[**图像解码之一——使用libjpeg解码jpeg图片**](http://www.cnblogs.com/xiaoxiaoboke/archive/2012/02/13/2349763.html)

多媒体应用在现在电子产品中的地位越来越重要，尤其是在嵌入式设备中。本系列文章将会介绍利用libjpeg解码jpeg文件，[libpng解码png文件](http://my.unix-center.net/~Simon_fu/?p=1030)，libgif解码gif文件。本文为第一篇，介绍使用libjpeg解码jpeg文件。

**libjpeg简介**

    libjpeg是一个完全用[C语言](http://zh.wikipedia.org/wiki/C%E8%AF%AD%E8%A8%80)编写的[库](http://zh.wikipedia.org/wiki/%E5%BA%93)，包含了被广泛使用的[JPEG](http://zh.wikipedia.org/wiki/JPEG)解码、[JPEG](http://zh.wikipedia.org/wiki/JPEG)编码和其他的[JPEG](http://zh.wikipedia.org/wiki/JPEG)功能的实现。这个库由独立JPEG工作组维护。最新版本号是6b，于[1998](http://zh.wikipedia.org/wiki/1998)年发布。可以参考[维基百科关于libjpeg的介绍](http://zh.wikipedia.org/wiki/Libjpeg)。

**libjpeg库的数据结构**

    用libjpeg库解码jpeg数据的时候，最重要的数据类型为struct jpeg\_decompress\_struct，一般变量定义成cinfo变量，该变量保存着jpeg数据的详细信息，也保存着解码之后输出数据的详细信 息。一般情况下，每次调用libjpeg库API的时候都需要把这个变量作为第一个参数传入。另外用户也可以通过修改该变量来修改libjpeg行为，比 如输出数据格式，libjpeg库可用的最大内存等等。

**libjpeg库的使用**

**1、设置出错处理函数**

    “天有不测风云”，我们使用libjpeg库的时候难免会产生错误，所以我们在使用libjpeg解码之前，首先要做好错误处理。在libjpeg库中， 实现了默认错误处理函数，当错误发生时，比如如果内存不足（非常可能发生，后面会介绍）等，则默认错误处理函数将会调用exit函数结束整个进程，详细内 容可以参考jerror.c文件。这个对于很多用户来说这样的特性是不合适的，不过libjpeg提供了接口让我们注册自定义错误处理函数。

    在C语言中没有C++的异常处理机制，但是提供了setjmp和longjmp机制来实现类似的功能，如果你对这个机制不熟悉的话，请查阅C语言手册。本文下面的代码片段都是出自libjpeg的example.c文件，可以查阅之。

1: /\* We set up the normal JPEG error routines, then override error\_exit. \*/

2: cinfo.err = jpeg\_std\_error(&jerr.pub);

3: jerr.pub.error\_exit = my\_error\_exit;

4: /\* Establish the setjmp return context for my\_error\_exit to use. \*/

5: if (setjmp(jerr.setjmp\_buffer)) {

6: /\* If we get here, the JPEG code has signaled an error.

7: \* We need to clean up the JPEG object, close the input file, and return.

8: \*/

9: jpeg\_destroy\_decompress(&cinfo);

10: fclose(infile);

11: return 0;

12: }

    上述代码中的重点在于我们向libjpeg注册了一个my\_error\_exit回调函数，当发生错误的时候，该回调函数将会被调用。然后我们调用 setjmp函数，设置一个返回点。这样我们在my\_error\_exit回调函数处理完错误信息之后，就可以调用longjmp函数返回到这里，在这个 返回点进行资源的释放（非常重要，否则将会内存泄漏）。我们再看看my\_error\_exit回调函数的实现：

1: /\*

2: \* Here's the routine that will replace the standard error\_exit method:

3: \*/

4: METHODDEF(void)

5: my\_error\_exit (j\_common\_ptr cinfo)

6: {

7: /\* cinfo->err really points to a my\_error\_mgr struct, so coerce pointer \*/

8: my\_error\_ptr myerr = (my\_error\_ptr) cinfo->err;

9:

10: /\* Always display the message. \*/

11: /\* We could postpone this until after returning, if we chose. \*/

12: (\*cinfo->err->output\_message) (cinfo);

13:

14: /\* Return control to the setjmp point \*/

15: longjmp(myerr->setjmp\_buffer, 1);

16: }

    可以通过检查cinfo->err->msg\_code的值来判断错误类型，进行相应的处理。本例中只是简单的打印一个错误信息。最后调用longjmp跳转到setjmp调用的地方。

**2、初始化解码对象**

    要使用libjpeg解码jpeg数据，这步是必须要做的。

1: /\* Now we can initialize the JPEG decompression object. \*/

2: jpeg\_create\_decompress(&cinfo);

    这步之后，如果结束解码或者出错之后，需要调用jpeg\_destroy\_decompress销毁解码对象，否则将会内存泄漏。

**3、初始化源数据**

    在libjpeg库中仅仅提供了文件作为输入数据的接口，在example.c中代码如下：

1: /\* Step 2: specify data source (eg, a file) \*/

2: jpeg\_stdio\_src(&cinfo, infile);

    这个设计我个人觉得非常不合理，我觉得一个友好的库，需要能够接受各式各样的输入（内存数据，网络数据等等）。比较友好的做法是提供几种常用的输入数据支 持（在libjpeg中如：文件输入等）。然后还要提供一个用户注册自定义输入函数（回调函数）的接口，这样库就可以适配现实生活中各式各样的输入数据类 型了。Simon也在以前的博文中写过怎样修改libjpeg库，使之能够解码内存buffer中的jpeg数据，请参考《[LibJpeg解码内存中的Jpeg数据](http://my.unix-center.net/~Simon_fu/?p=565)》。当然Simon没有扩展libjpeg库让其支持用户注册自定义输入函数（回调函数），有兴趣的朋友可以自行实现。

**4、读取jpeg文件的头信息**

    这个和初始化解码对象一样，是必须要调用的，是约定，没什么好说的。

1: /\* Step 3: read file parameters with jpeg\_read\_header() \*/

2: (void) jpeg\_read\_header(&cinfo, TRUE);

**5、设置解码参数**

    很多情况下，这步非常重要。比如设置输出格式，设置scale（缩放）等等功能都是在这一步设置。参数设置通过修改上步得到cinfo的值来实现。这里简单介绍一下一些常用的字段。

**out\_color\_space**：输出的颜色格式，libjpeg定义如下：

1: /\* Known color spaces. \*/

2: typedef enum {

3: JCS\_UNKNOWN, /\* error/unspecified \*/

4: JCS\_GRAYSCALE, /\* monochrome \*/

5: JCS\_RGB, /\* red/green/blue \*/

6: JCS\_YCbCr, /\* Y/Cb/Cr (also known as YUV) \*/

7: JCS\_CMYK, /\* C/M/Y/K \*/

8: JCS\_YCCK, /\* Y/Cb/Cr/K \*/

9: #ifdef ANDROID\_RGB

10: JCS\_RGBA\_8888, /\* red/green/blue/alpha \*/

11: JCS\_RGB\_565 /\* red/green/blue in 565 format \*/

12: #endif

13: } J\_COLOR\_SPACE;

我们可以看出谷歌在Android扩展了几种输出格式，那么如果你需要的颜色格式输出格式libjpeg不支持（比 如：YUYV等颜色格式），那么请参考Android对libjpeg的扩展自行修改，不用担心复杂性，实现起来比较easy。请重点研究 jdcolor.c文件中的jinit\_color\_deconverter函数。

**scale\_num，scale\_denom**：因为实际的显示设备千变万化，我们可能需要根据实际情况对输出数据进行一些缩放才能够显示。libjpeg支持对输出数据进行缩放（scale），这个变量就是用来设置缩放的参数。目前libjpeg支持1/2，1/4，1/8三种缩放。

**mem**：可以指定内存管理相关的内容，比如分配和释放内存，指定libjpeg 可以使用的最大内存。默认情况下不同的平台下面都有一个libjpeg默认最大可用内存值，比如Android平台上面该值为 10000000L（10M），请参考jmemxxxx.c文件中的DEFAULT\_MAX\_MEM，了解不同平台的默认最大内存值。通过修改 mem->pub.max\_memory\_to\_use的值，库的使用者可以自定义libjpeg可以使用的最大内存值。

**6、开始解码**

    经过前面的参数设置，我们可以开始解码了，没有什么好说的。

1: /\* Step 5: Start decompressor \*/

2: (void) jpeg\_start\_decompress(&cinfo);

**7、读取解码数据**

1: /\* Here we use the library's state variable cinfo.output\_scanline as the

2: \* loop counter, so that we don't have to keep track ourselves.

3: \*/

4: while (cinfo.output\_scanline < cinfo.output\_height) {

5: /\* jpeg\_read\_scanlines expects an array of pointers to scanlines.

6: \* Here the array is only one element long, but you could ask for

7: \* more than one scanline at a time if that's more convenient.

8: \*/

9: (void) jpeg\_read\_scanlines(&cinfo, buffer, 1);

10: /\* Assume put\_scanline\_someplace wants a pointer and sample count. \*/

11: put\_scanline\_someplace(buffer[0], row\_stride);

12: }

    请注意虽然函数jpeg\_read\_scanlines可以指定一次读多少行，但是目前该函数还是只能支持一次只读1行。

**8、结束解码**

1: /\* Step 7: Finish decompression \*/

2: (void) jpeg\_finish\_decompress(&cinfo);

**9、释放解码对象**

1: /\* Step 8: Release JPEG decompression object \*/

2:

3: /\* This is an important step since it will release a good deal of memory. \*/

4: jpeg\_destroy\_decompress(&cinfo);

   至此一个jpeg数据已经解析完成了。虽然步骤不少但是对于常规的使用还是比较简单的。

**总结**

    libjpeg对于baseline的jpeg数据解码比较好，但是解码progressive的jpeg数据的时候，对内存的需求比较大（我测试过的 progressive的图片曾经发现过消耗70M内存）。如果你的硬件能够有硬件解码jpeg的能力，请尽可能使用硬件解码jpeg数据。