使用 IJG 读写 JPEG 格式文件

作者: ysm 日期: 2009 年 10 月 27 日发表评论 (0)查看评论

ysm

cleverysm@gmail.com

IJG 全称为 Independent JPEG Group，是一个用于处理 JPEG 文件格式的开源库。用于遥感和地理信息系统数据处理的 GDAL 就是用这个库进行 JPEG 格式解析的。

IJG 的主页是 http://www.ijg.org/，源代码和一些相关文档可以在 http://www.ijg.org/files/下找到。
本文中将依据文档中的 libjpeg.doc 介绍一些基本的 JPEG 文件解压缩和压缩处理方式，细节及其他高级操作可以参考 libjpeg.doc 中的具体介绍。

编译

编译需要的源代码包下载链接是 http://www.ijg.org/files/jpegsrc.v6b.tar.gz，下载到硬盘上解压，比如我们将其解压到 C:\jpeg-6b，在文件夹中有几个*.doc 的文件，是 IJG 的安装使用文档，比如 install.doc 就是安装的说明书。虽然这几个文件是以 doc 为扩展名，其实只是几个文本文件，用记事本之类的程序也可以打开。Libjpeg.doc 则是在你的程序中使用 IJG 库进行开发的使用说明。另外还有个 example.c 文件是一个示例代码文件，里面有使用 IJG 进行 JPEG 读写的样本代码，以及关于每段代码的详细解释，基本上来说，在实际应用时直接按照 example.c 的代码框架进行修改就基本能满足我们应用程序的需要。
IJG 支持多平台下的应用，不同平台下的编译安装细节都可以在 install.doc 中找到。在此，我以 MS VC 2005 为例进行说明。

在 windows 下使用的时候需要对源代码做几处修改。

在头文件 jmorecfg.h 中找到如下代码：

```
#ifdef NEED_FAR_POINTERS
#define FAR  far
#else
#define FAR
#endif
```

在这段代码下面添加一句

```
#define FAR       /*ysm fixed*/
```

还是在这个文件中将

```
#ifndef XMD_H                   /* X11/xmd.h correctly defines INT32 */
typedef long INT32;
#endif
```

修改为

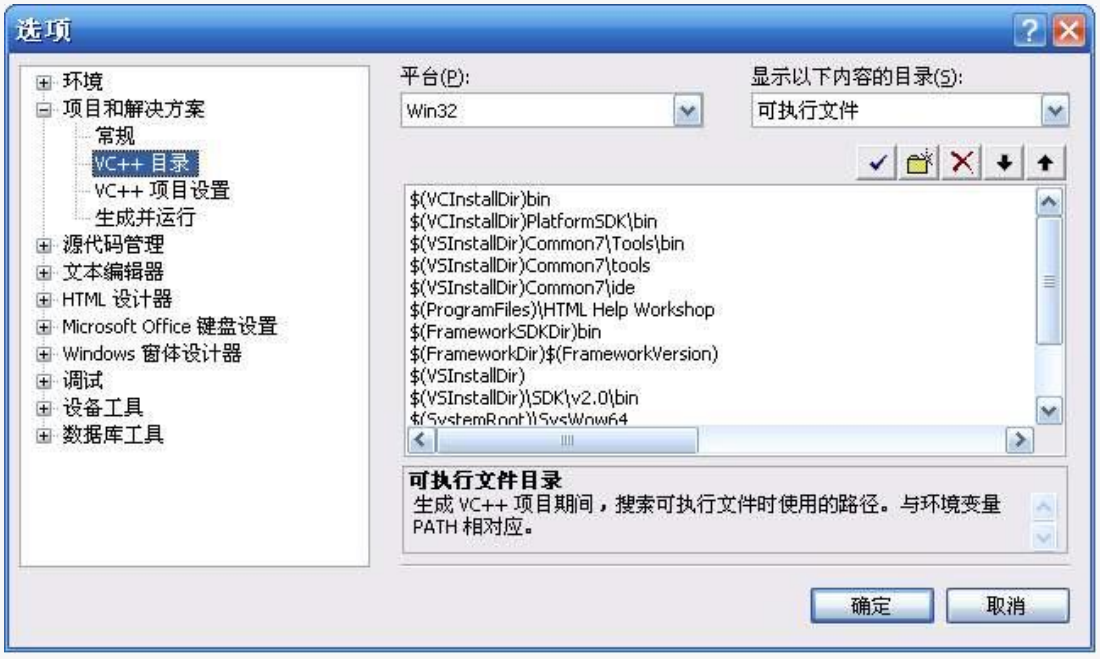
```
#ifndef XMD_H                       /* ysm fixed*/
#ifndef _BASETSD_H_
typedef long INT32;
#endif
#endif
```

开始编译前需要首先将 jconfig.vc 更名为 jconfig.h，然后在命令行下进入到 VC2005 的安装目录下，比如在我的机器上是 C:\Program Files\Microsoft Visual Studio 8\VC，在 bin 目录下有一个名为 vcvars32.bat 的批处理文件，在命令行下运行这个批处理，这个批处理的作用是注册 VC2005 相关的一些环境变量，然后不要关闭当前命令窗口，进入到 IJG 的安装目录，在此即 C:\jpeg-6b 下，执行命令 nmake /f makefile.vc，也就是开始编译 IJG 库，在屏幕迅速闪过一些编译信息后，如果没有错误出现，IJG 也就编译完成。编译过程中会可能会出现一些警告信息，大概就是说代码中使用的标准 C 函数在 VC2005 下被认为是不安全而过时的，不过没什么大关系，忽略就行，这个警告在用 VC6 编译的时候就不会出现。

编译完成后，IJG 的安装目录下会出现若干文件，我们所需要的是一个叫做 libjpeg.lib 的静态库文件，在编译我们程序的时候链接使用。除此一般还要用到 4 个头文件，jconfig.h、jerror.h、jmorecfg.h、jpeglib.h。当然，目录下还会有一些可执行文件，是 IJG 提供的工具程序，在 usage.doc 中有使用方法的介绍。但对程序员来说，最重要的还是 4 个头文件加一个库文件，以后用 IJG 开发的时候用这几个文件就足够了。

开发环境配置

在 VC2005 下使用 IJG 之前，首先要让 VC 能够找到那四个头文件，比如说将它们拷到使用 IJG 的项目当前目录下，系统的头文件目录下，或者在 VC 环境下做一下配置，也就是在菜单工具-选项-项目和解决方案-VC++目录下，将头文件和 lib 文件所在的目录分别添加到包含文件和库文件的列表中。



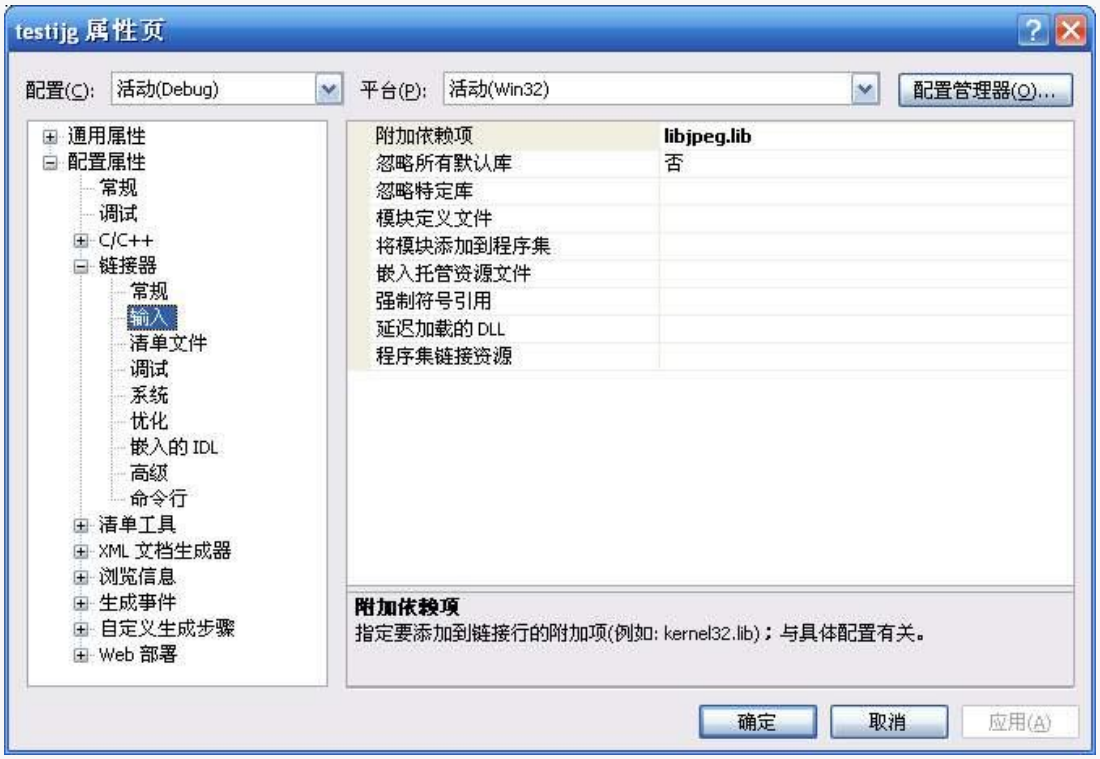
[图 1]

在程序中需要引入的头文件就是 jpeglib.h，由于 IJG 是用 C 语言写的，所以在 C++引入这个头文件就需要使用 extern "C"，如下所示：

```
extern "C"
{
#include "jpeglib.h"
}
```

此外，在 jmorecfg.h 中有段关于 INT32

而 lib 文件的引入则需要在项目的属性页下的配置属性-链接器-输入-附加依赖项中添加 libjpeg.lib。



[图 2]

至此，配置步骤完成，于是就可以开始用 IJG 写程序了。

JPEG 图像的解压缩操作

解压缩操作过程

1. 为 JPEG 对象分配空间并初始化
2. 指定解压缩数据源
3. 获取文件信息
4. 为解压缩设定参数，包括图像大小，颜色空间
5. 开始解压缩
6. 取出数据
7. 解压缩完毕
8. 释放资源

为 JPEG 对象分配空间并初始化

解压缩过程中使用的 JPEG 对象是一个 jpeg_decompress_struct 的结构体。同时还需要定义一个用于错误处理的结构体对象，IJG 中标准的错误结构体是 jpeg_error_mgr。

```
struct jpeg_decompress_struct cinfo;
struct jpeg_error_mgr jerr;
```

然后是将错误处理结构对象绑定在 JPEG 对象上。

cinfo.err = jpeg_std_error(&jerr);

这个标准的错误处理结构将使程序在出现错误时调用 exit()退出程序，如果不希望使用标准的错误处理方式，则可以通过自定义退出函数的方法自定义错误处理结构，详情见文章后面的专门章节。

初始化 cinfo 结构。

```
jpeg_create_decompress(&amp;cinfo);
```

指定解压缩数据源

利用标准 C 中的文件指针传递要打开的 jpg 文件。

```
FILE * infile;
if ((infile = fopen("sample.jpg", "rb")) == NULL)
{
        return 0;
}
jpeg_stdio_src(&amp;cinfo, infile);
```

获取文件信息

IJG 将图像的缺省信息填充到 cinfo 结构中以便程序使用。

```
(void) jpeg_read_header(&amp;cinfo, TRUE);
```

此时，常见的可用信息包括图像的宽 cinfo.image_width，高 cinfo.image_height，色彩空间 cinfo.jpeg_color_space，颜色通道数 cinfo.num_components 等。

为解压缩设定参数

在完成 jpeg_read_header 调用后，开始解压缩之前就可以进行解压缩参数的设定，也就是为 cinfo 结构的成员赋值。

比如可以设定解出来的图像的大小，也就是与原图的比例。使用 scale_num 和 scale_denom 两个参数，解出来的图像大小就是 scale_num/scale_denom，但是 IJG 当前仅支持 1/1, 1/2, 1/4, 和 1/8 这几种缩小比例。

比如要取得 1/2 原图的图像，需要如下设定：

```
cinfo.scale_num=1;
cinfo.scale_denom=2;
```

也可以设定输出图像的色彩空间，即 cinfo.out_color_space，可以把一个原本彩色的图像由真彩色 JCS_RGB 变为灰度 JCS_GRAYSCALE。如：

```
cinfo.out_color_space=JCS_GRAYSCALE;
```

开始解压缩

根据设定的解压缩参数进行图像解压缩操作。

```
(void) jpeg_start_decompress(&amp;cinfo);
```

在完成解压缩操作后，IJG 就会将解压后的图像信息填充至 cinfo 结构中。比如，输出图像宽度 cinfo.output_width，输出图像高度 cinfo.output_height，每个像素中的颜色通道数 cinfo.output_components（比如灰度为 1，全彩色为 3）等。

一般情况下，这些参数是在 jpeg_start_decompress 后才被填充到 cinfo 中的，如果希望在调用 jpeg_start_decompress 之前就获得这些参数，可以通过调用 jpeg_calc_output_dimensions()的方法来实现。

取出数据

解开的数据是按照行取出的，数据像素按照 scanline 来存储，scanline 是从左到右，从上到下的顺序，每个像素对应的各颜色或灰度通道数据是依次存储，比如一个 24-bitRGB 真彩色的图像中，一个 scanline 中的数据存储模式是 R,G,B,R,G,B,R,G,B,...，每条 scanline 是一个 JSAMPLE 类型的数组，一般来说就是 unsigned char，定义于 jmorecfg.h 中。

除了 JSAMPLE，IJG 还定义了 JSAMPROW 和 JSAMPARRAY，分别表示一行 JSAMPLE 和一个 2D 的 JSAMPLE 数组。

在此，我们定义一个 JSAMPARRAY 类型的缓冲区变量来存放图像数据。

```
JSAMPARRAY buffer;
```

然后是计算每行需要的空间大小，比如 RGB 图像就是宽度×3，灰度图就是宽度×1

```
row_stride = cinfo.output_width * cinfo.output_components;
```

为缓冲区分配空间，这里使用了 IJG 的内存管理器来完成分配。

JPOOL_IMAGE 表示分配的内存空间将在调用 jpeg_finish_compress，jpeg_finish_decompress，jpeg_abort 后被释放，而如果此参数改为 JPOOL_PERMANENT 则表示内存将一直到 JPEG 对象被销毁时才被释放。

row_stride 如上所说，是每行数据的实际大小。

最后一个参数是要分配多少行数据。此处只分配了一行。

```
buffer = (*cinfo.mem-&gt;alloc_sarray)((j_common_ptr) &amp;cinfo, JPOOL_IMAGE,
row_stride, 1);
```

output_scanline 表示当前已经读取的行数，如此即可依次读出图像的所有数据，并填充到缓冲区中，参数 1 表示的是每次读取的行数。

```
while (cinfo.output_scanline &lt; cinfo.output_height)
{
        (void) jpeg_read_scanlines(&amp;cinfo, buffer, 1);
        //do something
}
```

解压缩完毕

```
(void) jpeg_finish_decompress(&amp;cinfo);
```

释放资源

```
jpeg_destroy_decompress(&amp;cinfo);
fclose(infile);
```

退出程序

如果不再需要 JPEG 对象，则使用

```
jpeg_destroy_decompress(&amp;cinfo);
```

或

```
jpeg_destroy(&amp;cinfo);
```

而如果还希望继续使用 JPEG 对象，则可使用

```
jpeg_abort_decompress(&amp;cinfo);
```

或

```
jpeg_abort(&amp;cinfo);
```

完整例程

```
//变量定义
struct jpeg_decompress_struct cinfo;
struct jpeg_error_mgr jerr;
FILE * infile;
JSAMPARRAY buffer;
int row_stride;
//绑定标准错误处理结构
cinfo.err = jpeg_std_error(&amp;jerr);
//初始化 JPEG 对象
```

```
jpeg_create_decompress(&amp;cinfo);
//指定图像文件
if ((infile = fopen("sample.jpg", "rb")) == NULL)
{
        return;
}
jpeg_stdio_src(&amp;cinfo, infile);
//读取图像信息
(void) jpeg_read_header(&amp;cinfo, TRUE);
//设定解压缩参数，此处我们将图像长宽缩小为原图的 1/2
cinfo.scale_num=1;
cinfo.scale_denom=2;
//开始解压缩图像
(void) jpeg_start_decompress(&amp;cinfo);

//本程序功能是应用 GDI+在客户区绘制图像
CClientDC dc(this);
Bitmap bm( cinfo.output_width , cinfo.output_height);
Graphics graphics(dc.GetSafeHdc());
Graphics gdc(&amp;bm);
//分配缓冲区空间
row_stride = cinfo.output_width * cinfo.output_components;
buffer = (*cinfo.mem-&gt;alloc_sarray)((j_common_ptr) &amp;cinfo, JPOOL_IMAGE,
row_stride, 1);
//读取数据
while (cinfo.output_scanline &lt; cinfo.output_height)
{
        (void) jpeg_read_scanlines(&amp;cinfo, buffer, 1);
        //output_scanline 是从 1 开始，所以需要减 1
        int line=cinfo.output_scanline-1;
        for(int i=0;i
```

JPEG 图像的压缩操作

压缩操作过程

1. 为 JPEG 对象分配空间并初始化
2. 指定图像输出目标
3. 为压缩设定参数，包括图像大小，颜色空间
4. 开始压缩
5. 写入数据
6. 压缩完毕
7. 释放资源

为 JPEG 对象分配空间并初始化

压缩过程中使用的 JPEG 对象是一个 jpeg_compress_struct 的结构体。同时还需要定义一个用于错误处理的结构体对象，IJG 中标准的错误结构体是 jpeg_error_mgr。

```
struct jpeg_compress_struct cinfo;
struct jpeg_error_mgr jerr;
```

然后是将错误处理结构对象绑定在 JPEG 对象上。

```
cinfo.err = jpeg_std_error(&amp;jerr);
```

这个标准的错误处理结构将使程序在出现错误时调用 exit()退出程序，如果不希望使用标准的错误处理方式，则可以通过自定义退出函数的方法自定义错误处理结构，详情见文章后面的专门章节。

初始化 cinfo 结构。

```
jpeg_create_compress(&amp;cinfo);
```

指定图像输出目标

利用标准 C 中的文件指针传递要输出的 jpg 文件。

```
FILE * outfile;
if ((outfile = fopen(filename, "wb")) == NULL)
{
        return 0;
}
jpeg_stdio_dest(&amp;cinfo, outfile);
```

为压缩设定参数

在开始压缩数据之前需要为压缩指定几个参数和缺省参数。

设定缺省参数之前需要指定的几个参数是：图像宽度 cinfo.image_width，图像高度 cinfo.image_height，图像的颜色通道数 cinfo.input_components(比如 RGB 图像为 3，灰度图为 1)，图像颜色空间 cinfo.in_color_space(比如真彩色 JCS_RGB，灰度图 JCS_GRAYSCALE)。

如：

```
cinfo.image_width = 800;
cinfo.image_height = 600;
cinfo.input_components = 3;
cinfo.in_color_space = JCS_RGB;
```

然后是设定缺省设置

```
jpeg_set_defaults(&amp;cinfo);
```

注意此处，在 set default 之前，必须设定 in_color_space，因为某些缺省参数的设定需要正确的 color space 值。

在此之后还可以对其他的一些参数进行设定。具体有哪些参数可以查询 libjpeg.doc 文档。

比如最常用的一个参数就是压缩比。

```
jpeg_set_quality(&amp;cinfo, quality, TRUE);
```

quality 是个 0～100 之间的整数，表示压缩比率。

开始压缩

根据设定的压缩参数进行图像压缩操作。

```
jpeg_start_compress(&amp;cinfo, TRUE);
```

开始压缩过程后就不可以修改 cinfo 对象参数。

写入数据

```
row_stride = image_width * 3;    //假设用到的图示 RGB 真彩色三通道
```

同上文介绍的解压缩操作中介绍的，要写入的数据是按照行写入的，数据像素按照 scanline 来存储，与读取数据的不同是使用 jpeg_write_scanlines。

类似于解压缩操作中的 cinfo.output_scanline < cinfo.output_height 机制，压缩过程使用的 cinfo.next_scanline < cinfo.image_height 来判断是否完成写入数据。

在此，假设 image_buffer 是个 JSAMPARRAY 类型变量，其中保存的是要输出的图像数据，比如可以是用上文中的解压缩操作从某 JPEG 文件中获得的数据。

```
JSAMPROW row_pointer;
while (cinfo.next_scanline &lt; cinfo.image_height)
{
        //找到图像中的某一行，写入目标文件
        row_pointer = image_buffer[cinfo.next_scanline];
        (void) jpeg_write_scanlines(&amp;cinfo, &amp;row_pointer, 1);
}
```

压缩完毕

```
jpeg_finish_compress(&amp;cinfo);
```

释放资源

```
fclose(outfile);
jpeg_destroy_compress(&amp;cinfo);
```

退出程序

如果不再需要 JPEG 对象，则使用

```
jpeg_destroy_compress(&amp;cinfo);
```

或

```
jpeg_destroy(&amp;cinfo);
```

而如果还希望继续使用 JPEG 对象，则可使用

```
jpeg_abort_compress(&amp;cinfo);
```

或

```
jpeg_abort(&amp;cinfo);
```

完整例程

```
//变量定义
struct jpeg_compress_struct cinfo;
struct jpeg_error_mgr jerr;
FILE * outfile;
JSAMPROW row_pointer;
int row_stride;
//绑定标准错误处理结构
cinfo.err = jpeg_std_error(&amp;jerr);
//初始化 JPEG 对象
jpeg_create_compress(&amp;cinfo);
//指定目标图像文件
if ((outfile = fopen("dest.jpg", "wb")) == NULL)
{
        return;
}
jpeg_stdio_dest(&amp;cinfo, outfile);
//设定压缩参数
cinfo.image_width = image_width;
cinfo.image_height = image_height;
cinfo.input_components = 3;
cinfo.in_color_space = JCS_RGB;
jpeg_set_defaults(&amp;cinfo);
//此处设压缩比为 90%
jpeg_set_quality(&amp;cinfo, 90, TRUE);
//开始压缩
jpeg_start_compress(&amp;cinfo, TRUE);
//假设使用的是 RGB 图像
row_stride = image_width * 3;
//写入数据
while (cinfo.next_scanline &lt; cinfo.image_height)
{
        row_pointer = image_buffer[cinfo.next_scanline];
```

```
        (void) jpeg_write_scanlines(&amp;cinfo, &amp;row_pointer, 1);
}
//压缩完毕
jpeg_finish_compress(&amp;cinfo);
//释放资源
fclose(outfile);
jpeg_destroy_compress(&amp;cinfo);
```

错误处理

在使用默认错误处理结构 jpeg_error_mgr 的情况下，程序在遇到错误后将调用 exit 直接退出程序，用户如果不希望使用这种直接退出的方式处理错误的话就需要自定义错误处理结构。

依照 example.c 中的例子，IJG 推荐使用 C 语言的 setjmp 和 longjmp 机制来重写错误处理结构。

首先，需要定义一个包含标准错误处理结构类型变量的自定义结构。

同时，程序将需要引入头文件 setjmp.h。

```
#include
struct my_error_mgr
{
        struct jpeg_error_mgr pub;
        jmp_buf setjmp_buffer;
};
typedef struct my_error_mgr * my_error_ptr;
```

以及一个错误处理函数。在出现错误时程序将跳转到本函数中，而本函数将跳转到 setjmp 设定的程序位置。

```
METHODDEF(void) my_error_exit (j_common_ptr cinfo)
{
        my_error_ptr myerr = (my_error_ptr) cinfo->err;
        (*cinfo->err->output_message) (cinfo);
        longjmp(myerr->setjmp_buffer, 1);
}
```

以解压缩过程为例，原程序将被修改为如下形式。

```
struct jpeg_decompress_struct cinfo;
struct jpeg_error_mgr jerr;
//此处做了修改
//struct jpeg_error_mgr jerr;
struct my_error_mgr jerr;
////////////////////////////////////////////////
FILE * infile;
JSAMPARRAY buffer;
```

```
int row_stride;
//此处做了修改
//cinfo.err = jpeg_std_error(&amp;jerr);
cinfo.err = jpeg_std_error(&amp;jerr.pub);
jerr.pub.error_exit = my_error_exit;
if (setjmp(jerr.setjmp_buffer))
{
        //在正常情况下，setjmp 将返回 0，而如果程序出现错误，即调用 my_error_exit
        //然后程序将再次跳转于此，同时 setjmp 将返回在 my_error_exit 中由 longjmp 第二个
参数设定的值 1
        //并执行以下代码
        jpeg_destroy_decompress(&amp;cinfo);
        fclose(infile);
        return;
}
//////////////////////////////////////////////////
jpeg_create_decompress(&amp;cinfo);
if ((infile = fopen("sample.jpg", "rb")) == NULL)
{
        return;
}
//以下的代码与上文解压缩操作章节中相同，不再赘述
```