1. **简单图像处理**

[2.1 BMP图像处理 2](#_Toc1110)

[2.1.1 BMP文件格式解析 2](#_Toc26238)

[2.1.2 代码实现：将BMP文件解析为RGB格式，在LCD上显示 6](#_Toc20316)

[2.2 JPEG图像处理 9](#_Toc6061)

[2.2.1 JPEG文件格式和libjpeg编译 9](#_Toc25391)

[2.2.2 libjpeg接口函数的解析和使用 10](#_Toc19390)

[2.2.3 使用libjpeg把JPEG文件解析为RGB格式，在LCD上显示 11](#_Toc18302)

[2.3 PNG图像处理 13](#_Toc12261)

[2.3.1 PNG文件格式和libpng编译 13](#_Toc12270)

[2.3.2 libpng接口函数的解析和使用 14](#_Toc24451)

[2.3.3 使用libpng把png文件转为rgb格式，在LCD上显示 15](#_Toc28517)

[2.4 图像调整 18](#_Toc6859)

[2.4.1 图像的缩放 18](#_Toc8867)

[2.4.1.1 图像缩放算法浅析 18](#_Toc23410)

[2.4.1.2源码编写：图像缩放算法 18](#_Toc13492)

[2.4.2 图像的旋转 20](#_Toc32719)

[2.4.2.1 图像旋转算法浅析 20](#_Toc13322)

[2.4.2.2 源码编写：图像旋转算法 21](#_Toc18464)

前言：所有的图像文件，都是一种二进制格式文件，每一个图像文件，都可以通过解析文件中的每一组二进制数的含义来获得文件中的各种信息，如图像高度，宽度，像素位数等等。只是不同的文件格式所代表的二进制数含义不一样罢了。我们可以通过UltraEdit软件打开图像文件并查看里面的二进制数排列。

# 2.1 BMP图像处理

## 2.1.1 BMP文件格式解析

BMP是一种常见的图像格式，BMP文件可看成由4个部分组成：位图文件头(bitmap-file header)、位图信息头(bitmap-information header)、调色板(color palette)和定义位图的字节阵列。以最简单的24位真彩色BMP文件作例子讲解：

1. 位图文件头(bitmap-file header)

这部分可以理解为是一个结构体，里面的每一个成员都表示一个属性

位数文件头由以下信息组成：

|  |  |  |
| --- | --- | --- |
| 名称 | 字节数 | 含义 |
| bfType | 2字节 | 表明它是BMP格式的文件，内容固定为0x42,0x4D，即ASCII字符中的“B”“M” |
| bfSize | 4字节 | BMP文件的大小，单位为字节 |
| bfReserved1 | 2字节 | 保留 |
| bfReserved2 | 2字节 | 保留 |
| bfOffBits | 4字节 | 位图文件头+位图信息头+调色板 |

我们用UltraEdit打开一个BMP文件，可以看到如下信息

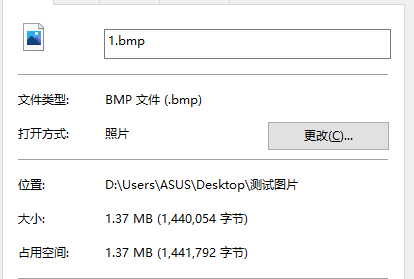


这是该BMP文件前32字节的数据，可以看到，前两个字节分别为0x42,0x4D；

接着后面4个字节依次是0x36,0xF9,0x15,0x00。

在BMP格式中，文件的存储方式是小端模式，即如果一个数据需要用几个字节来表示的话，那么，低位数据存在低位地址上，高位数据存在高位地址上。类似的，还有大端模式，即：如果一个数据需要用几个字节来表示的话，那么，低位数据存在高位地址上，高位数据存在低位地址上。

所以0x36,0xF9,0x15,0x00四个数据拼接方法应该是：0x0015F936(在数字中个位即最右边才是最低位)，它正好就是这个文件的大小：



紧接着是4个保留位字节，其数据必须为0x00。

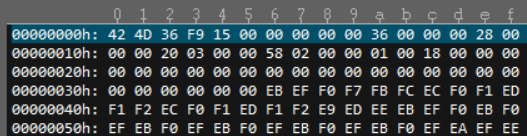
最后是4个字节的便宜位，可以看到位图文件头+位图信息头+调色板的大小应该是0x36。

1. 位图信息头(bitmap-information header)

位图信息头也可以理解为是一个结构体，其成员有：

|  |  |  |
| --- | --- | --- |
| 名称 | 字节数 | 含义 |
| biSize | 4 | 整个位图信息头结构体的大小 |
| biWidth | 4 | 图像宽度，单位为像素 |
| biHeight | 4 | 图像高度，单位为像素。  此外，这个数的正负可以判断图像是正向还是倒向的，若为正，则表示是正向；若为负，则表示反向。其实根本不同就是坐标系的建立方法不一样。后面写代码时会讲。 |
| biPlanes | 2 | 颜色平面书，其值总为1 |
| biBitCount | 2 | 即1个像素用多少位的数据来表示，其值可能为1，4，8，16，24，32。我们是以24位真彩色为例子讲解的 |
| biCompression | 4 | 数据的压缩类型 |
| biSizeImage | 4 | 图像数据的大小，单位为字节 |
| biXPelsPerMeter | 4 | 水平分辨率，单位是像素/米 |
| biYPelsPerMeter | 4 | 垂直分辨率，单位是像素/米 |
| biClrUsed | 4 | 调色板中的颜色索引数 |
| biClrImportant | 4 | 说明有对图像有重要影响的颜色索引的数目，若为0，表示都重要 |

对照源文件数据：



0E-11：00000028h = 40,表示这个结构体大小是40字节。

12-15：00000320h = 800，图像宽为800像素。

16-19：00000258h = 600，图像高为600像素，与文件属性一致。这是一个正数，说明图像是正向的，数据是以图像左下角为原点，以水平向右为X轴正方向，以垂直向上为Y轴正方向排列的。若为负，则说明图像是反向的，数据是以图像左上角角为原点，以水平向右为X轴正方向，以垂直向下为Y轴正方向排列的。

1A-1B：0001h, 该值总为1。

1C-1D：0018h = 24, 表示每个像素占24个比特，即24位真彩色

上面这几个信息跟文件属性是一致的：



1E-21：00000000h，BI\_RGB， 说明本图像不压缩。

22-25：00000000h，图像的大小，因为使用BI\_RGB，所以设置为0。

26-29：00000000h，水平分辨率，缺省。

2A-2D：00000000h，垂直分辨率，缺省。

2E-31：00000000h，对于24位真彩色来说，是没有调色板的，所以为0。

32-35：00000000h，对于24位真彩色来说，是没有调色板的，所以为0。

1. 调色板(color palette)

24位真彩色没有调色板，这里为了简化不赘诉。

1. 定义位图的字节阵列

这一部分就是真正的图像数据了，24位真彩色数据是按按BGR各一字节循环排列而成。

## 2.1.2 代码实现：将BMP文件解析为RGB格式，在LCD上显示

让BMP文件在开发板的LCD上显示出来，有几个需要注意的点：

1.开发板LCD上的显示格式是RGB格式的，而且有多种表示格式：可能用2字节表示（RGB565格式），可能用3字节表示（RGB888），而原始的24位真彩色BMP文件则是按BGR格式排列的，需要对原始的图像数据进行转化。

2.在转化过程中，LCD上的显存地址固定是以LCD左上角为首地址，而BMP格式中正向图像是以图片的左下角为数据首地址的。因此在进行数据转化时还需要注意坐标的变换。

代码清单2.1实现了将24位真彩色的BMP图像转化为RGB格式

代码清单2.1

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. \* 函数名称： IsBmp
3. \* 功能描述： 判断该文件是否为BMP文件
4. \* 输入参数： ptFileMap - 内含文件信息
5. \* 输出参数： 无
6. \* 返 回 值： 0 - 是BMP格式, -1 -不是BMP格式
7. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
8. **int** IsBmp(**FILE** \*\*ppFp, **const** **char** \*strFileName)
9. {
10. **char** strCheckHeader[2];
11. \*ppFp= fopen(strFileName, "rb+");
12. **if** (\*ppFp== NULL) {
13. **return** -1;
14. }
15. **if** (fread(strCheckHeader, 1, 2, \*ppFp) != 2)
16. **return** -1;
18. **if** (strCheckHeader[0] != 0x42 || strCheckHeader[1] != 0x4d)
19. **return** -1;
20. **else**
21. **return** 0;
22. }


26. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
27. \* 函数名称： MapFile
28. \* 功能描述： 使用mmap函数映射一个文件到内存,以后就可以直接通过内存来访问文件
29. \* 输入参数： PT\_PictureData ptData 内含图像数据
30. \* 输出参数： ptData->iFileSize     : 文件大小
31. \*                        ptData->pucFileData : 映射内存的首地址
32. \* 返 回 值： 0      - 成功其他值 - 失败
33. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
34. **int** MapFile(PT\_PictureData ptData)
35. {
36. **int** iFd;
37. **struct** stat tStat;
39. /\* 打开文件 \*/
40. iFd = fileno(ptData->ptFp);
41. fstat(iFd, &tStat);
42. ptData->iFileSize= tStat.st\_size;
43. ptData->pucFileData= (unsigned **char** \*)mmap(NULL , tStat.st\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, iFd, 0);
44. **if** (ptData->pucFileData == (unsigned **char** \*)-1)
45. {
46. printf("mmap error!\n");
47. **return** -1;
48. }
49. **return** 0;
50. }
52. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
53. \* 函数名称： DecodeBmp2Rgb
54. \* 功能描述：把BMP文件转化为rgb格式
55. \* 输入参数： strFileName - 文件名
56. \*                   ptData - 内含图像信息
57. \* 返 回 值： 0      - 成功其他值 - 失败
58. \*                         -1     - 文件不是BMP格式
59. \*                         -2     - 不支持的bpp
60. \*                         -3     - 图像缓存区分配失败
61. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
62. **static** **int** DecodeBmp2Rgb(**const** **char** \*strFileName, PT\_PictureData ptData) {
63. **int** x,y;
64. **int** iPos = 0;
65. **int** iLineWidthAlign;
66. BITMAPFILEHEADER \*ptBITMAPFILEHEADER;
67. BITMAPINFOHEADER \*ptBITMAPINFOHEADER;
68. unsigned **char** \*aFileHead;
69. unsigned **char** \*pucSrc;
70. unsigned **char** \*pucDest;
71. **int** iLineBytes;
73. /\* 判断该文件是否为BMP格式 \*/
74. **if** (IsBmp(&ptData->ptFp, strFileName))
75. **return** -1;
77. /\* 将BMP文件映射到内存空间 \*/
78. MapFile(ptData);

81. aFileHead = ptData->pucFileData;
83. ptBITMAPFILEHEADER = (BITMAPFILEHEADER \*)aFileHead;
84. ptBITMAPINFOHEADER = (BITMAPINFOHEADER \*)(aFileHead + **sizeof**(BITMAPFILEHEADER));
85. /\* 获取必要的图像信息 \*/
86. ptData->iWidth = ptBITMAPINFOHEADER->biWidth;
87. ptData->iHeight = ptBITMAPINFOHEADER->biHeight;
88. ptData->iBpp = ptBITMAPINFOHEADER->biBitCount;
89. iLineBytes    = ptData->iWidth\*ptData->iBpp/8;//一行数据的字节数
90. ptData->iBmpDataSize= ptData->iHeight \* iLineBytes;//整个BMP图像的字节数
91. /\*暂时只支持24bpp格式\*/
92. **if** (ptData->iBpp != 24)
93. {
94. printf("iBMPBpp = %d\n", ptData->iBpp);
95. printf("sizeof(BITMAPFILEHEADER) = %d\n", **sizeof**(BITMAPFILEHEADER));
96. **return** -2;
97. }
99. /\* 分配空间 \*/
100. ptData->pucBmpData = malloc(ptData->iBmpDataSize);
101. ptData->pucRgbData = malloc(ptData->iBmpDataSize);
103. **if** (NULL == ptData->pucBmpData||NULL == ptData->pucRgbData)
104. **return** -2;
106. /\* 从bmp文件中读取图像信息，24bpp的BMP图像为BGR格式 \*/
107. pucDest = ptData->pucBmpData;
108. iLineWidthAlign = (iLineBytes + 3) & ~0x3;   /\* 向4取整 \*/
109. pucSrc = aFileHead + ptBITMAPFILEHEADER->bfOffBits;
111. pucSrc = pucSrc + (ptData->iHeight - 1) \* iLineWidthAlign;
113. /\* 对于bmp文件中的源数据，是以左下角为原点计算坐标的，因此拷贝数据时需要转换坐标 \*/
114. **for** (y = 0; y < ptData->iHeight; y++)
115. {
116. memcpy(pucDest, pucSrc, ptData->iWidth\*3);
117. pucSrc  -= iLineWidthAlign;
118. pucDest += iLineBytes;
119. }

122. /\* 将得到的BGR数据转化为RGB数据 \*/
123. **for** (y = 0; y < ptData->iHeight; y++){
124. **for**(x = 0;x<ptData->iWidth\*3;x+=3){
125. ptData->pucRgbData[iPos++] = ptData->pucBmpData[y\*ptData->iWidth\*3+x+2];
126. ptData->pucRgbData[iPos++] = ptData->pucBmpData[y\*ptData->iWidth\*3+x+1];
127. ptData->pucRgbData[iPos++] = ptData->pucBmpData[y\*ptData->iWidth\*3+x+0];
128. }
129. }
131. **return** 0;
133. }

# 2.2 JPEG图像处理

## 2.2.1 JPEG文件格式和libjpeg编译

JPEG的后缀名为.jpg的图像文件。对于图像内容和信息相同的JPEG文件和BMP文件，JPEG格式的文件要比BMP格式的文件小得多，这是因为JPEG文件是经过JPEG压缩算法后得到的一种文件格式。

相对于BMP格式的文件,JPEG由于压缩算法的关系，其文件解析较为复杂，我们可以利用Linux系统开源的优点，使用开源工具对jpeg文件进行格式的解析和转换。

我们可以使用libjpeg库来对jpeg文件进行格式的解析和转换。libjpeg支持X86，ARM等架构。libjpeg是开源工具，所以可以在网上免费下载。

在使用libjpeg之前，我们先要交叉编译libjpeg的库文件和头文件并存到开发板的文件系统中。以下是libjpeg的编译过程：

1. 解压并进入文件目录

tar xzf libjpeg-turbo-1.2.1.tar.gz

cd libjpeg-turbo-1.2.1/

1. 交叉编译

tar xzf libjpeg-turbo-1.2.1.tar.gz

./configure --prefix=/work/projects/libjpeg-turbo-1.2.1/tmp/ --host=arm-linux

make

make install

1. 将编译出来的头文件和库文件拷贝到交叉编译器的相应目录下

cd /work/projects/libjpeg-turbo-1.2.1/tmp/include

cp \* /usr/local/arm/4.3.2/arm-none-linux-gnueabi/libc/usr/include

cd /work/projects/libjpeg-turbo-1.2.1/tmp/lib

cp \*so\* -d /usr/local/arm/4.3.2/arm-none-linux-gnueabi/libc/armv4t/lib

4. 将编译出来的头文件和库文件拷贝到开发板文件系统的相应目录下

cd /work/projects/libjpeg-turbo-1.2.1/tmp/lib

cp \*.so\* /work/nfs\_root/fs\_mini\_mdev\_new/lib/ -d

## 2.2.2 libjpeg接口函数的解析和使用

libjpeg的使用方法可以参考解压包中的使用说明libjpeg.txt和例程example.c。libjpeg的使用步骤简单总结如下：

1. 分配和初始化一个jpeg\_compress\_struct结构体

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

jpeg\_create\_decompress(&cinfo);

1. 指定源文件

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

参数1是步骤1中分配的jpeg\_compress\_struct类型的结构体，参数2是要解析的JPEG文件的文件句柄。

1. 获得jpg信息头并设置解压参数

jpeg\_read\_header(&cinfo, TRUE);

当调用完这个参数之后，我们就可以通过cinfo中的image\_width，image\_height等成员来获得图像的信息了。此外我们还可以设置cinfo中的scale\_num和scale\_denom等成员变量来设置解压参数。

1. 启动解压

jpeg\_start\_decompress(&cinfo);

调用这个函数后，就可以对cinfo所指定的源文件进行解压，并将解压后的数据存到cinfo结构体的成员变量中。

1. 读取解压后数据

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

调用这个函数后，可以读取RGB数据到buffer中，参数3能指定读取多少行

1. 完成读取

jpeg\_finish\_decompress(&cinfo);

1. 释放jpeg\_compress\_struct结构体

jpeg\_destroy\_decompress(&cinfo);

完成读取后释放结构体

## 2.2.3 使用libjpeg把JPEG文件解析为RGB格式，在LCD上显示

根据上节的解析，利用上述的库函数将JPEG文件解析为RGB格式了。

代码清单2.2

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. \* 函数名称： IsJpg
3. \* 功能描述：判断是否为Jpg文件
4. \* 输入参数： ptData - 内含图像信息
5. strFileName - 文件名
6. \* 返 回 值：0 - 不是JPG格式 其他-是JPG格式
7. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
8. **static** **int** IsJpg(PT\_PictureData ptData, **const** **char** \*strFileName)
9. {
10. **int** iRet;
12. jpeg\_stdio\_src(&ptData->tInfo, ptData->ptFp);
14. /\* 用jpeg\_read\_header获得jpeg信息\*/
15. iRet = jpeg\_read\_header(&ptData->tInfo, TRUE);
17. **return** (iRet == JPEG\_HEADER\_OK);
18. }
20. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
21. \* 函数名称： DecodeJpg2Rgb
22. \* 功能描述：把JPG文件解析为RGB888格式
23. \* 输入参数： ptData - 内含文件信息
24. \*                             strFileName - 文件名
25. \* 输出参数：PT\_PictureData->pucRgbData - 内含rgb数据
26. \* 返 回 值：0 - 成功 其他-失败
27. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
28. **static** **int** DecodeJpg2Rgb(**const** **char** \*strFileName, PT\_PictureData ptData){
29. **int** iRowSize;
30. unsigned **char** \*pucbuffer;
31. unsigned **char** \*pucHelp;//辅助拷贝变量
33. /\* 1.分配和初始化一个jpeg\_compress\_struct结构体 \*/
34. ptData->tInfo.err = jpeg\_std\_error(&ptData->tJerr);
35. jpeg\_create\_decompress(&ptData->tInfo);

38. /\* 2.指定源文件\*/
39. **if** ((ptData->ptFp= fopen(strFileName, "rb")) == NULL) {
40. fprintf(stderr, "can't open %s\n", strFileName);
41. **return** -1;
42. }
44. /\* 3.获得jpg信息头并设置解压参数并判断是否为JPEG格式文件 \*/
45. **if** (!IsJpg(ptData, strFileName)) {
46. printf("file is not jpg ...\n");
47. **return** -1;
48. }


52. /\* 默认尺寸为原尺寸 \*/
53. ptData->tInfo.scale\_num = 1;
54. ptData->tInfo.scale\_denom = 1;
55. /\* 4. 启动解压：jpeg\_start\_decompress \*/
56. jpeg\_start\_decompress(&ptData->tInfo);

59. /\* 解压完成后可以通过tInfo中的成员获得图像的某些信息 \*/
60. ptData->iWidth= ptData->tInfo.output\_width;
61. ptData->iHeight = ptData->tInfo.output\_height;
62. ptData->iBpp = ptData->tInfo.output\_components\*8;
63. /\* 计算一行的数据长度 \*/
64. iRowSize = ptData->iWidth \* ptData->tInfo.output\_components;
65. pucbuffer = malloc(iRowSize);
66. ptData->iRgbSize= iRowSize \* ptData->iHeight;
67. ptData->pucRgbData = malloc(ptData->iRgbSize);
69. /\* pucHelp指向ptData->pucRgbData首地址 \*/
70. pucHelp = ptData->pucRgbData;
71. /\*  5.循环调用jpeg\_read\_scanlines来一行一行地获得解压的数据 \*/
72. **while** (ptData->tInfo.output\_scanline < ptData->tInfo.output\_height)
73. {
74. /\* 调用jpeg\_read\_scanlines得到的时候会存到pucbuffer中 \*/
75. jpeg\_read\_scanlines(&ptData->tInfo, &pucbuffer, 1);
76. /\* 将数据一行行读到缓冲区中 \*/
77. memcpy(pucHelp,pucbuffer,iRowSize);
78. pucHelp  += iRowSize;
79. }
80. free(pucbuffer);
81. /\* 6.完成读取 \*/
82. jpeg\_finish\_decompress(&ptData->tInfo);
83. /\* 7.释放jpeg\_compress\_struct结构体 \*/
84. jpeg\_destroy\_decompress(&ptData->tInfo);
85. **return** 0;
86. }

# 2.3 PNG图像处理

## 2.3.1 PNG文件格式和libpng编译

跟JPEG文件格式一样，PNG也是一种使用了算法压缩后的图像格式，与JPEG不同，PNG使用从LZ77派生的无损数据压缩算法。对于PNG文件格式，也有相应的开源工具libpng。

libpng库可从官网上下载最新的源代码：

<http://www.libpng.org/pub/png/libpng.html>

在使用libpng之前，我们先要交叉编译libpng的库文件和头文件并存到开发板的文件系统中。以下是libpng的编译过程：

1. 解压并进入文件目录

tar xzf libpng-1.6.37.tar.gz

cd libpng-1.6.37/

1. 交叉编译

./configure --prefix=/work/projects/libpng-1.6.37/tmp/ --host=arm-linux

make

make install

1. 将编译出来的头文件和库文件拷贝到交叉编译器的相应目录下

cd /work/projects/libpng-1.6.37/tmp/include

cp \* /usr/local/arm/4.3.2/arm-none-linux-gnueabi/libc/usr/include

cd /work/projects/libpng-1.6.37/tmp/lib

cp \*so\* -d /usr/local/arm/4.3.2/arm-none-linux-gnueabi/libc/armv4t/lib

4. 将编译出来的头文件和库文件拷贝到开发板文件系统的相应目录下

cd /work/projects/libpng-1.6.37/tmp/lib

cp \*.so\* /work/nfs\_root/fs\_mini\_mdev\_new/lib/ -d

## 2.3.2 libpng接口函数的解析和使用

libpng的使用方法可以参考解压包中的使用说明libpng-manual.txt和例程example.c。libjpeg的使用步骤简单总结如下：

1. 分配和初始化两个与libpng相关的结构体png\_ptr，info\_ptr
2. png\_ptr = png\_create\_read\_struct(PNG\_LIBPNG\_VER\_STRING, NULL, NULL, NULL);

参数2，3，4分别是用户自定义的错误处理函数，若无，则填NULL。

1. info\_ptr = png\_create\_info\_struct(png\_ptr);
2. 设置错误返回点

setjmp(png\_jmpbuf(png\_ptr));

当出现错误时，libpng将会自动调用返回到这个点。在这个点我们可以进行一些清理工作。如果在调用png\_create\_read\_struct时没有设置自定义的错误处理函数，这一步是必须要做的。

1. 指定源文件

png\_init\_io(png\_ptr, fp);

参数1是步骤1中分配的png\_ptr结构体，参数2是需要解析的PNG文件的文件句柄。

1. 获取PNG图像的信息
2. 解析图片数据信息

png\_read\_png(png\_ptr, info\_ptr, png\_transforms, png\_voidp\_NULL);

该函数会把所有的图片数据解码到info\_ptr数据结构中。至于转化为什么格式，由参数png\_transforms决定，它是一个整型参数，可以使用libpng库中定义的宏进行传参。这个参数相关的宏有很多，具体的可以参考库中的相关文件的解析。

B.查询图像信息

此外，我们还可以通过png\_get\_image\_width，png\_get\_image\_height，png\_get\_color\_type等函数获得png图像的宽度，高度，颜色类型等信息，更多的图像信息获取函数可以在文件pngget.c中找到。

1. 将info\_ptr中的图像数据读取出来

有两种读取PNG图像信息的方法：

1. 一次性把所有的数据读入内存

png\_read\_image(png\_ptr, row\_pointers);

参数1是步骤1中分配的png\_ptr，参数2是存放图片数据的指针。

1. 也可以逐行读取

row\_pointers = png\_get\_rows(png\_ptr, info\_ptr);

参数1和参数2分别是步骤1中分配的png\_ptr, info\_ptr，返回值是每行数据的首地址。

参数1是步骤1中分配的png\_ptr，参数2是存放图片数据的指针。

1. 销毁内存

png\_destroy\_read\_struct(&png\_ptr, &info\_ptr, 0);

## 2.3.3 使用libpng把png文件转为rgb格式，在LCD上显示

代码清单2.3

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. \* 函数名称： IsnotPng
3. \* 功能描述：判断是否为PNG文件
4. \* 输入参数： ppFp - 文件句柄指针
5. strFileName - 文件名
6. \* 返 回 值：0 - 是PNG格式 其他-不是PNG格式
7. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
8. **int** IsnotPng(**FILE** \*\*ppFp, **const** **char** \*strFileName)
9. {
10. **char** strCheckHeader[8];
11. \*ppFp= fopen(strFileName, "rb");
12. **if** (\*ppFp== NULL) {
13. **return** -1;
14. }
15. /\* 读取PNG文件前8个字节，使用库函数png\_sig\_cmp即可判断是否为PNG格式 \*/
16. **if** (fread(strCheckHeader, 1, 8, \*ppFp) != 8)
17. **return** -1;
18. **return** png\_sig\_cmp(strCheckHeader, 0, 8);
20. }
22. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
23. \* 函数名称： DecodePng2Rgb
24. \* 功能描述：把PNG文件解析为RGB888格式
25. \* 输入参数： ptData - 内含文件信息
26. \*                             strFileName - 文件名
27. \* 输出参数：PT\_PictureData->pucRgbData - 内含rgb数据
28. \* 返 回 值：0 - 成功 其他-失败
29. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
30. **static** **int** DecodePng2Rgb(**const** **char** \*strFileName, PT\_PictureData ptData)
31. {
32. **int** i, j;
33. **int** iPos = 0;
34. png\_bytepp pucPngData;
35. /\* 0.判断该文件是否为PNG格式 \*/
36. **if** (IsnotPng(&ptData->ptFp, strFileName)) {
37. printf("file is not png ...\n");
38. **return** -1;
39. }
41. /\* 1.分配和初始化两个与libpng相关的结构体png\_ptr，info\_ptr \*/
42. ptData->ptPngStrPoint  = png\_create\_read\_struct(PNG\_LIBPNG\_VER\_STRING, NULL, NULL, NULL);
43. ptData->ptPngInfoPoint= png\_create\_info\_struct(ptData->ptPngStrPoint);
45. /\* 2.设置错误的返回点 \*/
46. setjmp(png\_jmpbuf(ptData->ptPngStrPoint));
47. rewind(ptData->ptFp); //等价fseek(fp, 0, SEEK\_SET);
49. /\* 3.指定源文件 \*/
50. png\_init\_io(ptData->ptPngStrPoint, ptData->ptFp);
52. /\* 4.获取PNG图像数据信息和通道数，宽度，高度等
53. \* 使用PNG\_TRANSFORM\_EXPAND宏做参数的作用是根据通道数的不同，
54. \* 将PNG图像转换为BGR888或ABGR8888格式\*/
55. png\_read\_png(ptData->ptPngStrPoint, ptData->ptPngInfoPoint, PNG\_TRANSFORM\_EXPAND, 0);
56. ptData->iChannels    = png\_get\_channels(ptData->ptPngStrPoint, ptData->ptPngInfoPoint);
57. ptData->iWidth    = png\_get\_image\_width(ptData->ptPngStrPoint, ptData->ptPngInfoPoint);
58. ptData->iHeight  = png\_get\_image\_height(ptData->ptPngStrPoint, ptData->ptPngInfoPoint);

61. /\* 5.将info\_ptr中的图像数据读取出来 \*/
62. pucPngData = png\_get\_rows(ptData->ptPngStrPoint, ptData->ptPngInfoPoint); //也可以分别每一行获取png\_get\_rowbytes();
63. **if** (ptData->iChannels == 4) { //判断是24位还是32位
64. ptData->iRawSize= ptData->iWidth \* ptData->iHeight\*4; //申请内存先计算空间
65. ptData->pucRawData= (unsigned **char**\*)malloc(ptData->iRawSize);
66. **if** (NULL == ptData->pucRawData) {
67. printf("malloc rgba faile ...\n");
68. png\_destroy\_read\_struct(&ptData->ptPngStrPoint, &ptData->ptPngInfoPoint, 0);
69. fclose(ptData->ptFp);
70. **return** -1;
71. }
72. /\* 从pucPngData里读出实际的RGBA数据出来
73. \* 源数据为ABGR格式\*/
74. **for** (i = 0; i < ptData->iHeight; i++)
75. **for** (j = 0; j < ptData->iWidth \* 4; j += 4) {
76. ptData->pucRawData[iPos++] = pucPngData[i][j + 3];
77. ptData->pucRawData[iPos++] = pucPngData[i][j + 2];
78. ptData->pucRawData[iPos++] = pucPngData[i][j + 1];
79. ptData->pucRawData[iPos++] = pucPngData[i][j + 0];
80. }
82. /\* 将得到的RGBA转换为RGB888格式 \*/
83. **if**(RgbaToRgb(ptData)!=0)
84. **return** -1;
86. }
87. **else** **if** (ptData->iChannels == 3 ) { //判断颜色深度是24位还是32位
88. ptData->iRgbSize= ptData->iWidth \* ptData->iHeight\*3; //申请内存先计算空间
89. ptData->pucRgbData = (unsigned **char**\*)malloc(ptData->iRgbSize);
90. **if** (NULL == ptData->pucRgbData) {
91. printf("malloc rgba faile ...\n");
92. png\_destroy\_read\_struct(&ptData->ptPngStrPoint, &ptData->ptPngInfoPoint, 0);
93. fclose(ptData->ptFp);
94. **return** -1;
95. }
96. /\* 从pucPngData里读出实际的RGB数据
97. \* 源数据为BGR格式\*/
98. **for** (i = 0; i < ptData->iHeight; i ++) {
99. **for** (j = 0; j < ptData->iWidth\*3; j += 3) {
100. ptData->pucRgbData[iPos++] = pucPngData[i][j+2];
101. ptData->pucRgbData[iPos++] = pucPngData[i][j+1];
102. ptData->pucRgbData[iPos++] = pucPngData[i][j+0];
103. }
104. }
105. ptData->iBpp = 24;//转化之后的格式为RGB888格式
106. }
107. **else** **return** -1;

110. /\* 6:销毁内存 \*/
111. png\_destroy\_read\_struct(&ptData->ptPngStrPoint, &ptData->ptPngInfoPoint, 0);
112. fclose(ptData->ptFp);

115. **return** 0;
116. }

# 2.4 图像调整

## 2.4.1 图像的缩放

### 2.4.1.1 图像缩放算法浅析

图像缩放算法有很多种，这里参考网友"lantianyu520"所著的"图像缩放算法"。

原理浅析

要理解这个图像缩放算法的原理，最重要的是需要理解：对于图像上的每一个像素点，它缩放前后，相对于整个图像的比例应该是一样的。

比如：

以一个长度和宽度分别为200，100的长方形为例，将其放大两倍，那么缩放后的长度和宽度为400，200。

为方便理解，我们建立一个笛卡尔坐标系，把这个长方形左下角的顶点放到坐标(0,0)位置，四个点的坐标分别为：(0,0),(0,100),(200,0),(200,100)。

假设此时对长方形中的坐标点(40,50)，它的x坐标相对于长的比值是40/200=0.2，y坐标相对于宽的比值是50/100=0.5，那么该点的变换后的坐标Dx,Dy则应满足：Dx/400 = 5;Dy/200 = 0.5，这样，缩放后的坐标就可以算出来了。

根据上面的分析，设缩放前的像素点坐标为(Sx,Sy)，对应的缩放后的像素点坐标为(Dx,Dy)，缩放前的图像长宽分别为Sw,Sh，缩放后的图像长宽分别为Dw,Dh，则有：

Sx/Dx = Sw/Dw，Sy/Dy = Sh/Dh

故有Sx = Dx \* Sw/Dw，Sy = Dy \* Sh/Dh，

### 2.4.1.2源码编写：图像缩放算法

有了这个上面两条等式后，图像缩放算法的代码就好理解了。

下面的函数实现了基于上述原理实现的图像缩放算法：

代码清单2.4

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. \* 函数名称： PicZoom
3. \* 功能描述： 近邻取样插值方法缩放图片
4. \*            注意该函数会分配内存来存放缩放后的图片,用完后要用free函数释放掉
5. \*            "近邻取样插值"的原理请参考网友"lantianyu520"所著的"图像缩放算法"
6. \* 输入参数： ptPicData - 内含缩放前后的图像数据
7. \*            fSize    - 缩放倍数
8. \* 输出参数： ptPicData->pucZoomData,内含缩放后的数据
9. \* 返 回 值： 0 - 成功, 其他值 - 失败
10. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
11. **int** PicZoom(PT\_PictureData ptPicData,**float** fSize)
12. {
13. ptPicData->iZoomWidth = ptPicData->iWidth \* fSize;
14. ptPicData->iZoomHeight= ptPicData->iHeight\* fSize;
15. unsigned **long**\* pdwSrcXTable;
16. unsigned **long** x;
17. unsigned **long** y;
18. unsigned **long** dwSrcY;
19. unsigned **char** \*pucDest;
20. unsigned **char** \*pucSrc;
21. unsigned **long** dwPixelBytes = ptPicData->iBpp/8;
22. ptPicData->pucZoomData= malloc(**sizeof**(unsigned **char**) \* ptPicData->iZoomWidth\*ptPicData->iZoomHeight\*ptPicData->iBpp/8);
23. pdwSrcXTable = malloc(**sizeof**(unsigned **long**) \* ptPicData->iZoomWidth);
24. **if** (NULL == pdwSrcXTable){
25. printf("malloc error!\n");
26. **return** -1;
27. }
29. /\* 这几个for循环的本质是Sx = Dx \* Sw/Dw，Sy = Dy \* Sh/Dh\*/
30. **for** (x = 0; x < ptPicData->iZoomWidth; x++){//生成表 pdwSrcXTable
31. /\* 第一个for循环对应x方向的坐标
32. \* pdwSrcXTable[x] 对应Sx,
33. \* x 对应Dx,
34. \* ptPicData->iWidth 对应Sw
35. \* ptPicData->iZoomWidth 对应 Dw\*/
36. pdwSrcXTable[x]=(x\*ptPicData->iWidth/ptPicData->iZoomWidth);
37. }
39. **for** (y = 0; y < ptPicData->iZoomHeight; y++){
40. /\* 第2个循环对应y方向的坐标
41. \* dwSrcY 对应Sy,
42. \* y 对应Dy,
43. \* ptPicData->iHeight 对应Sh
44. \* ptPicData->iZoomHeight 对应 Dh\*/
45. dwSrcY = (y \* ptPicData->iHeight / ptPicData->iZoomHeight);
46. /\* 根据这些可算得各像素点的RGB数据存放的地址 \*/
47. pucDest = ptPicData->pucZoomData + y\*ptPicData->iZoomWidth\*3;
48. pucSrc  = ptPicData->pucRgbData + dwSrcY\*ptPicData->iWidth\*3;
50. /\* 最后拷贝数据 \*/
51. **for** (x = 0; x <ptPicData->iZoomWidth; x++){
52. memcpy(pucDest+x\*dwPixelBytes, pucSrc+pdwSrcXTable[x]\*dwPixelBytes, dwPixelBytes);
53. }
54. }
56. free(pdwSrcXTable);
57. **return** 0;
58. }

## 2.4.2 图像的旋转

### 2.4.2.1 图像旋转算法浅析

这里的图像旋转算法原理参考网友"落叶的思维"所著的"图像旋转算法与实现"

原理浅析

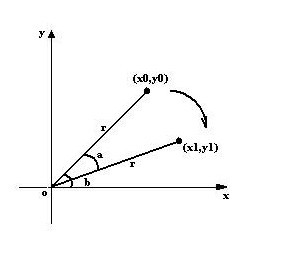
这个旋转算法的原理的关键点有两个：

1. 原图像是以图像的左下角为原点建立笛卡尔坐标系的，而旋转一般是以图像的中心作为旋转点旋转的。

因此为了便于转换，我们先约定两个坐标系，一个是以图像左下角为原点建立的坐标系，称为坐标系A，这也是原图像的坐标系。一个是以图像中心为原点建立的坐标系，称为坐标系B。

由此，可以知道这个旋转算法的步骤：先将坐标系A下的坐标转换为坐标系B下的坐标，然后在坐标系B下进行图像的旋转。

在坐标系B下，我们假设点（x0，y0）距离原点的距离为r，点与原点之间的连线与x轴的夹角为b，旋转的角度为a，旋转后的点为（x1，y1）, 如下图所示。



那么有以下结论：

x0=rcosb；y0=rsinb

x1 = rcos(b-a) = rcosbcosa+rsinbsina=x0cosa+y0sina；

y1=rsin(b-a)=rsinbcosa-rcosbsina=-x0sina+y0cosa；

最后，由于我们显示图像的RGB数据还是要在坐标系A下获取的，我们最后只需要将坐标系B下的x1,y1转换回坐标系A下的坐标就可以了。

旋转后的图像的长和宽会发生变化，因此要计算新图像的长和宽。

由几何关系可知，新图像的长和宽分别是旋转后，对角坐标相见后的最大值

### 2.4.2.2 源码编写：图像旋转算法

代码清单2.5

1. #define PI 3.1415926535
2. //角度到弧度转化
3. #define RADIAN(angle) ((angle)\*PI/180.0)




9. **typedef** **struct** ConcernCoor {
10. **int** iLTx;// left top x
11. **int** iLTy;//left top y
12. **int** iLBx;//left bottom x
13. **int** iLBy;//left bottom y
14. **int** iRTx;//right top x
15. **int** iRTy;//right top y
16. **int** iRBx;// right bottom x
17. **int** iRBy;// right bottom y
18. }T\_ConcernCoor, \*PT\_ConcernCoor;

21. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
22. \* 函数名称： max
23. \* 功能描述：比较两个参数，返回较大值
24. \* 输入参数：x，y均为int型
25. \* 输出参数： 无
26. \* 返 回 值： x，y中的较大值
27. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
28. **static** **int** max(**int** x,**int** y){
29. **return** x>y?x:y;
30. }
31. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
32. \* 函数名称： PicRotate
33. \* 功能描述： 旋转图片
34. \*            注意该函数会分配内存来存放缩放后的图片,用完后要用free函数释放掉
35. \*              参考网友"落叶的思维"所著的"图像旋转算法与实现"
36. \* 输入参数： ptPicData - 内含图片的象素数据
37. \*            fAngle    - 旋转角度，0<=angle<=360
38. \* 输出参数： ptPicData->pucRotateData,内含旋转后的rgb数据
39. \* 返 回 值： 0 - 成功, 其他值 - 失败
40. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
41. **int** PicRotate(PT\_PictureData ptPicData,**float** fAngle)
42. {
43. **int** i ,j;
44. T\_ConcernCoor tConCor,tRonCor;
45. //原图像每一行去除偏移量的字节数
46. //int iSrcLineSize = bitCount \* srcW / 8;
47. **int** iSrcLineSize = ptPicData->iBpp\* ptPicData->iZoomWidth / 8;
48. **int** iDesLineSize;
49. **int** iX;//旋转后的x坐标
50. **int** iY; //旋转后的y坐标
52. /\* 将坐标系A下的坐标转换为坐标系B下的坐标,
53. \* 用于计算旋转后的图像的宽和高
54. \* tConCor用于存放坐标系B下旋转前的坐标
55. \* tRonCor用于存放坐标系B下旋转后的坐标\*/
56. tConCor.iLTx = -ptPicData->iZoomWidth/2; tConCor.iLTy = ptPicData->iZoomHeight/2;
57. tConCor.iRTx = ptPicData->iZoomWidth/2; tConCor.iRTy = ptPicData->iZoomHeight/2;
58. tConCor.iLBx = -ptPicData->iZoomWidth/2;tConCor.iLBy = -ptPicData->iZoomHeight/2;
59. tConCor.iRBx = ptPicData->iZoomWidth/2;tConCor.iRBy = -ptPicData->iZoomHeight/2;

62. /\* 计算坐标系B下旋转后的坐标 \*/
63. **double** sina = sin(RADIAN(fAngle));
64. **double** cosa = cos(RADIAN(fAngle));
65. tRonCor.iLTx =tConCor.iLTx \* cosa + tConCor.iLTy \* sina;
66. tRonCor.iLTy = -tConCor.iLTx \* sina + tConCor.iLTy \* cosa;
67. tRonCor.iRTx =tConCor.iRTx \* cosa + tConCor.iRTy \* sina;
68. tRonCor.iRTy = -tConCor.iRTx \* sina + tConCor.iRTy \* cosa;
69. tRonCor.iLBx = tConCor.iLBx \* cosa + tConCor.iLBy \* sina;
70. tRonCor.iLBy = -tConCor.iLBx \* sina + tConCor.iLBy \* cosa;
71. tRonCor.iRBx = tConCor.iRBx \* cosa + tConCor.iRBy \* sina;
72. tRonCor.iRBy = -tConCor.iRBx \* sina + tConCor.iRBy \* cosa;

75. /\* 计算旋转后图像宽和高 \*/
76. ptPicData->iRotateWidth = max(abs(tRonCor.iRBx - tRonCor.iLTx),abs(tRonCor.iRTx - tRonCor.iLBx));
77. ptPicData->iRotateHeight = max(abs(tRonCor.iRBy - tRonCor.iLTy),abs(tRonCor.iRTy - tRonCor.iLBy));
79. /\* 像素信息要保证3字节对齐，否则数据有可能出错\*/
80. iDesLineSize = ((ptPicData->iRotateWidth\* ptPicData->iBpp+ 23) / 24) \* 3 ;
81. /\* 分配旋转后的空间，注意这里要用旋转后的宽和高 \*/
82. ptPicData->pucRotateData = malloc(iDesLineSize \* ptPicData->iRotateHeight);
83. **if**(NULL == ptPicData->pucRotateData){
84. printf("malloc error\n");
85. **return** -1;
86. }
88. /\* 通过新图像的坐标，计算对应的原图像的坐标\*
89. \* i,j坐标就是对应的坐标系B下的x1,y1\*/
90. **for** (i = 0; i < ptPicData->iRotateHeight; i++){
91. **for** (j = 0; j < ptPicData->iRotateWidth; j++){
92. /\* 坐标系B下的x,y1坐标，经过逆运算转换得到iX,iY,这两个值对应x0,y0 \*/
93. iX = (j - ptPicData->iRotateWidth / 2)\*cos(RADIAN(360 - fAngle)) + (-i + ptPicData->iRotateHeight / 2)\*sin(RADIAN(360 - fAngle));
94. iY = -(j - ptPicData->iRotateWidth / 2)\*sin(RADIAN(360 - fAngle)) + (-i + ptPicData->iRotateHeight / 2)\*cos(RADIAN(360 - fAngle));
95. /\*如果这个坐标不在原图像内，则不赋值\*/
96. **if** (iX > ptPicData->iZoomWidth / 2 || iX < -ptPicData->iZoomWidth / 2 || iY > ptPicData->iZoomHeight / 2 || iY < -ptPicData->iZoomHeight / 2){
97. **continue**;
98. }
99. /\* 再将坐标系B下的x0,y0坐标，转换为坐标系A下的坐标 \*/
100. **int** iXN = iX + ptPicData->iZoomWidth / 2;
101. **int** iYN = abs(iY - ptPicData->iZoomHeight  / 2);
102. /\* 值拷贝\*/
103. memcpy(&ptPicData->pucRotateData[i \* iDesLineSize + j \* 3],&ptPicData->pucZoomData[iYN \* iSrcLineSize + iXN \* 3],3);
104. }
105. }
106. **return** 0;
107. }