[**图像解码之三——giflib解码gif图片**](http://www.cnblogs.com/xiaoxiaoboke/archive/2012/02/13/2349770.html)

前面已经介绍过了[libjpeg解码jpeg图](http://my.unix-center.net/~Simon_fu/?p=1026)片和[libpng解码png图片](http://my.unix-center.net/~Simon_fu/?p=1030)，本文将会介绍怎样用giflib解码gif图片。giflib可以在[这里下载](http://sourceforge.net/projects/giflib/)。

**gif文件格式简单介绍**

    在解码jpeg图片和png图片的时候我们不需要对jpeg和png文件格式有了解就可以解码了（了解jpeg和png当然更好），但是在使用giflib解码gif的时候，我们必须要对gif文件有很简单的了解。

    gif文件中可以存放一帧或者多帧图像数据，并且可以存放图像控制信息，因此可以存储动画图片。

    gif文件由文件头开头，文件尾结尾，中间是一些连续的数据块（block）。这些数据块又分为图像数据块和扩展数据块（extension），图像数据块可以理解成存放一帧的图像数据。扩展数据块存放的是一些辅助信息，比如指示怎样显示图像数据等等。

    gif文件中的图像基于调色板的，因此一张gif文件中的图像最多只能有255中颜色，因此gif文件只能存储比较简单的图像。gif文件中有两种调色板 ——全局调色板和图像局部调色板。当一帧图像有局部调色板时，则以局部调色板来解码该帧图像，如果该帧图像没有局部调色板则用全局调色板来解码该图像。

    更详细的信息可以查阅giflib的文档中的gif89.txt文件，或者在网络搜索相关的信息。

**giflib中数据类型**

    在giflib中最重要的数据类型为GifFileType，定义如下：

1: typedef struct GifFileType {

2: int SWidth, SHeight, /\* Screen dimensions. \*/

3: SColorResolution, /\* How many colors can we generate? \*/

4: SBackGroundColor; /\* I hope you understand this one... \*/

5: ColorMapObject \*SColorMap; /\* NULL if not exists. \*/

6: int ImageCount; /\* Number of current image \*/

7: GifImageDesc Image; /\* Block describing current image \*/

8: struct SavedImage \*SavedImages; /\* Use this to accumulate file state \*/

9: VoidPtr Private; /\* The regular user should not mess with this one! \*/

10: } GifFileType

    以上代码来自gif\_lib.h文件，使用giflib库解码gif文件都需要包含这个文件。

1、以S开头的变量标识屏幕（Screen）。SaveImages变量用来存储已经读取过得图像数据。

2、Private变量用来保存giflib私有数据，用户不应该访问该变量。

3、其他变量都标识当前图像。

**初始化giflib**

    初四化giflib比较简单，只需要打开相应的gif数据就可以了。giflib和libpng一样提供了两种打开源数据的方式，一种是以文件流方式打开gif文件，另外一种用户可以自定义输入回调函数给giflib来完成初始化。

    文件流初始化giflib的代码如下：

1: if ((GifFile = DGifOpenFileName(\*FileName)) == NULL) {

2: PrintGifError();

3: exit(EXIT\_FAILURE);

4: }

    也可以以文件句柄方式初始化giflib，例如：

1: if ((GifFile = DGifOpenFileHandle(0)) == NULL) {

2: PrintGifError();

3: exit(EXIT\_FAILURE);

4: }

    重点介绍一下怎样用自定义输入回调函数来初始化giflib，因为这个可以适配各式各样的数据输入方式比如网络等，参考代码如下：

1: if ((GifFile = DGifOpen(&gif, gif\_input\_cb)) == NULL) {

2: PrintGifError();

3: exit(EXIT\_FAILURE);

4: }

    gif\_input\_cb为自定义输入回调函数，该函数负责giflib的数据输入。

    我们另外需要注意以上三个函数都返回一个GifFileType类型的指针，该指针以后在调用giflib的函数时，用作第一个参数传入。

**初始化屏幕**

    所有的gif图像共享一个屏幕（Screen），这个屏幕和我们的电脑屏幕不同，只是一个逻辑概念。所有的图像都会绘制到屏幕上面。

    首先我们需要给屏幕分配内存：

1: if ((ScreenBuffer = (GifRowType \*)

2: malloc(GifFile->SHeight \* sizeof(GifRowType \*))) == NULL)

3: GIF\_EXIT("Failed to allocate memory required, aborted.");

    另外我们需要以背景颜色（GifFile->SBackGroundColor）初始化屏幕buffer。

1: Size = GifFile->SWidth \* sizeof(GifPixelType);/\* Size in bytes one row.\*/

2: if ((ScreenBuffer[0] = (GifRowType) malloc(Size)) == NULL) /\* First row. \*/

3: GIF\_EXIT("Failed to allocate memory required, aborted.");

4:

5: for (i = 0; i < GifFile->SWidth; i++) /\* Set its color to BackGround. \*/

6: ScreenBuffer[0][i] = GifFile->SBackGroundColor;

7: for (i = 1; i < GifFile->SHeight; i++) {

8: /\* Allocate the other rows, and set their color to background too: \*/

9: if ((ScreenBuffer[i] = (GifRowType) malloc(Size)) == NULL)

10: GIF\_EXIT("Failed to allocate memory required, aborted.");

11:

12: memcpy(ScreenBuffer[i], ScreenBuffer[0], Size);

13: }

**解码gif数据**

    我们上面已经提到gif数据是以顺序存放的块来存储的，DGifGetRecordType函数用来获取下一块数据的类型。因此解码gif数据的代码组织如下：

1: do {

2: if (DGifGetRecordType(GifFile, &RecordType) == GIF\_ERROR) {

3: PrintGifError();

4: exit(EXIT\_FAILURE);

5: }

6:

7: switch (RecordType) {

8: case IMAGE\_DESC\_RECORD\_TYPE:

9: break;

10: case EXTENSION\_RECORD\_TYPE:

11: break;

12: case TERMINATE\_RECORD\_TYPE:

13: break;

14: default: /\* Should be traps by DGifGetRecordType. \*/

15: break;

16: }

17: } while (RecordType != TERMINATE\_RECORD\_TYPE);

    循环解析gif数据，并根据不同的类型进行不同的处理。

处理图像数据

    首先先介绍怎样处理IMAGE\_DESC\_RECORD\_TYPE类型的数据。这代表这是一个图像数据块，这个图像数据需要绘制到前面提到的屏幕buffer上面，相应的代码如下：

1: if (DGifGetImageDesc(GifFile) == GIF\_ERROR) {

2: PrintGifError();

3: exit(EXIT\_FAILURE);

4: }

5: Row = GifFile->Image.Top; /\* Image Position relative to Screen. \*/

6: Col = GifFile->Image.Left;

7: Width = GifFile->Image.Width;

8: Height = GifFile->Image.Height;

9: GifQprintf("\n%s: Image %d at (%d, %d) [%dx%d]: ",

10: PROGRAM\_NAME, ++ImageNum, Col, Row, Width, Height);

11: if (GifFile->Image.Left + GifFile->Image.Width > GifFile->SWidth ||

12: GifFile->Image.Top + GifFile->Image.Height > GifFile->SHeight) {

13: fprintf(stderr, "Image %d is not confined to screen dimension, aborted.\n",ImageNum);

14: exit(EXIT\_FAILURE);

15: }

16: if (GifFile->Image.Interlace) {

17: /\* Need to perform 4 passes on the images: \*/

18: for (Count = i = 0; i < 4; i++)

19: for (j = Row + InterlacedOffset[i]; j < Row + Height;

20: j += InterlacedJumps[i]) {

21: GifQprintf("\b\b\b\b%-4d", Count++);

22: if (DGifGetLine(GifFile, &ScreenBuffer[j][Col], Width) == GIF\_ERROR) {

23: PrintGifError();

24: exit(EXIT\_FAILURE);

25: }

26: }

27: }

28: else {

29: for (i = 0; i < Height; i++) {

30: GifQprintf("\b\b\b\b%-4d", i);

31: if (DGifGetLine(GifFile, &ScreenBuffer[Row++][Col],

32: Width) == GIF\_ERROR) {

33: PrintGifError();

34: exit(EXIT\_FAILURE);

35: }

36: }

37: }

38:

39: /\* Get the color map \*/

40: ColorMap = (GifFile->Image.ColorMap

41: ? GifFile->Image.ColorMap

42: : GifFile->SColorMap);

43: if (ColorMap == NULL) {

44: fprintf(stderr, "Gif Image does not have a colormap\n");

45: exit(EXIT\_FAILURE);

46: }

47:

48: DumpScreen2RGB(OutFileName, OneFileFlag,

49: ScreenBuffer, GifFile->SWidth, GifFile->SHeight);

    这里面有几点需要注意的是：

1、gif数据交织处理，参照上面的代码。

2、另外注意调色板的选择，如果当前图像数据中有局部调色板就用局部调色板来解码数据，否则用全局调色板来解码数据。

3、屏幕数据的解码，根据你的显示要求选择输出格式。

    解析屏幕数据，假设需要把数据转换成ARGB8888的格式，代码如下：

1: static void DumpScreen2RGBA(UINT8\* grb\_buffer, GifRowType \*ScreenBuffer, int ScreenWidth, int ScreenHeight)

2: {

3: int i, j;

4: GifRowType GifRow;

5: static GifColorType \*ColorMapEntry;

6: unsigned char \*BufferP;

7:

8: for (i = 0; i < ScreenHeight; i++) {

9: GifRow = ScreenBuffer[i];

10: BufferP = grb\_buffer + i \* (ScreenWidth \* 4);

11: for (j = 0; j < ScreenWidth; j++) {

12: ColorMapEntry = &ColorMap->Colors[GifRow[j]];

13: \*BufferP++ = ColorMapEntry->Blue;

14: \*BufferP++ = ColorMapEntry->Green;

15: \*BufferP++ = ColorMapEntry->Red;

16: \*BufferP++ = 0xff;

17: }

18: }

19: }

    解析屏幕数据的时候需要住处理透明色。这里先埋一个伏笔，等介绍完扩展块，再来重新实现这个函数。

处理扩展块

    上面提到扩展块主要实现一些辅助功能，扩展块影响其后的图像数据解码。这里面比较重要的扩展块是图像控制扩展块（Graphic control extension）。可以参考giflib文档中的gif89.txt文件了解图像控制扩展块的详细内容。这个扩展块中有两个内容我们比较关心：

**延时时间（delay time）**——后面的图像延时多长时间再显示，如果解码线程不是主线程的话，可以在这里延时一下再处理后面的数据。

**透明色（transparent color）**——在后面的图像解码时，遇到同样的颜色值，则跳过不解码，继续处理后续的点。

    如下是一种可供参考的处理方式：

1: UINT32 delay = 0;

2: if( ExtCode == GIF\_CONTROL\_EXT\_CODE

3: && Extension[0] == GIF\_CONTROL\_EXT\_SIZE) {

4: delay = (Extension[3] << 8 | Extension[2]) \* 10;

5: /\* Can sleep here \*/

6: }

7:

8: /\* handle transparent color \*/

9: if( (Extension[1] & 1) == 1 ) {

10: trans\_color = Extension[4];

11: }

12: else

13: trans\_color = -1;

    这里GIF\_CONTROL\_EXT\_CODE为0xF9表明该扩展块是一个图像控制扩展块，GIF\_CONTROL\_EXT\_SIZE为4，图像控制扩 展块的大小。我们可以看到解析出delay信息之后，就地delay。解析出透明颜色值之后，则标识透明色，否则标识为-1。解析图片的时候可以根据透明 色的值进行相应的处理，参考如下解析图像的函数：

1: static void DumpScreen2RGBA(UINT8\* grb\_buffer, GifRowType \*ScreenBuffer, int ScreenWidth, int ScreenHeight)

2: {

3: int i, j;

4: GifRowType GifRow;

5: static GifColorType \*ColorMapEntry;

6: unsigned char \*BufferP;

7:

8: for (i = 0; i < ScreenHeight; i++) {

9: GifRow = ScreenBuffer[i];

10: BufferP = grb\_buffer + i \* (ScreenWidth \* 4);

11: for (j = 0; j < ScreenWidth; j++) {

12: if( trans\_color != -1 && trans\_color == GifRow[j] ) {

13: BufferP += 4;

14: continue;

15: }

16:

17: ColorMapEntry = &ColorMap->Colors[GifRow[j]];

18: \*BufferP++ = ColorMapEntry->Blue;

19: \*BufferP++ = ColorMapEntry->Green;

20: \*BufferP++ = ColorMapEntry->Red;

21: \*BufferP++ = 0xff;

22: }

23: }

24: }

    注意我们这里假设grb\_buffer已经正确的初始化，如果他是垃圾数据，那么得到结果肯定是错误的。

    至此gif文件解析完成了，gif图片和gif动画都可以正确的解析，并显示了。

**总结**

    从上面的情况我们看出如果想使用giflib还是需要对gif格式有一个简单的了解，这点要求比libjpeg和libpng要求要高了一些。至此图像处理系列已经完成，欢迎大家批评指正。