ Write File 2的TIFF格式文件保存实例在封装大多数文件操作细节的前提下,仍保留了调色板和TIFF文件选项(TIFF Option)两个参数,供开发人员对文件保存过程进行控制。

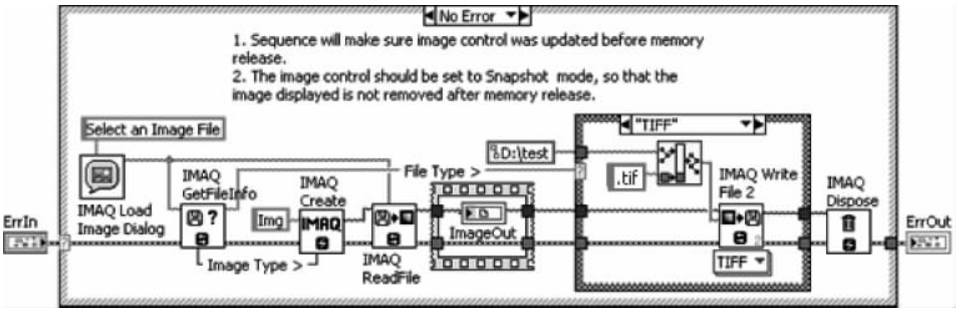

图 5-12 TIFF 文件逻辑结构

5.4 JPEG 文件

严格来说，JPEG 并非一种文件格式，它是由 ISO 和 CCITT 为静态图像所建立的第一个国际数字图像压缩标准。JPEG 主要是为了解决图像信息过于庞大的问题，它通过有损压缩方式去除图像中的冗余数据，可以用最少的磁盘空间仍能获得较好的图像质量。使用 JPEG 标准压缩的图像后缀名为“.jpg”或“.jpeg”。

JPEG 属于有损压缩，它通常通过分析图像，剔除人眼无法识别的信息，只保留连续色调图像中亮度和色相的变化，并将处理后的数据保存为 24 位的彩色图像。JPEG 是一种很灵活的格式，具有调节图像质量的功能，允许用不同的压缩比例对文件进行压缩。IMAQ Write File 2 的 JPEG 实例使用 Image Quality 参数指定对图像压缩的情况，它的值从 0～1000 可选。其中 0 级压缩比最高，图像品质最差，而 1000 级则图像的细节几乎无损。默认情况下，Image Quality 的值为 759。JPEG 格式压缩的主要是高频信息，对色彩的信息保留较好，比较适合应用于互联网，也普遍应用于需要连续色调的图像。

JPEG 文件大体上可以分成两个部分：标记码(Tag)和被压缩的数据。标记码由两个字节构成，其前一个字节是固定值 0xFF，后一个字节则根据不同意义有不同数值。在每个标记码之前还可以添加数目不限的无意义的 0xFF 填充，也就说连续的多个 0xFF 可以被理解为一个 0xFF，并表示一个标记码的开始。JPEG 常用的标记有 SOI (Start of Image)、APP(Application)、DQT(Define Quantization Table)、SOF0(Start of Frame)、DHT(Define Huffman Table)、DRI(Define Restart Interval)、SOS(Start of Scan)、EOI(End of Image)，这些标记在文件中以标记码形式出现，例如 SOI 的标记代码为 0xFFD8。在完整的节标记码后，一般直接存放该标记码对应的压缩数据流，记录关于文件的各种信息。表 5-6 列出了 JPEG 常用标记的标记码、占用的字节数和表示的意义。

表 5-6 JPEG 常用标记

标　记	字　段	字节数	说　明
SOI (Start of Image)	**0xFFD8**	2B	图像开始标记
APP0 (Application 0)	**0xFFE0**	2B	应用程序 0 标记符
	数据长度	2B	除标记符外 APP0 的长度
	标识符	5B	固定值 0x4A46494600,即字符串 JFIF0
	版本号	2B	一般是 0x0102,表示 JFIF 的版本号 1.2,也可能是其他版本
	X 和 *Y* 分辨率单位	1B	0：无单位； 1：点数/英寸； 2：点数/厘米
	X 分辨率	2B	
	Y 分辨率	2B	
	缩略图水平像素数	1B	无缩略图时值为 0
	缩略图垂直像素数	1B	无缩略图时值为 0
	缩略图 RGB 位图	3 的倍数	缩略图 RGB 位图数据
APP*n* (Application *n*)	**0xFFE*n***	2B	应用程序 *n*(1～15 可选)标记符，0xFFE1～0xFFEF
	数据长度	2B	除标记符外 APP*n* 的长度
	详细信息	可变	长度为数据长度－2B
DQT (Define Quantization Table)	**0xFFDB**	2B	定义量化表标记符
	数据长度	2B	除标记符外 DQT 的长度
	量化表	数据长度减去 2B	1) 首字节：高 4 位表示量化表的精度(0 代表 8 位,1 代表 16 位),低 4 位代表量化表 ID(取值范围：0～3)； 2) **64×(精度－1)**字节：量化表项,例如 8 位精度的量化表其表项长度为 64×(0+1)=64 字节； 3) 可包含最多 4 个量化表,ID 分别为 0～3
SOF0 (Start of Frame)	**0xFFC0**	2B	图像帧开始标记符
	数据长度	2B	除标记符外 SOF0 的长度
	精度	1B	每个数据样本的位数,通常是 8 位
	图像高度	2B	单位为像素
	图像宽度	2B	单位为像素
	颜色分量个数	1B	只有 3 个数值可选：1＝灰度图,3＝YC_bC_r 或 YIQ,4＝CMYK。 因 JFIF 文件中使用 YC_bC_r,故这里颜色分量数恒为 3
	颜色分量信息	分量数的 3 倍	字节 1：颜色分量 ID。 字节 2：高 4 位为水平采样因子,低 4 位为垂直采样因子。 字节 3：当前分量使用的量化表 ID

续表

标　记	字　段	字节数	说　明
DHT (Difine Huffman Table)	**0xFFC4**	2B	哈夫曼表定义标记符
	数据长度	2B	除标记符外 DHT 的长度
	哈夫曼表	2B	首字节：高 4 位表示表类型(0 代表直流 DC,1 代表交流 AC),低 4 位代表哈夫曼表的 ID。 **随后 16 字节**：不同位数的码字数量。 其后为编码内容
DRI (Define Restart Interval)	0xFFDD	2B	定义差分编码累计复位间隔的标记符
	数据长度	2B	除标记符外 DRI 的长度,固定为 4B
	MCU 块单元中的重新开始间隔	2B	设其值为 n,则表示每 n 个 MCU 块就有一个 RSTn 标记。第一个标记是 RST0,第二个是 RST1……RST7 后再从 RST0 重复。 如果没有 DRI,或间隔值为 0,则表示不存在重开始间隔和标记 RST
SOS (Start of Scan)	0xFFDA	2B	开始扫描标记符
	数据长度	2B	除标记符外 SOS 的长度
	颜色分量个数	1B	应该和 SOF 中的颜色分量数相同,恒为 3
	颜色分量信息	2B	字节 1：颜色分量 ID。 字节 2：高 4 位为直流分量使用的哈夫曼树编号,低 4 位为交流分量使用的哈夫曼树编号
	压缩图像数据	3B	字节 1：谱选择开始,固定为 0x00。 字节 2：谱选择结束,固定为 0x3F。 字节 3：谱选择,在基本 JPEG 中总为 0x00
EOI (End of Image)	0xFFD9	2B	图像数据结束标记符

此外,由于在 JPEG 文件中 0xFF 具有标志性的意义,所以在压缩数据流(真正的图像信息)中若出现 0xFF,就需要做特别处理。具体方法是,在数据 0xFF 后添加一个没有意义的 0x00。换句话说,如果在图像数据流中遇到 0xFF,应该检测其紧接着的字符,并根据获得的值分别进行处理：

(1) 0x00,表示 0xFF 是图像流的组成部分,需要进行译码。

(2) 0xD9,与 0xFF 组成标记 EOI,则图像流结束,同时图像文件结束。

(3) 0xD0～0xD7,则组成 RSTn 标记,且忽视整个 RSTn 标记,即不对当前 0xFF 和紧接着的 0xDn 两个字节进行译码,并按 RST 标记的规则调整译码变量。

(4) 0xFF,则忽视当前 0xFF,对后一个 0xFF 再作判断。

(5) 其他数值,则忽视当前 0xFF,并保留紧接的此数值用于译码。

JPEG 2000 作为 JPEG 的升级版,其压缩率比 JPEG 高约 30%左右,同时支持有损和无损压缩。JPEG 2000 格式有一个极其重要的特征在于它能实现渐进传输,即先传输图像的轮廓,然后逐步传输数据,不断提高图像质量,让图像由朦胧到清晰显示。此外,JPEG 2000 还支持所谓的感兴趣区域特性,可以任意指定影像上感兴趣区域的压缩质量,还可以选择指定的部分先解压缩。JPEG 2000 和 JPEG 相比优势明显,且向下兼容,因此可取代传统的

JPEG 格式。JPEG 2000 既可应用于传统的 JPEG 市场,如扫描仪、数码相机等,又可应用于新兴领域,如网络传输、无线通信等。

由于 JPEG 优异的品质和杰出的表现,它的应用也非常广泛。目前各类浏览器均支持 JPEG 这种图像格式,因为 JPEG 格式的文件尺寸较小,下载速度快,使得它成为网络上较受欢迎的图像格式。虽然压缩时图像亮度和色相的变化常会被保留,但 JPEG 采用的有损压缩算法却会导致图像不能完全被重建。因此,若需要在重建的图像上进行精确测量,一般并不建议使用 JPEG 格式。

5.5 PNG 文件

PNG(Portable Network Graphic)格式是 20 世纪 90 年代中期开始开发的图像文件存储格式,其目的是企图替代 GIF 和 TIFF 文件格式,同时增加一些 GIF 文件格式所不具备的特性。使用 PNG 格式来存储灰度图像时,图像的位深度可达 16 位;存储彩色图像时,图像的深度可多达 48 位,并且还可存储多达 16 位的 Alpha 通道数据。PNG 图像与硬件设备无关,且可在一个文件中保存多幅图像。PNG 使用从 LZ77 派生的无损数据压缩算法,而且还可以在不损伤图像数据的前提下,保存一些图像需要的文本辅助信息。

PNG 文件格式保留了 GIF 文件格式的以下特性:

- 支持 8 位调色板(256 种颜色)。
- 支持流式读写方式(streamability)。图像文件格式允许连续读出和写入图像数据,这个特性适合于在通信过程中生成和显示图像。
- 支持逐次逼近显示(progressive display)。这种特性可使在通信链路上传输图像文件的同时就在终端上显示图像,把整个轮廓显示出来之后逐步显示图像的细节,也就是先用低分辨率显示图像,然后逐步提高它的分辨率。
- 支持透明性(transparency)。这个性能可使图像中某些部分不显示出来,用来创建一些有特色的图像。
- 支持辅助信息(ancillary information)。这个特性可用来在图像文件中存储一些文本注释信息。

PNG 文件格式中增加了下列 GIF 文件格式所没有的特性:

- 每个像素可使用多达 48 位来表示真彩色。
- 每个像素可使用多达 16 位来表示灰度。
- 可为灰度图和真彩色图添加 Alpha 通道。
- 可保存图像的 Gamma 信息。
- 使用循环冗余码校验(Cyclic Redundancy Code,CRC)检测损害的文件。
- 加快图像显示的逐次逼近显示方式。
- 标准的读/写工具包。
- 可在一个文件中存储多幅图像。

PNG 图像格式文件由 PNG 文件署名域(PNG File Signature)和按照特定结构组织的数据块(chunk)组成。PNG 文件署名用来识别该文件是不是 PNG 文件,它共包含 8 个字

节，如表 5-7 所示。

表 5-7 PNG 文件署名结构

字节值(十六进制)	目 的
89	通过将最高位置 1 检测不支持 8 位数据的传输系统，并减少文本文件与 PNG 文件相互误读的几率
50 4E 47	"PNG"的 ASCII 码，用于识别文件格式
0D 0A	DOS 风格的行结束符(CRLF)，用于 DOS/UNIX 环境下识别行结束
1A	用于 DOS 环境下使用文件结束符时终止图像显示
0A	UNIX 风格的行结束符，用于检测行转换结束

PNG 定义了关键数据块(Critical Chunk)和辅助数据块(Ancillary Chunk)两种类型的数据块。关键数据块为标准的数据块，它定义了 PNG 文件必须包含的标准数据块，PNG 读写软件也都必须支持这些数据块。辅助数据块为可选的数据块，虽然 PNG 文件规范没有要求 PNG 编译码器对可选数据块进行编码和译码，但规范提倡支持可选数据块。PNG 文件中每个数据块的结构如表 5-8 所示。

表 5-8 PNG 数据块结构

名 称	字节数	目 的
长度 (Length)	4B	指定数据块中数据域的长度，其长度不超过 $2^{31}-1$B
数据块类型码 (Chunk Type Code)	4B	数据块类型码由 ASCII 字母(A～Z 和 a～z)组成
数据块数据 (Chunk Data)	可变	存储按照 Chunk Type Code 指定的数据
循环冗余检测 (cyclic redundancy check)	4B	由数据块类型码和数据块数据计算得到，用于检测是否有存储错误

PNG 格式共定义了 4 种标准的关键数据块：图像头数据块(Image Header Chunk，IHDR)、调色板数据块(Palette Chunk，PLTE)、图像数据块(Image Data Chunk，IDAT)和图像结束数据块(Image Trailer Chunk，IEND)。调色板数据块 PLTE 对于索引彩色图像来说必须存在，而对于真彩色图像或带 Alpha 通道数据的真彩色图像来说为可选。当图像为灰度图时，它没有存在的意义。

图像头数据块(IHDR)包含 PNG 文件中存储的图像数据的基本信息。它必须作为第一个数据块出现在 PNG 数据流中，而且一个 PNG 数据流中只能有一个文件头数据块。文件头数据块由 13 个字节组成，它的格式如表 5-9 所示。

表 5-9 图像头数据块(IHDR)结构

名 称	字节数	目 的
图像宽度 (Width)	4B	图像宽度，以像素为单位
图像高度 (Height)	4B	图像高度，以像素为单位

续表

名　　称	字节数	目　　的
像素位深度 (Bit Depth)	1B	彩色索引图像：每通道为 1、2、4 或 8 位可选； 灰度图像：每通道为 1、2、4、8 或 16 位可选； 真彩色图像：每通道 8 或 16 位可选； 带 Alpha 通道的灰度图像，每通道 8 或 16 位可选； 带 Alpha 通道的真彩色图像，每通道 8 或 16 位可选
颜色类型 (Color Type)	1B	0：灰度图像，1、2、4、8 或 16 位； 1：灰度索引图像； 2：真彩色图像，8 或 16 位； 3：索引彩色图像，1、2、4 或 8 位； 4：带 Alpha 通道的灰度图像，每通道 8 或 16 位可选； 5：带 Alpha 通道的灰度索引图像； 6：带 Alpha 通道的真彩色图像，每通道 8 或 16 位可选
压缩方式 (Compression Method)	1B	压缩方法(LZ77 派生算法)
滤波方式 (Filter Method)	1B	滤波方式
隔行扫描方式 (Interlace method)	1B	0：非隔行扫描 1：Adam7(由 Adam M. Costello 开发的 7 遍隔行扫描方法)

调色板数据块(PLTE)包含有与索引彩色图像(indexed-color image)相关的彩色变换数据，它仅与索引彩色图像有关，而且要放在图像数据块(IDAT)之前。真彩色的 PNG 数据流也可以有调色板数据块，目的是便于非真彩色显示程序用它来量化图像数据，从而显示该图像。调色板数据块结构如表 5-10 所示。

表 5-10　调色板数据块(PLTE)结构

名　　称	字节数	目　　的
红(Red)	1B	0＝黑，255＝红
绿(Green)	1B	0＝黑，255＝绿
蓝(Blue)	1B	0＝黑，255＝蓝

图像数据块(IDAT)用来存储实际的图像数据，在 PNG 数据流中可包含多个连续顺序的图像数据块。图像结束数据(IEND)用来标记 PNG 文件或者数据流已经结束，它必须放在文件的尾部。除了表示数据块开始的 IHDR 必须放在最前面，表示 PNG 文件结束的 IEND 放在最后面之外，其他数据块的存放顺序并没有严格限制。

除了 4 个标准的关键数据块外，PNG 文件格式规范还制定了 10 个辅助数据块和一些专用公共数据块(special-purpose public chunk)。表 5-11 汇总了这些数据块的符号、特性及位置限制。PNG 不仅支持较高的像素分辨率，适合传输，数据组织方式灵活，而且还是目前为数不多的开源文件格式。

表 5-11 PNG 的数据块

数据块符号	数据块名称	标准数据块	辅助数据块	专用公共数据块	支持多数据块	可选否	位置限制及说明
IHDR	文件头数据块	√					第一块
cHRM	基色和白色度点数据块(Primary Chromaticities & white point)		√			√	在 PLTE 和 IDAT 之前
gAMA	图像 Gamma 数据块(Image Gamma)		√			√	在 PLTE 和 IDAT 之前
sBIT	样本有效位数据块(Significant Bits)		√			√	在 PLTE 和 IDAT 之前
PLTE	**调色板数据块**	√				√	在 IDAT 之前
bKGD	背景颜色数据块(Background)		√			√	在 PLTE 之后,IDAT 之前
hIST	图像直方图数据块(Image Histogram)		√			√	在 PLTE 之后,IDAT 之前
tRNS	图像透明数据块(Transparency)		√			√	在 PLTE 之后,IDAT 之前
oFFs	(专用公共数据块)			√		√	在 IDAT 之前
pHYs	物理像素尺寸数据块(physical pixel dimensions)		√			√	在 IDAT 之前
sCAL	(专用公共数据块)					√	在 IDAT 之前
IDAT	**图像数据块**	√			√		与其他 IDAT 连续
tIME	图像最后修改时间数据块(Image Last-Modification Time)		√			√	无限制
tEXt	文本信息数据块(Textual Data)		√		√	√	无限制
zTXt	压缩文本数据块(Compressed Textual Data)		√		√	√	无限制
fRAc	(专用公共数据块)			√	√	√	无限制
gIFg	(专用公共数据块)			√	√	√	无限制
gIFt	(专用公共数据块)			√	√	√	无限制
gIFx	(专用公共数据块)			√	√	√	无限制
IEND	**图像结束数据块**	√					最后一个数据块

虽然 PNG 文件格有很多重要的细节,但是 NI Vision 的文件读写 VI 却封装了对这些细节的处理,使得开发人员读写 PNG 文件时可不必关心这些细节。NI Vision 处理 PNG 图像文件的读写与处理其他图像文件读写的方法类似,也是使用 IMAQ Write File 2 对 PNG 图像文件读写的实例来处理。该实例也用 Image Quality 参数指定对图像压缩的情况,它的值从 0～1000 可选,数值越大,图像质量越好,默认情况下该参数的值为 750。例如,其中 0 级压缩比最高,图像品质最差,而 1000 级则图像的细节几乎无损。此外,如果要

将 16 位有符号的图像保存到 PNG 文件中，NI Vision 必须将数据转换为无符号类型并对数据进行移位，以确保最高位始终在最左。可以通过 Use bit depth 参数选择转换时是否使用 IMAQ Image Bit Depth 指定的位深度进行像素映射。若不使用位深度，则 VI 会先对所有像素值增加一个常量，使最小负像素值变为 0，再根据结果中最大像素值进行像素移位，将所有像素转换为 16 位无符号类型值。

图 5-13 显示了使用 IMAQ ReadFile 和 IMAQ Write File 2 读写文件的例子。程序增加了检测 PNG、BMP、JPEG、JPEG 2000 和 TIFF 格式文件的功能，并能根据不同文件格式选择 IMAQ Write File 2 与该格式对应的实例保存图像文件。若所选文件为 NI Vision 不支持的图像文件格式，则程序将不做任何处理。当程序检测到选择的文件类型为 PNG 格式时，会先用 IMAQ Write Custom Data 为内存中的图像数据添加自定义文本信息，再试图将其连同图像数据保存到 PNG 文件。如果此时直接使用 IMAQ Write PNG File 2，则只有图像数据会被存放至文件，而自定义文本并不会被保存。若要将两种信息均成功存放至 PNG 文件中，必须使用 IMAQ Write Image And Vision Info File 2。

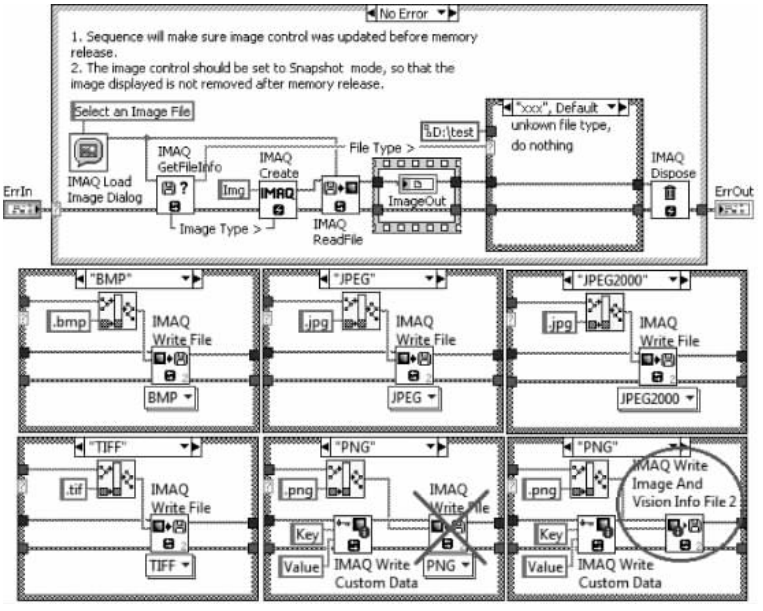

图 5-13　支持文件格式检查的文件读写

PNG 文件格式以数据块组织数据的方式赋予了它保存用户定义辅助信息的能力，这是其他几种常用图像文件格式不具备的。NI Vision 提供的 IMAQ Read Image And Vision Info 和 IMAQ Write Image And Vision Info File 2 专门用来一并读写 PNG 文件中的图像数据和机器视觉信息。它们位于 LabVIEW 的 Vision and Motion→Vision Utilities→Overlay 函数选板中。多数情况下，机器视觉信息由程序中的图像处理 VI 生成，包括系统校准、灰度或彩色图像的模式匹配模板、无损图层、几何匹配模板、黄金匹配模板以及其他与图像关联的自定义数据等，而自定义数据则由 IMAQ Write Custom Data 事先添加到内存图像中。表 5-12 汇总了与图像中机器视觉信息和自定义数据操作相关的 VI，需要注意 IMAQ Read Image And Vision Info 和 IMAQ Write Image And Vision Info File 2 两个 VI

用于图像文件的读写，而其他 VI 则基于内存中的图像数据进行操作。

表 5-12 与图像中机器视觉信息和自定义数据操作相关的 VI

分 类	属性或操作	NI Vision VI
文件读写	保存图像数据和机器视觉信息	IMAQ Read Image And Vision Info IMAQ Write Image And Vision Info File 2
内存图像	机器视觉信息	IMAQ Is Vision Info Present 2 IMAQ Remove Vision Info 2 IMAQ Read Image And Vision Info
	图像中用户自定数据	IMAQ Read Custom Data IMAQ Write Custom Data IMAQ Get Custom Keys IMAQ Remove Custom Data

5.6 AVI 文件

AVI(Audio Video Interleaved)是由微软公司在 1992 年 11 月推出的一种多媒体文件格式。它可将单个或多个视频流和音频流以交叠顺序封装在一个文件中，以达到音频与视频可同步播放的效果。AVI 文件对视频文件采用了一种压缩比较高的有损压缩方式，虽然画面质量不是很高，但其应用范围仍非常广泛。AVI 格式支持 256 色和 RLE 压缩，主要用来保存电视、电影等多媒体影像信息。对于机器视觉和图像处理分析应用来说，若对图像的质量要求不是很高，使用 AVI 文件可以将多帧图像保存在同一个文件中，方便处理。

AVI 文件本质上是一种 RIFF(Resource Interchange File Format)文件格式。RIFF 格式文件使用诸如'RIFF'、'AVI '、'LIST'等四字符码(Four-Character Code，FOURCC)来组织数据。其中，四字符码允许含有空格，如'AVI '。在 RIFF 文件中，四字符码和数据的安排顺序如表 5-13 所示。此外还应注意，Windows 操作系统使用从低到高的字节存放顺序(Little-Endian，Intel)，因此诸如 DWORD 类型的数据 0xA8B9C0D1 在文件或内存中的存储顺序为 D1 C0 B9 A8。

表 5-13 RIFF 文件结构

顺序	字段	字节数	说 明
1	RIFF	4B	文件类型四字符码
2	文件大小	4B	文件大小四字符码 等于实际数据长度加上文件类型域的大小(4B)；也就是说，文件大小的值不包括 RIFF 域和文件大小域本身的大小
3	文件类型	4B	文件大小四字符码，如'AVI '、'WAVE'等
4	实际数据	可变	实际保存的数据，常使用 List 和 Chunk 组织数据

RIFF 文件通常使用列表(List)和块(Chunk)来组织文件中的实际数据，且列表可以嵌套子列表和数据块。其中列表和数据块的结构如表 5-14 所示。

表 5-14 列表(List)的结构

域	字段	字节数	说 明
列表	**LIST**	4B	列表结构四字符码
	List Size	4B	整个列表的大小； 等于实际的列表数据长度与 listType 域的大小之和(4B)。也就是说 List Size 的值不包括 LIST 域和 List Size 域本身的大小
	List Type	4B	本列表的类型
	List Data	可变	列表中实际保存的数据,列表可以嵌套子列表和数据块
块	**ckID**	4B	块的四字符码
	ckSize	4B	除 ckID 域和 ckSize 域外数据块的大小
	ckData	可变	块中的实际数据

AVI 文件类型用一个四字符码'AVI '来表示。整个 AVI 文件由 RIFF 文件头、两个媒体流列表(一个用于描述媒体流格式,另一个用于保存媒体流数据)和一个可选的索引块 3 部分构成。AVI 文件的展开结构大致如图 5-14 所示,其中 LIST(listType(listData))形式表示列表,ckID(ckData)形式表示数据块,[Optional Element]表示可选项。

```
RIFF ('AVI '
      LIST ('hdrl'
            'avih'(主 AVI 信息头数据)
            LIST ('strl'
                  'strh'(流的头信息数据)
                  'strf'(流的格式信息数据)
                  ['strd'(可选的额外的头信息数据)]
                  ['strn'(可选的流的名字)]
                  …
                  )
              …
            )
      LIST ('movi'
            { SubChunk I LIST ('rec '
                               SubChunk1
                               SubChunk2
                               …
                              )
               …
            }
            …
           )
      ['idxl'(可选的 AVI 索引块数据)]
     )
```

图 5-14 AVI 文件的展开结构

文件头一开始用 RIFF ('AVI '…)表示 AVI 文件类型。紧跟其后的是 AVI 文件必需的第一个列表 hdrl,它用于描述 AVI 文件中各个流的格式信息(AVI 文件中的每一路媒体数据都称为一个流)。hdrl 列表嵌套了一系列块和子列表,第一个嵌套的 avih 块用于记录 AVI 文件的全局信息,如流的数量、视频图像的宽和高等。它通常可以通过下面的

AVIMAINHEADER 数据结构来操作：

```
typedef struct _avimainheader
{
    FOURCC fcc;                    // 必须为'avih'
    DWORD cb;                      // 本数据结构的大小,不包括最初的 8 个字节(fcc 和 cb 两个域)
    DWORD dwMicroSecPerFrame;      // 视频帧间隔时间(以毫秒为单位)
    DWORD dwMaxBytesPerSec;        // 这个 AVI 文件的最大数据率
    DWORD dwPaddingGranularity;    // 数据填充的粒度
    DWORD dwFlags;                 // AVI 文件的全局标记,比如是否含有索引块等
    DWORD dwTotalFrames;           // 总帧数
    DWORD dwInitialFrames;         // 为交互格式指定初始帧数(非交互格式应该指定为 0)
    DWORD dwStreams;               // 本文件包含的流的个数
    DWORD dwSuggestedBufferSize;   // 建议读取本文件的缓存大小(应能容纳最大的块)
    DWORD dwWidth;                 // 视频图像的宽(以像素为单位)
    DWORD dwHeight;                // 视频图像的高(以像素为单位)
    DWORD dwReserved [4];          // 保留
} AVIMAINHEADER;
```

在 avih 块后嵌套了一个或多个 strl 子列表，这些 strl 子列表按顺序对媒体流进行说明。也就是说第一个 strl 子列表用来说明第一个流，第二个 strl 子列表用于说明第二个流，以此类推。文件中有多少个流，这里就对应有多少个 strl 子列表。每个 strl 子列表至少包含一个 strh 块和一个 strf 块，而保存编解码器配置信息的 strd 块和保存流名字的 strn 块为可选。strl 子列表中的 strh 块包含了流的结构信息，它可以通过下面的 AVISTREAMHEADER 数据结构来操作：

```
typedef struct _avistreamheader
{
    FOURCC fcc;                    // 必须为'strh'
    DWORD cb;                      // 本数据结构的大小,不包括最初的 8 个字节(fcc 和 cb 两个域)
    FOURCC fccType;                // 流的类型: 'auds'(音频流)、'vids'(视频流)、
                                   //'mids'(MIDI 流)、'txts'(文字流)
    FOURCC fccHandler;             // 指定流的处理者,对于音视频来说就是解码器
    DWORD dwFlags;                 // 标记: 是否允许这个流输出,调色板是否变化
    WORD wPriority;                // 流的优先级(有多个相同类型的流时,优先级最高的为默认流)
    WORD wLanguage;
    DWORD dwInitialFrames;         // 为交互格式指定初始帧数
    DWORD dwScale;                 // 这个流使用的时间尺度
    DWORD dwRate;
    DWORD dwStart;                 // 流的开始时间
    DWORD dwLength;                // 流的长度(单位与 dwScale 和 dwRate 的定义有关)
    DWORD dwSuggestedBufferSize;   // 读取这个流数据建议使用的缓存大小
    DWORD dwQuality;               // 流数据的质量指标(0 ～ 10 000)
    DWORD dwSampleSize;            // Sample 的大小
    struct {
        short int left;
```

```
        short int top;
        short int right;
        short int bottom;
    } rcFrame;                    // 指定流(视频流或文字流)在视频主窗口中的显示位置
                                  // 视频主窗口由 AVIMAINHEADER 结构中的 dwWidth 和 dwHeight 决定
} AVISTREAMHEADER;
```

strl 子列表 strf 块用于说明流的具体格式。如果流数据为视频流,使用 BITMAPINFO 数据结构来描述;而如果是音频流,则使用一个 WAVEFORMATEX 数据结构来描述。当 AVI 文件中的所有流都使用一个 strl 子列表说明以后,hdrl 列表的任务也就完成了。

紧随 hdrl 列表之后的就是 AVI 文件必需的另一列表 movi,它用于保存真正的视频和音频媒体流数据。可以将数据块直接嵌在 movi 列表里面,也可以将几个数据块组合成一个 rec 列表后再编排进 movi 列表。读取 AVI 文件内容时,一般需要将一个 rec 列表中的所有数据块一次性读出。当 AVI 文件中包含多个流时,数据块使用四字符码对它们进行区别。该四字符码由 2 个字节的流编号和 2 个字节的类型码组成。类型码包括 db(非压缩视频帧)、dc(压缩视频帧)、pc(改用新的调色板)和 wb(音缩视频)4 种。例如第一个流为音频,第二个流是压缩视频,则表示音频流数据块的四字符码为 00wb,表示压缩视频数据块的四字符码为 01dc。

对于视频数据来说,在 AVI 数据序列中间还可以定义一个新的调色板,每个改变的调色板数据块用编号和类型码 pc 来表示,新的调色板使用数据结构 AVIPALCHANGE 来定义。应注意的是,若一个流的调色板中途可能改变,则应在这个流格式的描述中,也就是 AVISTREAMHEADER 结构的 dwFlags 中包含 AVISF_VIDEO_PALCHANGES 标记。另外,文字流数据块可以使用随意的类型码表示。

最后,紧跟在 hdrl 列表和 movi 列表之后的是 AVI 文件可选的索引块。索引块对文件中每一个媒体数据块进行索引,并且记录它们相对于 AVI 文件开头或相对于 movi 列表的偏移。索引块中的信息赋予了 AVI 文件灵活随机存取内数据的能力。若 AVI 文件包含索引块,则应在 AVIMAINHEADER 结构的 dwFlags 中包含 AVIF_HASINDEX 标记。索引块使用四字符码 idx1 来表征,整个块可使用数据结构 AVIOLDINDEX 来定义:

```
typedef struct _avioldindex
{
    FOURCC fcc;                   // 必须为'idx1'
    DWORD cb;                     // 本数据结构的大小,不包括最初的 8B(fcc 和 cb 两个域)
    struct _avioldindex_entry {
        DWORD    dwChunkId;       // 表征本数据块的四字符码
        DWORD    dwFlags;         // 说明本数据块是不是关键帧、是不是'rec '列表等信息
        DWORD    dwOffset;        // 本数据块在文件中的偏移
        DWORD    dwSize;          // 本数据块的大小
    } aIndex[];                   // 这是一个数组!为每个媒体数据块都定义一个索引信息
} AVIOLDINDEX;
```

此外，还有一种专门用于内部数据对齐的特殊的数据块，用一个四字符码'JUNK'来表征，应用程序应忽略这些数据块的实际意义。值得一提的是，很多 AVI 文件还支持由 Matrox OpenDML 集团于 1996 年 2 月开发的格式后缀，这些文件非正式地称为 AVI 2.0，并得到微软公司的支持。上述 AVI 文件格式的介绍，并不包括 OpenDML AVI 文件格式扩展部分的内容。

AVI 本身只是一个数据组织的框架，内部的图像数据和声音顺据格式可以任意形式编码。AVI 格式可以跨多个平台使用，但它体积往往很大，而且压缩标准不统一，解码时需要对应版本的解码器。另外，由于索引块放在了文件尾部，所以在播放流媒体时往往需要较长时间的等待。

NI Vision 同样提供了高度模块化的 AVI 文件操作 VI，如图 5-15 所示。使用这些 VI 可以创建或打开 AVI 文件、获取 AVI 文件中图像的帧数及类型信息、读写 AVI 文件中的图像帧以及关闭 AVI 文件。由于 AVI 文件格式没有统一的压缩标准，不同的编码器的速度、生成的 AVI 文件大小以及文件中图像的质量各不相同。因此读写文件时通常还需要获知系统已安装的 AVI 编码解码器(CODEC)的名字，根据需求选择相应的 CODEC，NI Vision 也提供了针对此操作的 VI。

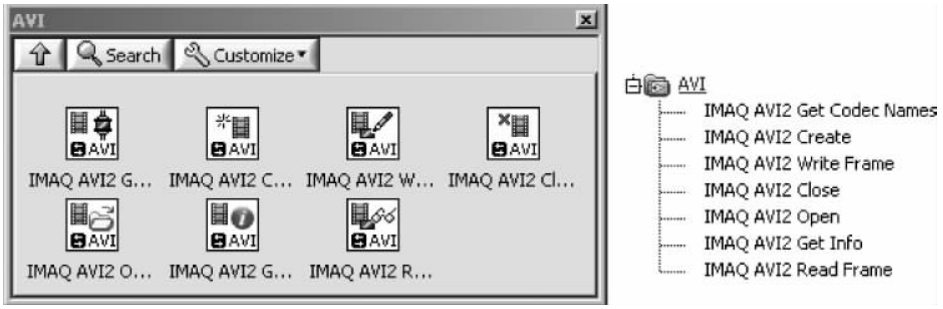

图 5-15 AVI 文件的操作 VI

有了 NI Vision 提供的 AVI 文件操作 VI，读写 AVI 文件就变得极为简单。图 5-16 展示了将一个文件夹中的 PNG 文件写入 AVI 文件的例子。程序开始时先使用 List Folder 函数列出文件夹\pcb 中的所有 PNG 文件，然后取出文件夹中的第一个 PNG 文件，通过 IMAQ GetFileInfo 获取它在 X、Y 方向上的分辨率，用于统一要写入 AVI 文件的图像大小参考。IMAQ Create 为图像的读取在内存中分配缓冲区，此后 IMAQ ReadFile 会将第一幅图像读入内存。有了第一幅图像数据作为参考，IMAQ Codec 就可以根据该图像的类型、尺寸、像素位深度等信息，枚举出系统中已安装的与图像兼容的 AVI 编码解码器名称。在此例子中，未对各种编码解码器的性能进行比较，而是直接使用枚举出的第一个 AVI 编码解码器进行 AVI 文件操作。

IMAQ AVI2 Create 根据第一个编码解码器的名称，在\pcb 目录中新创建了一个名为 new.avi 的文件。创建 AVI 文件时，可以指定帧率和文件的质量。帧率指播放 AVI 文件时每秒显示的帧数，通常与视频制式统一，如 PAL 为 25 帧/秒，NTSC 为 30 帧/秒。此处为了便于观察效果，设置帧率为 3。文件的质量指编码器对图像压缩时的相对质量，用 0～1000 范围内的数字表示，数字越大，文件中图像的质量越好。紧随其后，For 循环内的 IMAQ

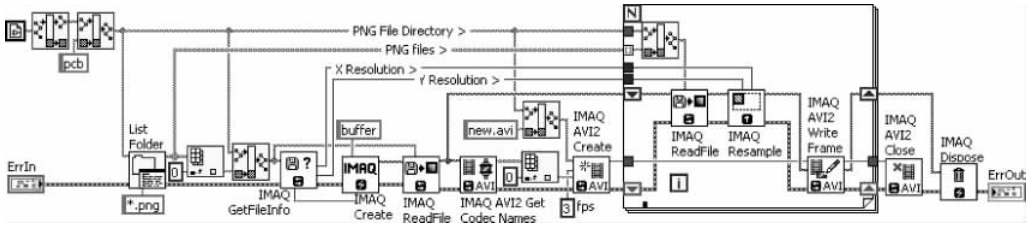

图 5-16 将多幅图像写入 AVI 文件

ReadFile 可逐一将\pcb 目录中的 PNG 文件取出，并由 IMAQ Resample 按照第一幅图像的大小对它们重新采样后写入 AVI 文件中。等全部图像都写入 AVI 文件后，程序将关闭打开的 AVI 文件，并释放分配的内存缓冲区。程序执行完成后，如果检查\pcb 目录，就会发现其中多了一个包含全部文件的 new. avi 文件。

读取 AVI 文件中图像帧的程序更为简单，如图 5-17 所示。程序中 IMAQ AVI2 Open 首先打开用户通过对话框选择的 AVI 文件，IMAQ AVI2 Get Info 立即从打开的文件中获得总帧数、帧率及图像类型信息。一旦 IMAQ Create 为图像帧的读取在内存中分配了缓冲，则 For 循环就会逐一读取 AVI 文件中的图像帧，如果图像的读取速度较每帧应播放的时间(帧率的倒数)快，则应相应等待以按照帧率进行播放。所有图像读取完成后，应关闭打开的 AVI 文件并释放缓冲区，才能退出程序。

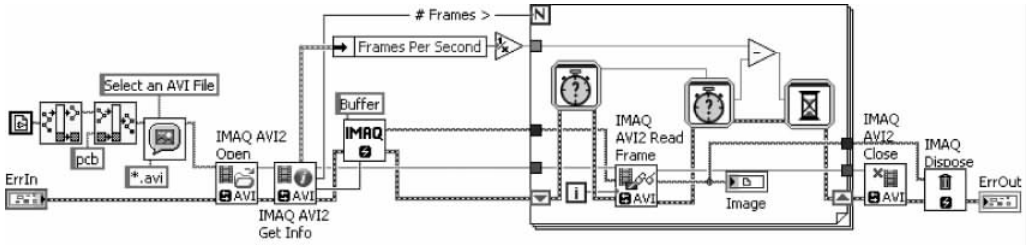

图 5-17 将多幅图像写入 AVI 文件

5.7 本章小结

图像数据可以用多种文件格式保存至存储设备，常见的标准格式有 BMP、TIFF、PNG、JPEG、JPEG 2000 等。如果需要，也可开发保存浮点数、复数或 HSL 类型图像的专用图像文件格式，或将连续多帧图像数据保存在 AVI 视频格式的文件中。不同格式的图像文件，组织图像数据的方式和保存的图像类型各不相同。而不同类型的图像，其像素位深度、色彩等特点也千差万别。

NI Vision 提供了各种类型图像文件读写的 VI，使用这些 VI 可以不必关注文件保存操作的细节，轻而易举地实现图像数据的存储。由于图像的数据量较大，因此在图像文件保存操作过程中，常会对图像数据进行压缩。根据压缩时是否损失信息，图像数据的压缩方式可被分为无损压缩和有损压缩。NI Vision 文件操作 VI 为开发人员提供了选择图像压缩质量

的参数 Image Quality，它的值从 0～1000 可选，数值越大，图像质量越好，默认情况下该参数的值为 750。例如，其中 0 级压缩比最高，图像品质最差，而 1000 级则图像的细节几乎无损。

BMP、TIFF、PNG、JPEG、JPEG 2000 和 AVI 几种图像文件格式各有优缺点，应视机器视觉系统的实际开发情况择优使用。特别地，如果要保存系统校准、灰度或彩色图像的模式匹配模板、无损图层、几何匹配模板、黄金匹配模板等机器视觉信息或其他与图像关联的自定义数据，则只能使用 PNG 文件格式。