

一个自由、开源的图形库

FreeImage开发小组 翻译:湖南地震局 黎品忠

2005年12月29日

目录

第一章 介绍	1
1.1 前言	1
1.2 译者的话	1
1.3 FreeImage的目标	2
1.4 库参考	2
第二章 位图函数参考	3
2.1 通用函数	3
FreeImage_Initialise	3
FreeImage_DeInitialise	3
FreeImage_GetVersion	3
FreeImage_GetCopyrightMessage	3
FreeImage_SetOutputMessage	4
2.2 位图管理函数	4
FreeImage_Allocate	6
FreeImage_AllocateT	6
FreeImage_Load	7
FreeImage_LoadU	8
FreeImage_LoadFromHandle	9
FreeImage_Save	9
FreeImage_SaveU	10
FreeImage_SaveToHandle	10
FreeImage_Clone	11
FreeImage_Unload	12
2.3 位图信息函数	12
FreeImage_GetImageType	12
FreeImage_GetColorsUsed	12
FreeImage_GetBPP	13
FreeImage_GetWidth	13

FreeImage_GetHeight	13
FreeImage_GetLine	13
FreeImage_GetPitch	13
FreeImage_GetDIBSize	13
FreeImage_GetPalette	13
FreeImage_GetDotsPerMeterX	14
FreeImage_GetDotsPerMeterY	14
FreeImage_SetDotsPerMeterX	14
FreeImage_SetDotsPerMeterY	14
FreeImage_GetInfoHeader	14
FreeImage_GetInfo	15
FreeImage_GetColorType	15
FreeImage_GetRedMask	16
FreeImage_GetGreenMask	16
FreeImage_GetBlueMask	16
FreeImage_GetTransparencyCount	16
FreeImage_GetTransparencyTable	17
FreeImage_SetTransparencyTable	17
FreeImage_SetTransparent	18
FreeImage_IsTransparent	18
FreeImage_HasBackgroundColor	18
FreeImage_GetBackgroundColor	18
FreeImage_SetBackgroundColor	18
2.4 文件类型函数	19
FreeImage_GetFileType	19
FreeImage_GetFileTypeU	20
FreeImage_GetFileTypeFromHandle	21
FreeImage_GetFileTypeFromMemory	21
2.5 像素访问函数	21
FreeImage_GetBits	23
FreeImage_GetScanLine	25
FreeImage_GetPixelIndex	28
FreeImage_GetPixelColor	29
FreeImage_SetPixelIndex	29
FreeImage_SetPixelColor	29
2.6 转换函数	29
FreeImage_ConvertTo4Bits	30
FreeImage_ConvertTo8Bits	30

FreeImage_ConvertToGreyscale	30
FreeImage_ConvertTo16Bits555	30
FreeImage_ConvertTo16Bits565	31
FreeImage_ConvertTo24Bits	31
FreeImage_ConvertTo32Bits	31
FreeImage_ColorQuantize	31
FreeImage_ColorQuantizeEx	32
FreeImage_Threshold	33
FreeImage_Dither	34
FreeImage_ConvertFromRawBits	34
FreeImage_ConvertToRawBits	35
FreeImage_ConvertToStandardType	35
FreeImage_ConvertToType	36
FreeImage_ConvertToRGBF	37
2.7 调和映射操作算子	37
FreeImage_ToneMapping	37
FreeImage_TmoDrago03	38
FreeImage_TmoReinhard05	38
2.8 ICC 轮廓函数	39
FreeImage_GetICCProfile	40
FreeImage_CreateICCProfile	40
FreeImage_DestroyICCProfile	40
2.9 插件函数	41
FreeImage_GetFIFCount	41
FreeImage_SetPluginEnabled	41
FreeImage_IsPluginEnabled	41
FreeImage_GetFIFFromFormat	41
FreeImage_GetFIFFromMime	42
FreeImage_GetFIFMimeType	42
FreeImage_GetFormatFromFIF	42
FreeImage_GetFIFExtensionList	42
FreeImage_GetFIFDescription	44
FreeImage_GetFIFRegExpr	44
FreeImage_GetFIFFromFilename	44
FreeImage_GetFIFFromFilenameU	45
FreeImage_FIFSupportsReading	45
FreeImage_FIFSupportsWriting	46
FreeImage_FIFSupportsExportType	47

FreeImage_FIFSupportsExportBPP	47
FreeImage_FIFSupportsICCProfiles	48
FreeImage_RegisterLocalPlugin	49
FreeImage_RegisterExternalPlugin	49
2.10 多页函数	50
FreeImage_OpenMultiBitmap	50
FreeImage_CloseMultiBitmap	50
FreeImage_GetPageCount	50
FreeImage_AppendPage	51
FreeImage_InsertPage	51
FreeImage_DeletePage	51
FreeImage_LockPage	51
FreeImage_UnlockPage	51
FreeImage_MovePage	51
FreeImage_GetLockedPageNumbers	51
2.11 内存输入/输出流	52
FreeImage_OpenMemory	52
FreeImage_CloseMemory	52
FreeImage_LoadFromMemory	52
FreeImage_SaveToMemory	54
FreeImage_AcquireMemory	55
FreeImage_TellMemory	56
FreeImage_SeekMemory	56
2.12 压缩函数	56
FreeImage_ZLibCompress	57
FreeImage_ZLibUncompress	57
FreeImage_ZLibGZip	58
FreeImage_ZLibCRC32	59
FreeImage_ZlibGUnzip	59
2.13 帮助函数	59
FreeImage_IsLittleEndian	59
FreeImage_LookupX11Color	59
FreeImage_LookupSVGColor	60
第三章 元数据函数参考	61
3.1 介绍	61
FreeImage标签(Tag)	61
FreeImage元数据模型	63
3.2 标签的创建和销毁	64

FreeImage_CreateTag	64
FreeImage_DeleteTag	64
FreeImage_CloneTag	64
3.3 标签访问器	64
FreeImage_GetTagKey	64
FreeImage_GetTagDescription	65
FreeImage_GetTagID	65
FreeImage_GetTagType	65
FreeImage_GetTagCount	65
FreeImage_GetTagLength	65
FreeImage_GetTagValue	65
FreeImage_SetTagKey	65
FreeImage_SetTagDescription	65
FreeImage_SetTagID	66
FreeImage_SetTagType	66
FreeImage_SetTagCount	66
FreeImage_SetTagLength	66
FreeImage_SetTagValue	66
3.4 元数据迭代子	66
FreeImage_FindFirstMetadata	66
FreeImage_FindNextMetadata	67
FreeImage_FindCloseMetadata	67
3.5 元数据访问器	67
FreeImage_GetMetadata	67
FreeImage_SetMetadata	68
3.6 元数据帮助函数	69
FreeImage_GetMetadataCount	69
FreeImage_TagToString	70
第四章 工具包函数参考	71
4.1 旋转和翻转	71
FreeImage_RotateClassic	71
FreeImage_RotateEx	71
FreeImage_FlipHorizontal	72
FreeImage_FlipVertical	72
FreeImage_JPEGTransform	73
4.2 过采样/减像素采样	74
FreeImage_Rescale	74
4.3 颜色处理	75

FreeImage_AdjustCurve	75
FreeImage_AdjustGamma	76
FreeImage_AdjustBrightness	76
FreeImage_AdjustContrast	76
FreeImage_Invert	77
FreeImage_GetHistogram	77
4.4 通道处理	77
FreeImage_GetChannel	77
FreeImage_SetChannel	77
FreeImage_GetComplexChannel	78
FreeImage_SetComplexChannel	78
4.5 复制/粘贴/合成例程	78
FreeImage_Copy	78
FreeImage_Paste	78
FreeImage_Composite	79
附录	82
附录 A 选择正确的重采样滤镜	83
A.1 箱形(Box)滤镜	83
A.2 双线性(Bilinear)滤镜	83
A.3 B样条滤镜	83
A.4 二维立方(Bicubic)滤镜	83
A.5 Catmull-Rom滤镜	84
A.6 Lanczos滤镜	84
附录 B 各种重采样方法的比较	85
附录 C 使用旋转函数	87
C.1 FreeImage_RotateClassic	87
C.2 FreeImage_RotateEx	89
附录 D FreeImage元数据模型	91
D.1 FIMD_COMMENTS	91
D.2 FIMD_EXIF_*	91
D.3 FIMD_IPTC	92
D.4 FIMD_XMP	92
D.5 FIMD_GEOTIFF	92
D.6 FIMD_ANIMATION	92
D.7 FIMD_CUSTOM	92

附录 E FIMD_ANIMATION元数据模型规范

93

表格

2.1	FREE_IMAGE_FORMATS常量(FreeImage格式标识符)	5
2.2	FREE_IMAGE_TYPE常量(FreeImage数据类型标识符)	7
2.3	可选解码常量	8
2.4	可选编码常量	11
2.5	Free_Image_Color_Type常量	15
2.6	像素访问宏及相关掩码	22
2.7	Free_Image_Quantize常量	31
2.8	Free_Image_Dither常量	34
2.9	FreeImage允许的位图类型转换	36
2.10	FREE_IMAGE_TMO常量	37
3.1	FreeImage FITAG结构	62
3.2	FreeImage标签数据类型	63
3.3	FreeImage支持的元数据模型	64
4.1	FREE_IMAGE_JPEG_OPERATION常量	73
4.2	IMAGE_FILTER常量	74
4.3	FREE_IMAGE_COLOR_CHANNEL常量	75
4.4	FreeImage允许的位图文件格式	81

表格

表格

第一章 介绍

1.1 前言

感谢您下载FreeImage这个可用于Windows、Linux和Mac OS X等操作系统的自由和开源图象库。FreeImage因其速度和简洁而得到了广泛的应用和赞誉，至今它已经经历了5年以上的开发。

FreeImage由Floris den berg创建，最初开发它是为了给一个名为Magenta多媒体工具的写作工具提供载入位图功能支持。库的主要成分由Floris开发，但在它的漫长生命里，很多人对FreeImage作出了贡献，他们给FreeImage增添新的特性，并帮助对库进行测试，没有这些人们的帮助，FreeImage就不会有它的今天。无论是任何人，都可以提交他们对FreeImage的改进，并将其插入主源代码中（当然了，条件是FreeImage的开发者们都认为这些改动确实好）。在FreeImage.h头文件里列出的贡献者列表，只不过是那些每天给我们提供bug报告、建议、主意和源代码的人们中的一小部分。

Floris在2000年年中停止了库的开发，从那时候起，FreeImage由Hervé Drolon维护并继续开发。

1.2 译者的话

首先，非常感谢FreeImage开发组和所有对FreeImage开发作出贡献的人们！本文档蒙FreeImage开发组许可翻译成中文发表，原代码采用的是Latex格式，经编译成为pdf文档。由于本人英文水平、计算机图形学专业水平都不高，译文中难免有不通顺、不妥或错误之处，望各位读者朋友不吝指正，多多赐教！文档Latex原代码可向本人索取，联系电话：13873166913，电子邮件：Li_pin_zhong@tom.com.

Great thanks to the FreeImage development team and all those contributed to the FreeImage development!

1.3 FreeImage的目标

一个项目的清晰的目标图是非常重要的，因为它规定项目实现哪些特性，而哪些特性又是它所不提供的。

FreeImage支持：

- 位图构件—例如调色板和数据位—的便易访问；
- 将位图从一种位深度转换到另一种位深度；
- 当有多幅位图页—例如TIFF—时访问位图的不同页；
- 基本的位图处理，如旋转、翻转（flipping）和重采样，或点操作，如亮度和对比度调整；
- Alpha混合与合成（Compositing and blending）；

FreeImage不支持：

- 高级图象处理操作，如回卷（Convolution）和变换（Transform）；
- 位图绘制； 矢量图形。

1.4 库参考

FreeImage中的每一个函数名都被冠以”FreeImage_”，如FreeImage_Load、FreeImage_Save、FreeImage_UnLoad ... 等等。

在位图函数参考、元数据函数参考和工具函数参考这几章中给出了对FreeImage库所支持的每个函数的详细描述，对于每个入口都为C/C++展示了函数原型，并列出了函数的参数和对函数的解释。在这些章节中您将会看到在一些函数的上方的彩色方框内有数字，这些数字指示了函数可以在其上操作的输入图象的像素深度（注：即位深度），对于标准位图，这些像素深度可以是每像素1位、4位、8位、16位、24位或32位，或者对于特殊位图类型，这些像素深度可以是像素16位、32位、48位、64位、96位、128位或2×64位。如果没有显示方框中的数字，那么函数的操作与图象的像素深度无关（例如Load/Save和插件函数）。

第二章 位图函数参考

2.1 通用函数

以下函数对FreeImage提供支持的位图不做任何操作，它们是内部库管理函数，这并不意味着它们不重要，没有它们，您将完全不可能载入任何位图。

FreeImage_Initialise

DLL_API void DLL_CALLCONV FreeImage_Initialise(BOOL load_local_plugins_only, FI_DEFAULT(FALSE));

初始化FreeImage库。当load_local_plugins_only参数为TRUE时，FreeImage将不使用外部插件。

!当使用FreeImage DLL库（作为动态连接库来使用）时，这个函数以参数load_local_plugins_only被设置为FALSE的形式被自动调用。当把FreeImage作为静态连接库来使用时，您必须在程序的开始处唯一地调用一次该函数。

FreeImage_DeInitialise

DLL_API void DLL_CALLCONV FreeImage_DeInitialise();

撤消对FreeImage的初始化。

!当使用FreeImage DLL库（作为动态连接库来使用）时，这个函数被自动调用。当把FreeImage作为静态连接库来使用时，您必须在程序的结束处唯一地调用一次该函数，以清除在FreeImage库中分配的内存。

FreeImage_GetVersion

DLL_API const char *DLL_CALLCONV FreeImage_GetVersion();

返回包含DLL库动态连接库当前版本的一个字符串。

FreeImage_GetCopyrightMessage

DLL_API const char *DLL_CALLCONV FreeImage_GetCopyrightMessage();

返回包含一个标准版权信息的字符串，您可以在程序中展示该信息串。

FreeImage_SetOutputMessage

DLL_API void DLL_CALLCONV FreeImage_SetOutputMessage(FreeImage_OutputMessageFunctionomf);

在不能载入或保存某个位图时，通常对此有一个解释。例如，某种位图格式可能会因为专利限制而不被支持，或对于某种位图子类型存在着一个已知问题。无论何时FreeImage库的内部操作失败后都会产生一个日志字符串，这个字符串可以被操纵FreeImage库的应用程序捕获。您可以用FreeImage_SetOutPutMessage来捕获这个日志串，这样可以把它向程序用户展示。

```
/**
FreeImage 错误处理例程
@param fif Format / Plugin responsible for the error
@param message Error message
**/
void FreeImageErrorHandler(FREE_IMAGE_FORMAT fif, const
    char *message) {
    printf("_n***_n");
    printf("%s_format_n", FreeImage_GetFormatFromFIF(fif));
    printf(message);
    printf("_n***_n");
    //在您的主程序中...
    FreeImage_SetOutputMessage(FreeImageErrorHandler);
}
```

2.2 位图管理函数

FreeImage库中的位图管理函数无疑是最常用的，它让您可以为新位图分配内存、导入位图，这样就可以在内存中编辑它们，并输出到磁盘上。您将会看到，FreeImage位图函数是非常易于使用的。

虽然FreeImage可以处理20种以上的位图类型，但位图处理函数只有4个，一个特殊参数—名为FREE_IMAGE_FORMAT的枚举型常量—被用来指定将要载入或保存的位图的格式，目前，可用的FREE_IMAGE_FORMAT枚举型常量如下表：

表 2.1:FREE_IMAGE_FORMATS常量(FreeImage格式标识符)

FIF	Description
FIF_UNKNOWN	未知格式(仅作返回值用,万勿用作输入)
FIF_BMP	Windows或OS/2 Bitmap文件*.BMP
FIF_CUT	Dr. Halo (*.CUT)
FIF_DDS	DirectDraw Surface (*.DDS)
FIF_GIF	图象交换格式(*.GIF)
FIF_HDR	大动态范围(*.HDR)
FIF_ICO	Windows Icon文件(*.ICO)
FIF_IFF	Amiga IFF文件(*.IFF, *.LBM)
FIF_JNG	JPEG网络图象(*.JNG)
FIF_JPEG	独立JPEG小组(*.JPG, *.JIF, *.JPEG, *.JPE)
FIF_KOALA	Commodore 64 Koala格式(*.KOA)
FIF_MNG	网络多图象文件(*.MNG)
FIF_PBM	便携式位图(ASCII) (*.PBM)
FIF_PBMRAW	便携式位图(BINARY) (*.PBM)
FIF_PCD	柯达PhotoCD格式(*.PCD)
FIF_PCX	Zsoft Paintbrush PCX位图格式(*.PCX)
FIF_PGM	便携式Graymap (ASCII) (*.PGM)
FIF_PGMRAW	便携式Graymap (BINARY) (*.PGM)
FIF_PNG	便携式网络图象(*.PNG)
FIF_PPM	便携式Pixelmap (ASCII) (*.PPM)
FIF_PPMRAW	便携式Pixelmap (BINARY) (*.PPM)
FIF_PSD	Adobe Photoshop (*.PSD)
FIF_RAS	Sun Rasterfile (*.RAS)
FIF_TARGA	Truevision Targa文件(*.TGA, *.TARGA)
FIF_TIFF	Tagged图象文件格式(*.TIF, *.TIFF)
FIF_WBMP	无线位图格式(*.WBMP)
FIF_XBM	X11位图格式(*.XBM)
FIF_XPM	X11Pixmap 格式(*.XPM)

作为对FREE_IMAGE_FORMAT枚举型常量的扩展,您可以注册自己的位图格式。位图注册可以通过调用插件函数之一(参见[插件函数](#))来手工地进行,或者通过将一个预编译的FreeImage位图插件DLL动态连接库复制到FREEIMAGE.DLL库所在目录中来进行。当一种新的位图类型被注册时,它被赋予一个新的、唯一的插件标识号,您可以把它传送到您希望向其传递FREE_IMAGE_FORMAT的同一地方。

FreeImage_Allocate

1 4 8 16 24 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_Allocate(int width,int height,
int bpp,unsigned red_mask FI_DEFAULT(0), unsigned green_mask FI_DEFAULT(0),
unsigned blue_mask FI_DEFAULT(0));
```

如果您希望在内存中从头开始创建一个新位图、而不是从磁盘上载入一个预先制作好的位图，那么使用该函数。FreeImage_Allocate接受一个宽度参数、一个高度参数，还有一个bpp参数，它被用来指定图象的位深度。FreeImage_Allocate返回一个FIBITMAP。最后三个可选参数（redmask、greenmask和bluemask）用来告诉FreeImage图象的颜色分量的位布局，例如在一个像素中红、绿、蓝分量的储存位置。这里给您演示一下如何解释颜色掩码：当 redmask为0xFF000000时意味着在一个像素中的最后八位用于红色分量，当 greenmask为0x000000FF时意味着在一个像素中的最开始八位用于绿色分量。

!FreeImage_Allocate为一个空位图—例如一个完全以零填充的位图—分配内存，位图中的零通常被解释为黑色，这意味着如果您的位图配了色，那么它将包含一个完全为黑色的调色板。您可以访问调色板并因此通过使用FreeImage_GetPallete函数来分布（populate）调色板。

```
FIBITMAP *bitmap = FreeImage_Allocate(320, 240, 32);
if (bitmap) {
    // 位图创建成功!
    FreeImage_Unload(bitmap);
}
```

!FreeImage_Allocate是FreeImage_AllocateT的别称，可用这个调用来代替：FreeImage_AllocateT(FIT_BITMAP, width, height, bpp, red_mask, green_mask, blue_mask);

FreeImage_AllocateT

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_AllocateT(FREE_IMAGE_TYPE
type, int width, int height, int bpp FI_DEFAULT(8), unsigned red_mask FI_DEFAULT(0),
unsigned green_mask FI_DEFAULT(0), unsigned blue_mask FI_DEFAULT(0));
```

一方面，多数图象应用程序只处理摄影图象，而同时另一方面，很多科学应用程序需要处理高分辨率图象（例如16位灰度图象）、真实值像素（real valued pixels）或甚至复像素（例如，想象一下对一个8位灰度图象进行傅立叶变换的结果吧：结果是一个复图象）。

一个特殊参数—名为FREE_IMAGE_TYPE的枚举常量—被用来指定一个FIBITMAP的位图类型，这个枚举常量在头文件FREEIMAGE.H中定义。目前可用的FREE_IMAGE_TYPE常量如下：

表 2.2:FREE_IMAGE_TYPE常量(FreeImage数据类型标识符)

FIT	Description
FIT_UNKNOWN	未知格式(仅作返回值用,万勿用作输入)
FIF_BITMAP	标准图象: 1-, 4-, 8-, 16-, 24-, 32-位
FIT_UINT16	unsigned short类型数组: 16位无符号整数
FIT_UINT32	unsigned long类型数组: 32位无符号整数
FIT_INT32	long类型数组: 32位有符号整数
FIT_FLOAT	float类型数组: 32位IEEE浮点数
FIT_DOUBLE	DOUBLE类型数组: 64位IEEE浮点数
FIT_COMPLEX	FICOMPLEX类型数组: 2 x 64位IEEE浮点数
FIT_RGB16	48位RGB图象: 3 x 16位
FIT_RGBA16	64位RGBA图象: 4 x 16位
FIT_RGBF	96位浮点图象: 3 x 32位IEEE浮点数
FIT_RGBAf	128位RGBA浮点图象: 4 x 32位IEEE浮点数

!当您需要知道一个位图的数据类型时,可以使用函数*FreeImage_GetImageType*。

FreeImage_Load

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_Load(FREE_IMAGE_FORMAT fif,
const char *filename, int flags FI_DEFAULT(0));
```

这个函数对一个位图进行编码,为位图分配内存,并将其作为一个FIBITMAP来返回。第一个参数定义要载入的位图的类型,例如,当传递FIF_BITMAP时,一个BMP文件被载入内存(表 2.1 给出了可用的FREE_IMAGE_FORMAT常量一览表)。第二个参数告诉FreeImage它必须要解码的文件。最后一个参数用来改变函数的行为或激活位图插件中的一种特性,每个插件有它自己的参数集。

```
FIBITMAP *bitmap = FreeImage_Load(FIF_BITMAP, "mybitmap.bmp",
    BMP_DEFAULT);
if (bitmap) {
    // 载入位图成功!
    FreeImage_Unload(bitmap);
}
```

一些位图载入器可以接受改变载入行为的参数,当没有参数或未用到参数时,可以向函数传递0值或<位图类型>_DEFAULT(如BMP_DEFAULT、ICO_DEFAULT等等)。

表 2.3:可选解码常量

位图类型	标志	表述
GIF	GIF_DEFAULT	
	GIF_LOAD256	若图象为2色或16色，则将其作为具有未使用的调色板入口的256色图象来载入
	GIF_PLAYBACK	载入图象时'播放'GIF以产生每一帧(以32bpp的方式),而不是返回原始帧数据
ICO	ICO_MAKEALPHA	载入图象时将其转换为32位，并从AND-掩码中产生一个alpha通道
JPEG	JPEG_DEFAULT	牺牲一些性能以尽可能快地载入文件
	JPEG_FAST	牺牲一些性能以尽可能快地载入文件
	JPEG_ACCURATE	牺牲一些速度性能以最好的质量载入文件
	JPEG_CMYK	该标志会将CMYK位图作为32位CMYK分色位图来载入
PCD	PCD_DEFAULT	一幅PhotoCD图象有很多不同的大小。该标志会载入那些大小为768 x 512的PhotoCD图象
	PCD_BASE	该标志会载入那些大小为768 x 512的PhotoCD图象
	PCD_BASEIV4	该标志会载入那些大小为384 x 256的PhotoCD图象
	PCD_BASEIV16	该标志会载入那些大小为192 x 128的PhotoCD图象
PNG	PNG_IGNOREGAMMA	避免gamma校正
TARGA	TARGA_LOAD_RGB888	若设置了该载入器，则将RGB555和ARGB8888转换为RGB888
TIFF	TIFF_CMYK	该标志会将CMYK位图作为32位CMYK分色位图来载入

FreeImage_LoadU

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_LoadU(FREE_IMAGE_FORMAT
fif, const wchar_t *filename, int flags FI_DEFAULT(0));
```

该函数和*FreeImage_Load*所起作用一样，但它还支持Unicode文件名。请注意该函数只在32位Windows操作系统下起作用，在其他操作系统下函数不做任何事，并返回NULL。

FreeImage_LoadFromHandle

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_LoadFromHandle(FREE_IMAGE_FORMAT fif, FreeImageIO *io, fi_handle handle, int flags FI_DEFAULT(0));

FreeImage具有从任意资源载入一个位图的独特性能，这个资源可以是一个—比方说—打包（Cabinet）文件、Zip压缩文件或互连网上的流文件。在FREEIMAGE.DLL中这些任意的资源并不是直接处理的，但可以通过添加一个FREEIMAGE.H中定义的FreeImageIO结构来轻松地添加这些处理。

FreeImageIO结构是一个包含4个函数指针的结构，这4个函数指针一个用来从位图资源中读取，一个用来将位图写入位图资源，一个用来在资源中定位（seek），而最后一个告诉我们当前处在资源中的位置。当您收集（populate）具有指向函数的指针的FreeImageIO结构并将该结构传递给FreeImage_LoadFromHandle时，FreeImage将会调用您的函数来在文件中读取、定位和给出所处位置。句柄参数（左起第三个参数）在此用来在不同的设备（Context）—例如不同的文件或不同的互连网上的流—中区分它们。

!在FreeImageIO结构中的函数指针使用stdcall调用约定，这意味着它们所指向的函数也必须使用stdcall调用约定。选择这种调用约定是为了与C++以外的语言（例如Visual Basic）的兼容性。

```
FreeImageIO io;
io.read_proc = ReadProc; // 指向调用函数的指针 fread
io.write_proc = NULL;    // 载入时不需要
io.seek_proc = SeekProc; // 指向调用函数的指针 fseek
io.tell_proc = TellProc; // 指向调用函数的指针 ftell
FILE *f = fopen("mybitmap.bmp", "rb");
FIBITMAP *bitmap = FreeImage_LoadFromHandle(FIF_BMP, &io,
    (fi_handle)f, 0);
fclose(f);
if (bitmap) { // 载入位图成功!
    FreeImage_Unload(bitmap); }
```

FreeImage_Save

DLL_API BOOL DLL_CALLCONV FreeImage_Save(FREE_IMAGE_FORMAT fif, FIBITMAP *dib, const char *filename, int flags FI_DEFAULT(0));

该函数把先前载入的FIBITMAP保存到一个文件中，第一个参数定义要保存的位图类型，例如，如果传递FIF_BMP，则会保存为一个BMP文件（可能的FREE_IMAGE_FORMAT常量一览表见表2.1）。第三个参数是要保存的位图名。若文件已存在则会被覆盖。请注意一些位图保存插件对所能保存的位图类型有限制*。例如JPEG插件仅能保存24位和8位灰度位图。最后一个参数用来在位图插件中改变行为或激活一种特性。每个插件有它自己的参数集。

*在JPEG插件中8位配色 (Palletised) 位图在保存时被转换成透明的24位。

```
//本段代码假设已载入一个由叫做"bitmap" 的变量代表的位图
if (FreeImage_Save(FIF_BMP, bitmap, "mybitmap.bmp", 0))
{
    //位图保存成功!
```

一些位图保存器可以接受参数以改变保存时的行为。如果没有这些参数或没有使用这些参数，可以向它传递0值或<位图类型>_DEFAULT (例如BMP_DEFAULT、ICO_DEFAULT等等)。

FreeImage_SaveU

```
DLL_API BOOL DLL_CALLCONV FreeImage_SaveU(FREE_IMAGE_FORMAT fif,
FIBITMAP *dib, const wchar_t *filename, int flags FI_DEFAULT(0));
```

该函数和`FreeImage_Save`所起作用一样，但它还支持UNICODE文件名。请注意该函数只在32位Windows操作系统下起作用，在其他操作系统下它不做任何事，并返回FALSE。

FreeImage_SaveToHandle

```
DLL_API BOOL DLL_CALLCONV FreeImage_SaveToHandle(FREE_IMAGE_FORMAT
fif, FIBITMAP *dib, FreeImageIO *io, fi_handle handle, int flags FI_DEFAULT(0));
```

前面描述的用来从任意一个资源中载入一个位图的FreeImageIO结构同样可以用来保存位图。这里我们再次看到，FreeImage并没有实现保存位图的方法，但它让您通过收集 (populate) 一个具有指向函数的指针的FreeImageIO结构来实现所需要的函数。现在，FreeImage将会调用您的函数来在一个流中写、定位 (seek) 和给出所处位置。

```
//本段代码假设已载入一个由叫做"image" 的变量代表的位图
FreeImageIO io;
io.read_proc = NULL;          // 保存位图时不需要
io.write_proc = WriteProc;    // 指向调用函数的指针 fwrite
io.seek_proc = SeekProc;      // 指向调用函数的指针 fseek
io.tell_proc = TellProc;      // 指向调用函数的指针 ftell
FILE *f = fopen("mybitmap.bmp", "wb");
if (FreeImage_SaveToHandle(FIF_BMP, bitmap, &io, (
    fi_handle)f, 0)){ }
    // 载入位图成功!
    fclose(f);
```

表 2.4:可选编码常量

位图类型	标志	表述
BMP	BMP_DEFAULT	保存时不作任何压缩
	BMP_SAVE_RLE	保存时用RLE压缩来对位图进行压缩
JPEG	JPEG_DEFAULT	以良好品质保存(75:1)
	JPEG_QUALITYSUPERB	以超级(superb)品质保存(100:1)
	JPEG_QUALITYGOOD	以良好品质保存(75:1)
	JPEG_QUALITYNORMAL	以一般品质保存(50:1)
	JPEG_QUALITYAVERAGE	以平均品质保存(25:1)
	JPEG_QUALITYBAD	以较差品质保存(10:1)
	1—100间的整数x	以x:1的品质保存
BMP,PGM,PPM	PNM_DEFAULT	将位图保存为二进制文件
	PNM_SAVE_RAW	将位图保存为二进制文件
	PNM_SAVE_ASCII	将位图保存为ASCII文件
TIFF	TIFF_DEFAULT	用CCITTFAX4压缩来压缩1位位图,用LZW压缩来压缩其他位图
	TIFF_CMYK	为CMYK分色位图储存标签(用I来与TIFF压缩标志组合)
	TIFF_PACKBITS	用PACKBITS压缩来对图象进行压缩后保存
	TIFF_DEFLATE	用DEFLATE压缩(即ZLIB压缩)来对图象进行压缩后保存
	TIFF_ADOBE_DEFLATE	用ADOBE DEFLATE压缩来对图象进行压缩后保存
	TIFF_NONE	不作任何压缩地保存图象
	TIFF_CCITTFAX3	用CCITTFAX Group 3 fax 编码来保存图象
	TIFF_CCITTFAX4	用CCITTFAX Group 4 fax 编码来保存图象
	TIFF_LZW	用LZW压缩来对图象进行压缩后保存
	TIFF_JPEG	用JPEG压缩来对图象进行压缩后保存(仅用于8位灰度图象和24位图象。对于其他位深度的图象默认用LZW压缩)

FreeImage_Clone

```
DLL_API FIBITMAP * DLL_CALLCONV FreeImage_Clone(FIBITMAP *dib);
```

为一个现有位图制作一个完全一样的克隆。

```
// 本段代码假设已载入一个由叫做"dib"的变量代表的位图
FIBITMAP *clone = FreeImage_Clone(dib);
if (clone) {
    // 克隆成功!
    FreeImage_Unload(clone);
}
```

FreeImage_Unload

DLL_API void DLL_CALLCONV FreeImage_Unload(FIBITMAP *dib);

从内存中删除一个先前载入的FIBITMAP。

!一旦完成对一个位图的处理,您总需要调用该函数,否则将会有内存泄漏。

2.3 位图信息函数

一旦一个位图被载入内存,您可以从中获取位图的所有信息,或访问位图的指定部分,例如像素位和调色板。

FreeImage_GetImageType

DLL_API FREE_IMAGE_TYPE DLL_CALLCONV FreeImage_GetImageType

(FIBITMAP *dib);

返回一个位图的数据类型(见表 2.2)。

FreeImage_GetColorsUsed

1 4 8 16 24 32

DLL_API unsigned DLL_CALLCONV FreeImage_GetColorsUsed(FIBITMAP *dib);

返回在一个位图中使用的颜色数。对于配色位图,该函数返回调色板大小,而对于多色位图(high-colour bitmap)返回0。

?关于这个函数存在一些考证(criticism)。一些用户预期这个函数返回一个位图中实际使用的颜色数,而实际上函数返回的是调色板大小。这个函数名起源于在BITMAPINFOHEADER中一个名为biClrUsed的成员,这个函数实际上返回的是该内容。

FreeImage_GetBPP

DLL_API unsigned DLL_CALLCONV FreeImage_GetBPP(FIBITMAP *dib);

以位为单位返回位图中每点的大小。例如当每个点在位图中占32位空间时，该函数返回32。对于标准位图，可能的位深度值有1、4、8、16、24和32位，而对于非标准位图，可能的位深度值有16、32、48、64、96和128位。

FreeImage_GetWidth

DLL_API unsigned DLL_CALLCONV FreeImage_GetWidth(FIBITMAP *dib);

以像素为单位返回位图宽度。

FreeImage_GetHeight

DLL_API unsigned DLL_CALLCONV FreeImage_GetHeight(FIBITMAP *dib);

以像素为单位返回位图高度。

FreeImage_GetLine

DLL_API unsigned DLL_CALLCONV FreeImage_GetLine(FIBITMAP *dib);

以字节为单位返回位图宽度。参见：*FreeImage_GetPitch*

?关于这个函数存在一些考证 (criticism)。一些用户预期这个函数返回像素数据中的一个扫描行，而实际上它返回的是以字节为单位的位图宽度。据我所知，“行”这个项是普遍用于以字节为单位的位图宽度的技术，至少它用在了微软的DirectX中。

FreeImage_GetPitch

DLL_API unsigned DLL_CALLCONV FreeImage_GetPitch(FIBITMAP *dib);

以字节为单位返回对齐到下一个**32位字节边界**(因为考虑性能的原因)的位图宽度，又称位深度或线宽度(stride)或扫描宽度。

!在FreeImage中，因为考虑性能的原因每个扫描行开始于一个**32位字节边界**。这在使用低层像素处理函数时(参见**像素访问函数**一节) 在根本上(essential)提高了速度。

FreeImage_GetDIBSize

DLL_API unsigned DLL_CALLCONV FreeImage_GetDIBSize(FIBITMAP *dib);

返回内存中一个FIBITMAP的DIB分量的大小，即BITMAPINFOHEADER+调色板+数据位(注意这不是FIBITMAP的真正大小，仅是其DIB分量的大小)。

FreeImage_GetPalette

1 4 8 16 24 32

DLL_API RGBQUAD *DLL_CALLCONV FreeImage_GetPalette(FIBITMAP *dib);

返回指向位图调色板的指针。如果位图没有调色板(即当像素的位深度大于8时)，该函数返回NULL。


```
//本段代码假设已载入一个由叫做"dib"的变量代表的位图
if (FreeImage_GetBPP(dib) == 8) {
    // 构造一个灰度调色板 Build a greyscale palette
    RGBQUAD *pal = FreeImage_GetPalette(dib);
    for (int i = 0; i < 256; i++) {
        pal[i].rgbRed = i;
        pal[i].rgbGreen = i;
        pal[i].rgbBlue = i;}
}
```

FreeImage_GetDotsPerMeterX

DLL_API unsigned DLL_CALLCONV FreeImage_GetDotsPerMeterX(FIBITMAP *dib);
以点/米为单位返回位图的目标设备的水平分辨率。

FreeImage_GetDotsPerMeterY

DLL_API unsigned DLL_CALLCONV FreeImage_GetDotsPerMeterY(FIBITMAP *dib);
以点/米为单位返回位图的目标设备的垂直分辨率。

FreeImage_SetDotsPerMeterX

DLL_API void DLL_CALLCONV FreeImage_SetDotsPerMeterX(FIBITMAP *dib,
unsigned res);
以点/米为单位设置位图的目标设备的水平分辨率。

FreeImage_SetDotsPerMeterY

DLL_API void DLL_CALLCONV FreeImage_SetDotsPerMeterY(FIBITMAP *dib,
unsigned res);
以点/米为单位设置位图的目标设备的垂直分辨率。

FreeImage_GetInfoHeader

1 4 8 16 24 32

DLL_API BITMAPINFOHEADER *DLL_CALLCONV FreeImage_GetInfoHeader
(FIBITMAP *dib);
返回指向一个FIBITMAP中的DIB分量的BITMAPINFOHEADER结构的
指针。

FreeImage_GetInfo

1 4 8 16 24 32

DLL_API BITMAPINFO *DLL_CALLCONV FreeImage_GetInfo(FIBITMAP *dib);
等同于FreeImage_GetInfoHeader，但返回的是指向一个BITMAPINFO结构的指针，而不是指向一个BITMAPINFOHEADER结构的指针

FreeImage_GetColorType

1 4 8 16 24 32

DLL_API FREE_IMAGE_COLOR_TYPE DLL_CALLCONV FreeImage_GetColorType(FIBITMAP *dib);
通过读取位图的像素位来调查位图的色彩类型，并对其加以分析。FreeImage_GetColorType可以返回以下值：

表 2.5:Free_Image_Color_Type常量

值	表述
FIC_MINISBLACK	黑白位图(1位)：第一个调色板入口是黑色。配色位图(4或8位)：位图具有一个灰度调色板
FIC_MINISWHITE	黑白位图(1位)：第一个调色板入口是白色。配色位图(4或8位)：位图具有一个反转灰度调色板
FIC_PALETTE	配色位图(1位、4位或8位)
FIC_RGB	多色位图(16位、24位或32位)
FIC_RGBALPHA	带一个alpha通道的多色位图(仅适用于32位)
FIC_CMYK	CMYK位图(仅适用于32位)

?一个位图要被判定为灰度位图，它必须具有一个带这些特征的调色板：
-每个调色板入口的红色、绿色、蓝色的值必须相等，
-相邻的调色板入口之间的间隔值必须为正并且等于1。
?如果需要用于印刷业或出版业的一幅图象，则推荐用CMYK色彩模型(即FIC.CMYK)。在几乎所有的情况下，这种工作都是由图象艺术家来完成的：他们照一张RGB的相(例如用数码照相机照相)并校正颜色值作为图象的近似(单个像素，亮度，对比度...)。最后他们输出一幅CMYK分色相片，这张相片将会直接送到一个排版程序中，然后输出到打印机器。大多数FreeImage用户将永远不会需要使用CMYK分色图象，因为打印机驱动程序将会做转换工作。但在专业打印中，样张(proofed)转换对于获得清晰的打印结果来说是非常重要的(此时可没有驱动程序来做类似转换的工作)。这就是为什么在某些杂志上印刷的图象比我们在家里制作的打印图象好看得多的原因。

FreeImage_GetRedMask

1 4 8 16 24 32

DLL_API unsigned DLL_CALLCONV FreeImage_GetRedMask(FIBITMAP *dib);

返回一个位模式(bit pattern)来描述一个FIBITMAP结构中的一个像素的红色分量。

FreeImage_GetGreenMask

1 4 8 16 24 32

DLL_API unsigned DLL_CALLCONV FreeImage_GetGreenMask(FIBITMAP *dib);

返回一个位模式(bit pattern)来描述一个FIBITMAP结构中的一个像素的绿色分量。

FreeImage_GetBlueMask

1 4 8 16 24 32

DLL_API unsigned DLL_CALLCONV FreeImage_GetBlueMask(FIBITMAP *dib);

返回一个位模式(bit pattern)来描述一个FIBITMAP结构中的一个像素的蓝色分量。

```
//本段代码假设已载入一个由叫做" dib " 的变量代表的位图
unsigned red_mask , green_mask , blue_mask ;
red_mask = FreeImage_GetRedMask ( dib ) ;
green_mask = FreeImage_GetGreenMask ( dib ) ;
blue_mask = FreeImage_GetBlueMask ( dib ) ;
if (FreeImage_GetBPP ( dib ) == 16) {
    if (( red_mask == FI16_565_RED_MASK ) && ( green_mask ==
        FI16_565_GREEN_MASK ) && ( blue_mask ==
        FI16_565_BLUE_MASK )) {
        //已处于RGB16 565 模式}
    else { // 已处于RGB16 555 模式}
}
}
```

FreeImage_GetTransparencyCount

1 4 8 16 24 32

DLL_API unsigned DLL_CALLCONV FreeImage_GetTransparencyCount(FIBITMAP *dib);

返回配色位图中透明色的数目。FreeImage_GetTransparencyCount对于非配色位图的情况总是返回0。

FreeImage_GetTransparencyTable

8

DLL_API BYTE * DLL_CALLCONV FreeImage_GetTransparencyTable(FIBITMAP *dib);
 返回一个指向位图透明表的指针。只有配色位图才有透明表。多色位图直接把透明表储存在位图的位中, 对于这些位图, FreeImage_GetTransparencyTable返回NULL。

FreeImage_SetTransparencyTable

8

DLL_API void DLL_CALLCONV FreeImage_SetTransparencyTable(FIBITMAP *dib, BYTE *table, int count);
 设置位图的透明表。只有配色位图才有透明表。多色位图直接把透明表储存在位图的位中, 对于这些位图, FreeImage_SetTransparencyTable不做任何操作。

```
\#include "FreeImage.h"
int main(int argc, char* argv[]) {
    FIBITMAP *hDIB24bpp = FreeImage_Load(FIF_BMP, "test.bmp", 0);
    if (hDIB24bpp) { //颜色量子化 24bpp (形成8bpp 来设置透明度)
        FIBITMAP *hDIB8bpp = FreeImage_ColorQuantize(
            hDIB24bpp, FIQ_WUQUANT);
        // 获取调色板并找到亮绿色
        RGBQUAD *Palette = FreeImage_GetPalette(hDIB8bpp);
        BYTE Transparency[256];
        for (unsigned i = 0; i < 256; i++) {
            Transparency[i] = 0xFF;
            if (Palette[i].rgbGreen >= 0xFE && Palette[i].
                rgbBlue == 0x00 && Palette[i].rgbRed == 0x00) {
                Transparency[i] = 0x00;}}
        // 设置透明表
        FreeImage_SetTransparencyTable(hDIB8bpp, Transparency,
            256);
        // 将8bpp 图象保存为透明PNG
        FreeImage_Save(FIF_PNG, hDIB8bpp, "test.png", 0);
        FreeImage_Unload(hDIB24bpp);
        FreeImage_Unload(hDIB8bpp);}
return 0;}
```

FreeImage_SetTransparent

8 32

DLL_API void DLL_CALLCONV FreeImage_SetTransparent(FIBITMAP *dib, BOOL enabled);

告诉FreeImage是否应该使用可能随位图携带的透明表。当调用这个函数来处理一个其位深度不同于8位或32位的位图时，无论布尔型变量enabled取何值，透明性都被禁止。

FreeImage_IsTransparent

1 4 8 16 24 32

DLL_API BOOL DLL_CALLCONV FreeImage_IsTransparent(FIBITMAP *dib);

当透明表被激活(8位图象)或输入的dib包含alpha通道值(32位图象)时返回TRUE，其他情况下返回FALSE。

FreeImage_HasBackgroundColor

8 24 32

DLL_API BOOL DLL_CALLCONV FreeImage_HasBackgroundColor(FIBITMAP *dib);

当图象具有文件背景色时返回TRUE，否则返回FALSE。

FreeImage_GetBackgroundColor

8 24 32

DLL_API BOOL DLL_CALLCONV FreeImage_GetBackgroundColor(FIBITMAP *dib, RGBQUAD *bkcolor);

获取一幅图象的文件背景色，若成功则返回TRUE，否则返回FALSE。对于8位图象，调色板中的颜色索引号被返回到bkcolor参数的rgbReserved成员中。

FreeImage_SetBackgroundColor

8 24 32

DLL_API BOOL DLL_CALLCONV FreeImage_SetBackgroundColor(FIBITMAP *dib, RGBQUAD *bkcolor);

设置一幅图象的文件背景色。当将一幅图象保存为PNG文件时，这个背景色被透明地保存到PNG文件中。

当参数bkcolor为NULL时，背景色从图象中被删除。

2.4 文件类型函数

以下函数通过从一个位图中读取最多达16字节来获取Free_Image.FORMAT,并对其进行分析。

请注意,对于某些位图而言,并无Free_Image.FORMAT可供读取,这时只能通过位图类型的位布局来获取位图格式,而这有时候与FreeImage的文件类型感知系统不兼容。不能识别的格式有: CUT、MNG、PCD、TARGA和WBMP。不过可以通过使用*FreeImage_GetFIFFromFilename*函数来识别这些格式。

FreeImage_GetFileType

```
DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_GetFileType(const char
*filename, int size FI_DEFAULT(0));
```

指令FreeImage分析位图的签名。然后函数返回预定义的Free_Image.FORMAT常量之一,或返回一个由插件注册的位图标识符数字。size参数目前没有用到,可将其设为0。

!因为并非所有格式都可由其文件头来识别(一些图象并没有文件头或文件头位于文件末),若存在一个供分析文件用的插件,则FreeImage_GetFileType可以返回FIF_UNKNOWN。在此情况下可用*FreeImage_GetFIFFromFilename*来从文件的扩展名猜测文件格式,但后者比较慢,而且也不太准确。

```

/** 通用图象载入器
@param lpszPathName 指向文件全名的指针
@param flag 可选的载入标志常量
@return 若成功, 返回载入的 dib, 否则返回NULL
**/
FIBITMAP* GenericLoader(const char* lpszPathName, int
    flag) {
    FREE_IMAGE_FORMAT fif = FIF_UNKNOWN;
    // 检查文件签名并推断其格式
    // 目前(没有用到第二个参数FreeImage)
    fif = FreeImage_GetFileType(lpszPathName, 0);
    if(fif == FIF_UNKNOWN) {
        // 没有签名(signature) ?
        // 试着从文件扩展名来猜测文件格式
        fif = FreeImage_GetFIFFromFilename(lpszPathName);
    }
    // 检查插件是否具有读取该格式的能力...
    if((fif != FIF_UNKNOWN) && FreeImage_FIFSupportsReading
        (fif)) {
        // 好, 让我们来载入文件
        FIBITMAP *dib = FreeImage_Load(fif, lpszPathName,
            flag);
        // 除非是坏的文件格式, 不然的话就大功告成了!
        return dib;
    }
    return NULL;
}

```

FreeImage_GetFileTypeU

DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_GetFileTypeU(const
wchar_t *filename, int size FI_DEFAULT(0));

该函数和`FreeImage_GetFileType`所起作用一样, 但它还支持Unicode文件名。请注意该函数只在32位Windows操作系统下起作用, 在其他操作系统下函数不做任何事, 并返回FIF_UNKNOWN。

FreeImage_GetFileTypeFromHandle

DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_GetFileTypeFromHandle
(FreeImageIO *io, fi_handle handle, int size FI_DEFAULT(0));
用位图管理函数一节中描述的FreeImageIO结构来标识一个位图类型。现在，可以从任意一个位置获取位图的位了。

FreeImage_GetFileTypeFromMemory

DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_GetFileTypeFromMemory
(FIMEMORY *stream, int size FI_DEFAULT(0));
用一个内存中的句柄来标识一个位图类型。可以从任意一个位置获取位图的位(关于内存句柄的更多信息请参见关于内存中的I/O流一节)。

2.5 像素访问函数

像素访问函数为您提供了一个便易的方法，以对FIBITMAP数据进行逐个像素读写并处理。

FreeImage不仅能对标准位图数据(例如1位、4位、8位、16位、24位和32位)进行处理，而且还可以处理诸如16位灰度图象的科学数据或由long类型、double类型或complex类型数值组成的图象(这些图象经常用于信号和图象处理算法中)。表 2.2 中给出了被支持的数据类型总览。

!在FreeImage中，FIBITMAP是基于这样一个坐标系统——它相对于通常约定俗成的图象坐标的右下方(upside down)——的。因此，扫描行的储存是从右下方开始的，即内存中的第一扫描行是图象中最下面的扫描行。(这段话可能翻译得不对或不准确，请参考英文原著——译者注)

位格式

在FIBITMAP中，位的格式是由像素的位深度来定义的，该位深度可以通过FreeImage_GetBPP调用来读取(参见FreeImage_GetImageType)。可能的位深度包括1位、4位、8位、16位、24位、32位、48位、64位、96位和128位。所有格式都共同遵守以下规则：

- 每个扫描行以双字对齐，扫描行以双字对齐方式被缓冲；缓冲区被设为0。
- 每个扫描行以从右下方开始的方式储存，内存中的第一扫描行是图象中最下面的扫描行。

每种格式的特征如下：

- 1位DIB是通过以其每一位作为颜色表中的索引的方式来储存的，最重要(significant)的位是最左边的像素。

- 4位DIB是通过以其每四位代表颜色表中的索引的方式来储存的，最重要的末尾(nibble)是最左边的像素。
- 8位DIB是最易于储存的，因为每一字节是颜色表中的一个索引。
- 24位DIB用每三个字节来代表一种颜色，用的顺序与RGBTRIPLE结构的顺序相同。
- 32位DIB用每四个字节来代表一种与一个alpha通道值(用于指示透明性)相关联的颜色，用的顺序与RGBQUAD结构的顺序相同。
- 诸如short类型、long类型、float类型或double类型的非标准位图不具有颜色表，像素的储存方式类似于8位DIB。
- 复图象类型以类似于24位或32位DIB的储存方式来储存像素，用的顺序与FICOMPLEX结构的顺序相同。
- 16位RGB[A]或float类型RGB[A]图象类型以类似于24位或32位DIB的储存方式来储存像素，用的顺序与FIRGB[A]16或FIRGB[A]F结构的顺序相同。

色彩模型

色彩模型是一种摘要机制模型，用来描述可以用数字tuples—特别是用三个或四个颜色分量值(例如RGB和CMYK都是色彩模型)—来表示颜色的方式。FreeImage主要使用RGB[A]色彩模型来表示内存中的像素。

然而，由这种模型使用的像素布局是依赖于操作系统的。通过使用一种逐个字节顺序排列的方式来标示出像素布局，则FreeImage在Little Endian处理器中使用BGR[A]像素布局(Windows和Linux)，而在Big Endian处理器中使用RGB[A]像素布局(Mac OS或任何Big Endian的Linux/Unix)。作出这种选择是为了便易地使用FreeImage图形图象API。(译者注：Little Endian和Big Endian是多字节数据结构在内存中的字节排序的两种方式，前者表示低位字节在内存低端地址，高位字节在内存高端地址，后者恰好相反。)

无论如何这种微妙的差异还是留给了用户。为了使像素访问与操作系统无关，FreeImage定义了一个宏集，用来在24位或32位DIB中设置或获取单独的颜色分量。

表 2.6:像素访问宏及相关掩码

通道	像素位置	相关掩码
红色	FI_RGBA_RED	FI_RGBA_RED_MASK
绿色	FI_RGBA_GREEN	FI_RGBA_GREEN_MASK
蓝色	FI_RGBA_BLUE	FI_RGBA_BLUE_MASK
Alpha	FI_RGBA_ALPHA	FI_RGBA_ALPHA_MASK

!当访问一个24位或32位DIB的单个颜色分量时,您永远应该使用FreeImage宏或RGBTRIPLE/RGBQUAD结构,以写出与操作系统平台无关的代码。

以下例子展示了在处理一个32位DIB时如何使用这些宏:

```
// 给一个32 位 dib 分配内存
FIBITMAP *dib = FreeImage_Allocate(512, 512, 32,
    FLRGBA_RED_MASK, FLRGBA_GREEN_MASK,
    FLRGBA_BLUE_MASK);
// 计算每像素的字节数(24 位像素为3 字节而32 位像素为4 字节)
int bytespp = FreeImage_GetLine(dib) / FreeImage_GetWidth
    (dib);
for(unsigned y = 0; y < FreeImage_GetHeight(dib); y++) {
    BYTE *bits = FreeImage_GetScanLine(dib, y);
    for(unsigned x = 0; x < FreeImage_GetWidth(dib); x++) {
        // 设置像素点颜色为绿色, 透明度为128
        bits[FLRGBA_RED] = 0;
        bits[FLRGBA_GREEN] = 255;
        bits[FLRGBA_BLUE] = 0;
        bits[FLRGBA_ALPHA] = 128;
        // 跳转到下一个像素
        bits += bytespp;
    }
}
```

FreeImage_GetBits

DLL_API BYTE *DLL_CALLCONV FreeImage_GetBits(FIBITMAP *dib);

返回一个指向位图的数据位的指针。根据调用FreeImage_GetBPP、FreeImage_GetRedMask、FreeImage_GetGreenMask、FreeImage_GetBuleMask的不同结果,用户自己已经合适于完成正确地解释这些字节的任务。

?出于性能的考虑,由FreeImage_GetBits返回的地址是以对齐于16字节边界的方式来对齐的。

```
// 本段代码假设已载入一个由叫做"dib"的变量代表的位图
unsigned width = FreeImage_GetWidth(dib);
unsigned height = FreeImage_GetHeight(dib);
unsigned pitch = FreeImage_GetPitch(dib);
FREEIMAGE_TYPE image_type = FreeImage_GetImageType(dib);
// 测试对像素的访问, 避免扫描行的计算以加速图象处理
if(image_type == FIT_RGBF) {
    BYTE *bits = (BYTE*)FreeImage_GetBits(dib);
    for(y = 0; y < height; y++) {
        FIRGBF *pixel = (FIRGBF*)bits;
        for(x = 0; x < width; x++) {
            pixel[x].red = 128;
            pixel[x].green = 128;
            pixel[x].blue = 128;
        }
        // 下一行
        bits += pitch;
    }
}
else if((image_type == FIT_BITMAP) && (FreeImage_GetBPP(
    dib) == 24)) {
    BYTE *bits = (BYTE*)FreeImage_GetBits(dib);
    for(y = 0; y < height; y++) {
        BYTE *pixel = (BYTE*)bits;
        for(x = 0; x < width; x++) {
            pixel[FLRGBA_RED] = 128;
            pixel[FLRGBA_GREEN] = 128;
            pixel[FLRGBA_BLUE] = 128;
            pixel += 3;
        }
        // 下一行
        bits += pitch;
    }
}
```

FreeImage_GetScanLine

DLL_API BYTE *DLL_CALLCONV FreeImage_GetScanLine(FIBITMAP *dib, int scan-
line);

返回一个指向位图数据位中给定扫描行行首的指针。

根据调用**FreeImage_GetBPP**、**FreeImage_GetImageType**(参见以下例子)的不同结果, 用户自己已经合适于完成正确地解释这些字节的任务。

```
// 本段代码假设已载入一个由叫做"image" 的变量代表的位图
unsigned x, y;
FREEIMAGE_TYPE image_type = FreeImage_GetImageType(image
);
// 测试对像素点的访问
switch(image_type) {
    case FIT_BITMAP:
        if(FreeImage_GetBPP(image) == 8) {
            for(y = 0; y < FreeImage_GetHeight(image); y++) {
                BYTE *bits = (BYTE *)FreeImage_GetScanLine(image,
                    y);
                for(x = 0; x < FreeImage_GetWidth(image); x++) {
                    bits[x] = 128;
                }
            }
        }
        break;
    case FIT_UINT16:
        for(y = 0; y < FreeImage_GetHeight(image); y++) {
            unsigned short *bits = (unsigned short *)
                FreeImage_GetScanLine(image, y);
            for(x = 0; x < FreeImage_GetWidth(image); x++) {
                bits[x] = 128;
            }
        }
        break;
    case FIT_INT16:
        for(y = 0; y < FreeImage_GetHeight(image); y++) {
            short *bits = (short *)FreeImage_GetScanLine(image,
                y);
            for(x = 0; x < FreeImage_GetWidth(image); x++) {
                bits[x] = 128;
            }
        }
        break;
}
```

```
    }  
  }  
  break;  
case FIT_UINT32:  
  for(y = 0; y < FreeImage_GetHeight(image); y++) {  
    unsigned long *bits = (unsigned long *)  
      FreeImage_GetScanLine(image, y);  
    for(x = 0; x < FreeImage_GetWidth(image); x++) {  
      bits[x] = 128;  
    }  
  }  
  break;  
case FIT_INT32:  
  for(y = 0; y < FreeImage_GetHeight(image); y++) {  
    long *bits = (long *)FreeImage_GetScanLine(image, y  
    );  
    for(x = 0; x < FreeImage_GetWidth(image); x++) {  
      bits[x] = 128;  
    }  
  }  
  break;  
case FIT_FLOAT:  
  for(y = 0; y < FreeImage_GetHeight(image); y++) {  
    float *bits = (float *)FreeImage_GetScanLine(image,  
    y);  
    for(x = 0; x < FreeImage_GetWidth(image); x++) {  
      bits[x] = 128;  
    }  
  }  
  break;  
case FIT_DOUBLE:  
  for(y = 0; y < FreeImage_GetHeight(image); y++) {  
    double *bits = (double *)FreeImage_GetScanLine(  
    image, y);  
    for(x = 0; x < FreeImage_GetWidth(image); x++) {  
      bits[x] = 128;  
    }  
  }
```

```
    }  
    break;  
case FIT_COMPLEX:  
    for(y = 0; y < FreeImage_GetHeight(image); y++) {  
        FICOMPLEX *bits = (FICOMPLEX *)  
            FreeImage_GetScanLine(image, y);  
        for(x = 0; x < FreeImage_GetWidth(image); x++) {  
            bits[x].r = 128;  
            bits[x].i = 128;  
        }  
    }  
    break;  
case FIT_RGB16:  
    for(y = 0; y < FreeImage_GetHeight(image); y++) {  
        FIRGB16 *bits = (FIRGB16 *)FreeImage_GetScanLine(  
            image, y);  
        for(x = 0; x < FreeImage_GetWidth(image); x++) {  
            bits[x].red = 128;  
            bits[x].green = 128;  
            bits[x].blue = 128;  
        }  
    }  
    break;  
case FIT_RGBF:  
    for(y = 0; y < FreeImage_GetHeight(image); y++) {  
        FIRGBF *bits = (FIRGBF *)FreeImage_GetScanLine(  
            image, y);  
        for(x = 0; x < FreeImage_GetWidth(image); x++) {  
            bits[x].red = 128;  
            bits[x].green = 128;  
            bits[x].blue = 128;  
        }  
    }  
    break;
```

```

case FIT_RGBA16:
    for(y = 0; y < FreeImage_GetHeight(image); y++) {
        FIRGBA16 *bits = (FIRGBA16 *)FreeImage_GetScanLine(
            image, y);
        for(x = 0; x < FreeImage_GetWidth(image); x++) {
            bits[x].red = 128;
            bits[x].green = 128;
            bits[x].blue = 128;
            bits[x].alpha = 128;
        }
    }
    break;
case FIT_RGBAF:
    for(y = 0; y < FreeImage_GetHeight(image); y++) {
        FIRGBAF *bits = (FIRGBAF *)FreeImage_GetScanLine(
            image, y);
        for(x = 0; x < FreeImage_GetWidth(image); x++) {
            bits[x].red = 128;
            bits[x].green = 128;
            bits[x].blue = 128;
            bits[x].alpha = 128;
        }
    }
    break;
}

```

FreeImage_GetPixelFormat

1 4 8

DLL_API BOOL DLL_CALLCONV FreeImage_GetPixelFormat(FIBITMAP *dib, unsigned x, unsigned y, BYTE *value);

获取一幅配色图象在位置(x,y)处的像素索引，包括范围检查(会降低访问速度)。参数x是像素点在水平向的位置，而y是像素点在垂直向的位置。函数调用成功时返回TRUE，否则返回FALSE(例如对于RGB[A]图象)。

FreeImage_GetPixelColor

16 24 32

DLL_API BOOL DLL_CALLCONV FreeImage_GetPixelColor(FIBITMAP *dib, unsigned x, unsigned y, RGBQUAD *value);

获取一幅16位、24位或32位图象在位置(x,y)处的像素颜色，包括范围检查(会降低访问速度)。参数x是像素点在水平向的位置，而y是像素点在垂直向的位置。函数调用成功时返回TRUE，否则返回FALSE(例如对于配色图象)。

FreeImage_SetPixelIndex

1 4 8

DLL_API BOOL DLL_CALLCONV FreeImage_SetPixelIndex(FIBITMAP *dib, unsigned x, unsigned y, BYTE *value);

设置一幅配色图象在位置(x,y)处的像素索引，包括范围检查(会降低访问速度)。参数x是像素点在水平向的位置，而y是像素点在垂直向的位置。函数调用成功时返回TRUE，否则返回FALSE(例如对于RGB[A]图象)。

FreeImage_SetPixelColor

16 24 32

DLL_API BOOL DLL_CALLCONV FreeImage_SetPixelColor(FIBITMAP *dib, unsigned x, unsigned y, RGBQUAD *value);

设置一幅16位、24位或32位图象在位置(x,y)处的像素颜色，包括范围检查(会降低访问速度)。参数x是像素点在水平向的位置，而y是像素点在垂直向的位置。函数调用成功时返回TRUE，否则返回FALSE(例如对于配色图象)。

2.6 转换函数

下面的函数使得将一个位图从一种位深度转换到另一种位深度成为可能。

在一个Little Endian操作系统下(PC机上的Windows和Linux)，位图在内存中总是以先蓝色、其次绿色、再其次红色最后alpha((BGR[A]约定)的方式储存的。在一个Big Endian操作系统下，FreeImage使用RGB[A]约定。然而这些可移植性的考虑是由转换函数透明地处理的，因此接下来您可以以操作系统平台无关的方式来保存这些转换后的位图。

FreeImage_ConvertTo4Bits

1 4 8 16 24 32

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertTo4Bits(FIBITMAP *dib);

将一个位图转换为4位位图。如果位图为多色位图(16位、24位或32位)或黑白位图或灰度位图(1位或8位)，最后的结果将会是一个灰度位图，否则(1位配色位图)它将会是一个配色位图。返回的是输入位图的一个4位位图克隆。

FreeImage_ConvertTo8Bits

1 4 8 16 24 32

16

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertTo8Bits(FIBITMAP *dib);

将一个位图转换为4位位图。如果位图为多色位图(16位、24位或32位)或黑白位图或灰度位图(1位或4位)，最后的结果将会是一个灰度位图，否则(1位或4位配色位图)它将会是一个配色位图。返回的是输入位图的一个8位位图克隆。

?当创建灰度调色板时，结果像素的灰度幅值是基于相应源像素的红色、绿色、蓝色水平的，用的是以下公式：

$$\text{grey} = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

0.299、0.587和0.114这三个值代表相应的红色、绿色、蓝色的浓度。

对于16位灰度图象(类型为FIT_UINT16的图象)，转换是通过将16位通道除以256来完成的(参见[FreeImage_ConvertToStandardType](#))。对于其他非标准位图类型，返回的是NULL。

FreeImage_ConvertToGreyscale

1 4 8 16 24 32

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertToGreyscale(FIBITMAP *dib);

用线性斜率(linear ramp)将一个位图转换为8位灰度图象。与FreeImage_ConvertTo8Bits函数相反，1位、4位和8位配色图象如同具有FIC_MINIWHITE色彩类型的图象一样被正确地转换。

FreeImage_ConvertTo16Bits555

1 4 8 16 24 32

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertTo16Bits555(FIBITMAP *dib);

将一个位图转换为16位，其中每个像素具有一个5位红色、5位绿色、5位蓝色的色彩图案，有1位没有用到。

返回的是输入位图的一个16位555位图克隆(译者注：555代表上述色彩图案中5位红色、5位绿色、5位蓝色的位布局)。

FreeImage_ConvertTo16Bits565

1 4 8 16 24 32

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertTo16Bits565(FIBITMAP *dib);
将一个位图转换为16位，其中每个像素具有一个5位红色、6位绿色、5位蓝色的色彩图案。

返回的是输入位图的一个16位565位图克隆(译者注：565代表上述色彩图案中5位红色、6位绿色、5位蓝色的位布局)。

FreeImage_ConvertTo24Bits

1 4 8 16 24 32

48

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertTo24Bits(FIBITMAP *dib);
将一个位图转换为24位。返回的是输入位图的一个24位位图克隆。
对于48位RGB图象，转换是通过将16位通道除以256来完成的。对于其他非标准位图类型，返回的是NULL。

FreeImage_ConvertTo32Bits

1 4 8 16 24 32

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertTo32Bits(FIBITMAP *dib);
将一个位图转换为32位。返回的是输入位图的一个32位位图克隆。

FreeImage_ColorQuantize

24

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ColorQuantize(FIBITMAP *dib, FREE_IMAGE_QUANTIZE quantize);
将一个24位多色位图量子化为8位配色位图。量子化参数指定了要用到的颜色消减算法(color reduction algorithm):

表 2.7:Free_Image_Quantize常量

参数	量子化方法
FIQ_WUQUANT	Xiaolin Wu 颜色量子化算法
FIQ_NNQUANT	ANthony Dekker的NeuQuant neural-net量子化算法

参考文献

Wu, Xiaolin, Efficient Statistical Computations for Optimal Color Quantization. In Graphics Gems, vol. II, p. 126-133. [Online]

<http://www.ece.mcmaster.ca/~xwu/>

Dekker A. H., Kohonen neural networks for optimal color quantization. Network: Computation in Neural Systems, Volume 5, Number 3, Institute of Physics Publishing, 1994. [Online]

<http://members.ozemail.com.au/~dekker/NEUQUANT.HTML>

FreeImage_ColorQuantizeEx

24

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ColorQuantizeEx(FIBITMAP *dib,  
FREE_IMAGE_QUANTIZE quantize FI_DEFAULT(FIQ_WUQUANT), int PaletteSize FI_DEFAULT(256),  
int ReserveSize FI_DEFAULT(0), RGBQUAD *ReservePalette FI_DEFAULT(NULL));
```

FreeImage_ColorQuantizeEx是对`FreeImage_ColorQuantize`函数的一个扩展，它提供了额外的选项用来将一个24位图象量子化为任意颜色数(最多可达256)，并且还可以用部分或全部调色板来量子化一个24位图象。

*PaletteSize*参数是想得到的输出调色板的大小。*ReserveSize*是给定的调色板的大小，它由输入数组*ReservePalette*给出。

```

// 本段代码假设已载入一个由叫做"dib"的变量代表的24 位位图
RGBQUAD web_palette[216];
// 216 "web-safe" 颜色列表 colors (RGB 增量 51)
// ...
// 用用户提供的调色板来执行颜色量子化
// FreeImage_ColorQuantizeEx 的目的是以基于输入图象的最佳选择填
// 充剩下的39 个调色板入口, 然后用大小为255 的调色板来将图象量子化。
// 输出的调色板将包含216 色和39 色的混合, 但并没有特定顺序。255 号
// 调色板入口第(256 个入口)在图象中没有用到, 它在调色板中会是黑色。
// 这就允许用户为透明度而使用255 号调色板入口,
// 而不必担心使得有效像素变成透明
FIBITMAP *dib8_a = FreeImage_ColorQuantizeEx(dib,
    FIQ_NNQUANT, 255, 216, web_palette);
// 函数的其他用途。只使用255 色, 因此第256 号颜色可用于透明度
FIBITMAP *dib8_b = FreeImage_ColorQuantizeEx(dib,
    FIQ_NNQUANT, 255, 0, NULL);
// 不产生多余的色彩, 只使用web-safe 颜色
FIBITMAP *dib8_c = FreeImage_ColorQuantizeEx(dib,
    FIQ_NNQUANT, 216, 216, web_palette);
// 来自一个不同的dib 的调色板来量子化
RGBQUAD another_palette[256];
// ...
FIBITMAP *dib8_d = FreeImage_ColorQuantizeEx(dib,
    FIQ_NNQUANT, 256, 256, another_palette);
// ...
FreeImage_Unload(dib8_a);
FreeImage_Unload(dib8_b);
FreeImage_Unload(dib8_c);
FreeImage_Unload(dib8_d);

```

FreeImage_Threshold

1 4 8 16 24 32

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_Threshold(FIBITMAP *dib, BYTE T);

通过使用一个在[0 ... 255]之间的阈值T来将一个位图转换为1位黑白位图。函数首先将位图转换为8位灰度位图, 然后凡低于T的亮度级都被设置为0, 否则

被设置为1。对于1位输入位图，函数对其进行克隆，并构造一个黑白调色板。

FreeImage_Dither

1 4 8 16 24 32

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_Dither(FIBITMAP *dib, FREE_
IMAGE_DITHER algorithm);

通过使用一种抖动算法来将一个位图转换为1位黑白位图。对于1位输入位图，函数对其进行克隆，并构造一个黑白调色板。

参数algorithm指定了要使用的抖动算法。函数首先将位图转换为8位灰度位图，然后用以下算法之一对位图进行抖动：

表 2.8:Free_Image_Dither常量

参数	抖动法
FID_FS	Floyd & Steinberg 误差扩散算法
FID_BAYER4x4	Bayer 弥散点抖动(秩为 2 - 4x4 -抖动矩阵)
FID_BAYER8x8	Bayer 弥散点抖动(秩为 3 - 8x8 -抖动矩阵)
FID_CLUSTER6x6	集簇点抖动(秩为 3 - 6x6 矩阵)
FID_CLUSTER8x8	有序(Ordered)集簇点抖动(秩为 4 - 8x8 矩阵)
FID_CLUSTER16x16	有序(Ordered)集簇点抖动(秩为 8 - 16x16 矩阵)

参考文献

Ulichney, R., Digital Halftoning. The MIT Press, Cambridge, MA, 1987.
Hawley S., Ordered Dithering. Graphics Gems, Academic Press, 1990.

FreeImage_ConvertFromRawBits

1 4 8 16 24 32

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertFromRawBits(BYTE *bits,
int width, int height, int pitch, unsigned bpp, unsigned red_mask, unsigned green_mask,
unsigned blue_mask, BOOL topdown FI_DEFAULT(FALSE));

将内存中某处的原始位图(raw bitmap)转换为一个FIBITMAP。该函数中的参数用来描述原始位图，第一个参数是指向原始位的一个指针；width和height参数描述位图的大小；pitch定义了源位图中一个扫描行的全部宽度(包括可能要用到的填充字节)；bpp参数告诉FreeImage位图的位深度是多少；red_mask、green_mask和blue_mask参数告诉FreeImage位图中颜色分量的位布局；最后一个参数topdown呢，当它为TRUE时将会首先储存最左上方的像素，而当它为FALSE时将会首先储存最左下方的像素。

?当源位图使用32位填充时，您可以用以下公式来计算(位)深度：

```
int pitch = (((bpp * width) + 31) / 32) * 4;
```

FreeImage_ConvertToRawBits

1 4 8 16 24 32

DLL_API void DLL_CALLCONV FreeImage_ConvertToRawBits(BYTE *bits, FIBITMAP *dib, int pitch, unsigned bpp, unsigned red_mask, unsigned green_mask, unsigned blue_mask, BOOL topdown FLDEFAULT(FALSE));

将一个FIBITMAP转换为内存中的一个原始块。内存的布局由传递的参数来描述，这些参数与上一个函数的参数是相同的。最后一个参数topdown呢，当它为TRUE时将会首先储存位图中最左上方的像素，而当它为FALSE时将会首先储存位图中最左下方的像素。

```
// 本段代码假设已载入一个由叫做" dib " 的变量代表的位图
// 转换一个位图为32 位原始缓冲块 ( 首先储存最左上方像素 )
// -----
FIBITMAP *src = FreeImage_ConvertTo32Bits(dib);
FreeImage_Unload(dib);
// 分配一个原始缓冲块内存
int width = FreeImage_GetWidth(src);
int height = FreeImage_GetHeight(src);
int scan_width = FreeImage_GetPitch(src);
BYTE *bits = (BYTE*)malloc(height * scan_width);
// 将位图储存为原始位 ( 首先储存最左上方像素 )
FreeImage_ConvertToRawBits(bits, src, scan_width, 32,
    FLRGBA_RED_MASK, FLRGBA_GREEN_MASK, FLRGBA_BLUE_MASK,
    TRUE);
FreeImage_Unload(src);
// 将内存中的一个32 位原始缓冲块转换为一个FIBITMAP
// ( 首先储存最左上方像素 )
// -----
FIBITMAP *dst = FreeImage_ConvertFromRawBits(bits, width,
    height, scan_width, 32, FLRGBA_RED_MASK,
    FLRGBA_GREEN_MASK, FLRGBA_BLUE_MASK, FALSE);
```

FreeImage_ConvertToStandardType

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertToStandardType(FIBITMAP *src, BOOL scale_linear FLDEFAULT(TRUE));

将一个色彩类型为FIC_MINISBLACK的非标准图象转换为一个标准8位位图(允许的转换见表 2.9)。当参数*scale_linear*为TRUE时,转换是通过将每个像素值从[*min*, *max*]线性地映射到到[0 ... 255]之间的一个整数值来完成的,这里*min*和*max*是图象中像素的最小和最大值。参数*scale_linear*为FALSE时,转换是通过将每个像素值舍入到[0 ... 255]之间的一个整数值来完成的,舍入用的是下面的公式:

dst_pixel = (BYTE) MIN(255, MAX(0, *q*)) 这里 int *q* = int(*src_pixel* + 0.5);

函数返回转换后的8位灰度图象。对于标准图象,返回的是它的克隆。

FreeImage_ConvertToType


1 4 8 16 24 32

16 32 64

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertToType(FIBITMAP *src, FREE_IMAGE_TYPE dst_type, BOOL scale_linear FI_DEFAULT(TRUE));

将任意类型的一幅图象转换为*dst_type*类型。当*dst_type*等于FIT_BITMAP时,该函数调用FreeImage_ConvertToStandardType函数,其他情况下转换通过使用C语言的类型转换约定来完成。当一种转换不被允许时,返回NULL值并扔出一个错误信息(可用FreeImage_SetOutputMessage 函数来捕捉到)。以下是目前被FreeImage库允许的转换(需要的话,其他转换也可以容易地添加):

表 2.9:FreeImage允许的位图类型转换

	FIT_BITMAP	FIT_UINT16	FIT_INT16	FIT_UINT32	FIT_INT32	FIT_FLOAT	FIT_DOUBLE	FIT_COMPLEX
FIT_BITMAP	●	●	●	●	●	●	●	●
FIT_UINT16	●	●				●	●	●
FIT_INT16	●		●			●	●	●
FIT_UINT32	●				●	●	●	●
FIT_INT32	●				●	●	●	●
FIT_FLOAT	●					●	●	●
FIT_DOUBLE	●						●	●
FIT_COMPLEX								●

FreeImage_ConvertToRGBF

24 32

48 96

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertToRGBF(FIBITMAP *dib);
将一幅24位或32位RGB(A)标准图象或一幅48位RGB图象转换为一个FIT_BITMAP类型图象。转换是这样完成的：即把源图象中的整数值像素复制到目的图象中的浮点像素值，并除以源图象中的最大像素值(即255或65535)，这样输出图象的像素值是在[0 ... 1]之间。如果在源图象中有alpha通道，它会被转换函数简单地忽略掉。对于RGBF输入图象，返回的是它的克隆。

2.7 调和映射操作算子

调和映射操作被用于将像素的一个大亮度范围压缩到一个较小的、适于在动态范围有局限的设备(诸如CRT显示器或液晶显示器和打印媒体等)上显示的范围。

这个问题的原理是简单的：我们需要把一幅具有大数字范围的图象转换为一幅包含0到255范围内整数的图象，这样我们就可以在打印机上打印或在显示器上显示它。这就提出了线性比例(linear scaling)作为可行方案。但是这种近似是有缺陷的，因为图象中的明亮区域和黑暗区域的细节会由于后续的量子化(subsequent quantization)而丢失，显示的图象也因而不会被感到与拍照的景象相同。因为这个原因，提出了更精细复杂的算法——叫做调和映射——用来精确地渲染高动态图象。

FreeImage_ToneMapping

48 96

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ToneMapping(FIBITMAP *dib, FREE_IMAGE_TMO tmo, double first_param FI_DEFAULT(0), double second_param FI_DEFAULT(0));
将一幅高动态范围图象(48位RGB或96位RGBF)转换为适合于显示的24位RGB图象。tmo参数指定了要用到的调和映射操作算子。函数首先(用FreeImage_ConvertToRGBF函数)将输入图象转换为96位RGBF图象，然后对图象以下列算法之一进行调和映射：

表 2.10:FREE_IMAGE_TMO常量

参数	调和映射操作算子
FITMO_DRAGO03	Adaptive logarithmic mapping (F. Drago, 2003)
FITMO_REINHARD05	Dynamic range reduction inspired by photoreceptor physiology (E. Reinhard, 2005)

*first_param*参数和*second_param*参数的意义取决于选择的算法(参见下面每个调和映射操作算子的定义)。当这两个参数都被设置为0时,使用一组默认的参数。

```
// 载入一幅高动态渲染浮点图象RGB
FIBITMAP *src = FreeImage_Load(FIF_HDR, "memorial.hdr",
    0);
// 创建一幅适合于显示的、经过调和映射的图象
FIBITMAP *dst = FreeImage_ToneMapping(src, FITMO_DRAGO03)
    ;
// ...
FreeImage_Unload(src);
FreeImage_Unload(dst);
```

FreeImage_TmoDrago03

48 96

```
DLL_API FIBITMAP* DLL_CALLCONV FreeImage_TmoDrago03(FIBITMAP *src,
double gamma FLDEFAULT(2.2), double exposure FLDEFAULT(0));
```

模拟人对光的反应,使用一个基于亮度值对数压缩的全局操作算子来将一幅高动态范围图象转换为一幅24位RGB图象。引入了一个偏置幂函数(bias power function)以适当地变动对数的底,形成对清晰度和对比度的良好保护。

在入口, *gamma*(此处 $\gamma > 0$)是一个 γ 校正,它在调和映射之后被调用。 γ 值为1表示无校正,用于原作者论文中的默认值,是作为一个较好的初始值被推荐的。

*exposure*参数是曝光比例因子,它允许用户将输入图象的亮度调整到他们的显示环境。默认值(0)表示未使用校正,较高的值使图象更明亮,而较低的值使图象更暗。

参考文献

F. Drago, K. Myszkowski, T. Annen and N. Chiba, Adaptive logarithmic mapping for displaying high contrast scenes. Proceedings of Eurographics2003, Vol.22, No, 3, pp. 419-426, 2003.

FreeImage_TmoReinhard05

48 96

```
DLL_API FIBITMAP* DLL_CALLCONV FreeImage_TmoReinhard05(FIBITMAP *src,
double intensity FLDEFAULT(0), double contrast FLDEFAULT(0));
```

使用灵感源于人类视觉系统的图象感官生理学的一种全局算子，将一幅高动态图象转换为了一幅24位RGB图象。

在入口，在[-8,8]范围内的*intensity*参数控制着总体图象明暗度，默认值0表示无校正，较高的值使图象更明亮，而较低的值使图象更暗。

在[-0.3,1.0]范围内的*contrast*参数控制着总体图象对比度，当使用默认值(0)时，该参数是自动被计算的。

参考文献

E. Reinhard and K. Devlin, Dynamic Range Reduction Inspired by Photoreceptor Physiology. IEEE Transactions on Visualization and Computer Graphics, 11(1), Jan/Feb 2005.

2.8 ICC 轮廓函数

无论何时，只要位图中存在ICC(Image Color Correction,图象色彩校正)轮廓，那么它在FIBITMAP中是被透明地装载和储存的。反过来，假如输出的*FREEIMAGE_FORMAT*支持ICC轮廓(可用*FreeImage_FIFSupportsICCProfiles*来申请对ICC轮廓支持的一个插件)，那么无论何时只要ICC轮廓被储存在一个FIBITMAP中，当保存位图时它就被透明地储存在位图中。

FreeImage定义了一个用来访问ICC轮廓、叫做FIICCPROFILE的结构，然后就可以把它用在任意色彩管理引擎中，以便在两个ICC轮廓中执行位图传输。

如果FIICCPROFILE被设置了COLOR_IS_CMYK标志，那么位图就代表一个CMYK分色位图。这个信息与色彩管理信息都是重要的，因为轮廓数据和位图必须驻留在相同的色彩模型中(例如RGB或CMYK)。

在几乎所有情况下，位图是作为代表RGB的位图来装载的，原色是否被保护会依赖于*FreeImage_Load*的特殊标志。

```
// 从文件中载入一个位图，强制性保护 TIFF 中的CMYK 分色数据
// ( 不作RGB 转换)
FIBITMAP *bitmap = FreeImage_Load (FIF_TIFF, name,
    TIFF_CMYK);
if (bitmap) { // 测试RGB 或CMYK 色彩空间
if ((FreeImage_GetICCProfile(bitmap) > flags &
    FIICC_COLOR_IS_CMYK) == FIICC_COLOR_IS_CMYK)
// 我们是在CMYK 色彩空间
else
// 我们是在RGB 色彩空间}
```

?目前ICC轮廓被TIFF、PNG和JPEG插件所支持。

FreeImage_GetICCProfile

DLL_API FIICCPROFILE *DLL_CALLCONV FreeImage_GetICCProfile(FIBITMAP *dib);

获取指向位图的FIICCPROFILE数据的指针。当源图象的格式不支持轮廓时，该函数也可以被安全地调用。

```
// 本段代码假设已载入一个由叫做"bitmap" 的变量代表的位图
// 获取指向FIICCPROFILE 数据结构的指针
FIICCPROFILE *profile = FreeImage_GetICCProfile(bitmap);
If (profile->data) { } // 有轮廓数据存在
```

FreeImage_CreateICCProfile

DLL_API FIICCPROFILE *DLL_CALLCONV FreeImage_CreateICCProfile(FIBITMAP

*dib, void *data, long size);

从先前在文件中读取、或由一个色彩管理系统建立的ICC轮廓数据中创建一个新FIICCPROFILE 块，轮廓数据是与位图相关联的。函数返回指向被创建的FIICCPROFILE结构的一个指针。

```
// 本段代码假设已载入一个由叫做"bitmap" 的变量代表的位图
DWORD size = _filelength( fileno( hProfile));
// 从文件中读取数据和ASCII 零结束符
if (size && (data = (void *)malloc(size + 1))) {
    size = fread(data, 1, size, hProfile);
    *(data + size) = 0;
    // 将获取的轮廓数据与位图相关联
    FIICCPROFILE *profile = FreeImage_CreateICCProfile (
        bitmap, data, size);
    free (data);
}
```

FreeImage_DestroyICCProfile

DLL_API void DLL_CALLCONV FreeImage_DestroyICCProfile(FIBITMAP *dib);

该函数销毁一个先前由FreeImage_CreateICCProfile创建的FIICCPROFILE。调用该函数后位图里将不包含轮廓信息。要确保一幅保存后的位图将不包含任何轮廓信息，应该调用该函数。

```
// 本段代码假设已载入一个由叫做"bitmap" 的变量代表的位图
// 销毁可能存在的轮廓
FreeImage_DestroyICCProfile(bitmap);
// 保存没有轮廓的位图
FreeImage_Save (FIF_TIFF, bitmap, name, flags);
```

2.9 插件函数

通过一般地使用FreeImage您不可能注意到这样的事实：即FreeImage是插件驱动的。每个位图载入器/保存器实际上是一个连接在集成插件管理器内的模块。直到决定编写自己的插件之前，您不会注意到这个事实的。

几乎每个FreeImage中的插件是直接并入DLL中的。为什么这样做有历史进展和设计的混合原因。FreeImage的第一版(实际上，大约在其生命中第一年的全年)还没有插件的概念，这意味着所有位图函数只有在主DLL库中才提供。第二年，Floris决定创建插件，因为他想要支持诸如GIF格式的一些在其上有许可限制的位图格式。因为怕要将所有位图载入器/保存器放到多个小的DLL(它们会使硬盘速度慢变得结结巴巴)中，他的主要'客户'强烈鼓动他将尽可能多的位图格式保持在一个DLL中，他采纳了这个意见，导致了今天您在这里看到的设计。

实际上的插件系统从一些非常简单的东西，演化成了他现在在其他软件中经常重复用到的非常灵活的一种机制。目前插件已经可能存在于主DLL中、在外部DLL中、甚至直接在驱动FreeImage的应用程序中。

FreeImage_GetFIFCount

DLL_API int DLL_CALLCONV FreeImage_GetFIFCount();
 获取当前正在注册的FREE_IMAGE_FORMAT标识符的数目。在FREE_IMAGE_FORMAT结构中通过演化，变成了与插件的同义词。

FreeImage_SetPluginEnabled

DLL_API int DLL_CALLCONV FreeImage_SetPluginEnabled(FREE_IMAGE_FORMAT fif, BOOL enable);
 激活或禁止一个插件。被禁止的插件不能用来导入或输出位图，也不能标识位图。当该函数被调用时，返回先前的插件状态(TRUE/1或FALSE/0)，或如果插件不存在时返回 - 1。

FreeImage_IsPluginEnabled

DLL_API int DLL_CALLCONV FreeImage_IsPluginEnabled(FREE_IMAGE_FORMAT fif);
 如果插件被激活返回TRUE，当插件被禁止则返回FALSE，其他情况下返回-1。

FreeImage_GetFIFFromFormat

DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_GetFIFFromFormat(const char *format);
 从用来注册FIF的格式串中返回一个FREE_IMAGE_FORMAT标识符。

FreeImage_GetFIFFromMime

DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_GetFIFFromMime
(const char *mime);
从**MIME**内容类型串(**MIME**代表**Multipurpose Internet Mail Extension**)中返回一个**FREE_IMAGE_FORMAT**标识符。

```
FREE_IMAGE_FORMAT fif = FreeImage_GetFIFFromMime("image/
    png");
If( fif != FIF_UNKNOWN) {
    assert( fif == FIF_PNG);
}
```

FreeImage_GetFIFMimeType

DLL_API const char *DLL_CALLCONV FreeImage_GetFIFMimeType(FREE_IMAGE_
_FORMAT fif);
给定一个**FREE_IMAGE_FORMAT**标识符, 返回一个**MIME**内容类型
串(**MIME**代表**Multipurpose Internet Mail Extension**)。

FreeImage_GetFormatFromFIF

DLL_API const char *DLL_CALLCONV FreeImage_GetFormatFromFIF(FREE_IMAGE_
_FORMAT fif);
返回一个用来从系统指定的**FREE_IMAGE_FORMAT**中注册插件的
串。

FreeImage_GetFIFExtensionList

DLL_API const char *DLL_CALLCONV FreeImage_GetFIFExtensionList(FREE_IMAGE_
_FORMAT fif);
返回一个以逗号分隔的扩展名列表, 它描述给定插件所能读和/或写的位图
格式。

```
//建造一个字符串对序列, 它指定可用来装载文件的过滤器以用于打开文件对  
//话框 (GetOpenFileName 或 CFileDialog)中。  
//@param szFilter 输入和输出参数。szFilter 是一个字符数组, 其全  
//部长度应为 1024 或 1024 以上。  
//@return 返回被支持的输入格式的数目  
int GetOpenFilterString(char *szFilter) {  
    int i, iCount;  
    char Filter[1024];  
    char *token;  
    // 为 'All image files' 建造一个串 '
```

```

Filter[0] = '\0';
for(i = 0; i < FreeImage_GetFIFCount(); i++) {
    if(FreeImage_FIFSupportsReading((FREE_IMAGE_FORMAT)i)
        ) {
        strcat(Filter, FreeImage_GetFIFExtensionList((
            FREE_IMAGE_FORMAT)i));
        strcat(Filter, ",");
    }
}
Filter[strlen(Filter)-1] = '\0';
strcpy(szFilter, "All_image_files|");
token = strtok(Filter, ",");
while(token != NULL) {
    strcat(szFilter, "*.");
    strcat(szFilter, token);
    strcat(szFilter, ";");
    // 获取下一个 token
    token = strtok(NULL, ",");
}
szFilter[strlen(szFilter)-1] = '|';
// 为 'All_files' 建造一个串
strcat(szFilter, "All_Files_(*)|*.*|");
// 为每种格式建造一个串
Filter[0] = '\0';
iCount = 0;
for(i = 0; i < FreeImage_GetFIFCount(); i++) {
    if(FreeImage_FIFSupportsReading((FREE_IMAGE_FORMAT)i)
        ) {
        // 描述
        sprintf(Filter, "%s_(%s)|",
            FreeImage_GetFIFDescription((FREE_IMAGE_FORMAT)i
            ), FreeImage_GetFIFExtensionList((
                FREE_IMAGE_FORMAT)i));
        strcat(szFilter, Filter);
        // 扩展名
        strcpy(Filter, FreeImage_GetFIFExtensionList((
            FREE_IMAGE_FORMAT)i));
    }
}

```

```

    token = strtok(Filter, ",");
    while(token != NULL) {
        strcat(szFilter, ".*");
        strcat(szFilter, token);
        strcat(szFilter, ";");
        // 获取下一个 token
        token = strtok(NULL, ",");
    }
    szFilter[strlen(szFilter)-1] = '|';
    iCount++;
}
}
strcat(szFilter, "|");
return iCount;
}

```

FreeImage_GetFIFDescription

DLL_API const char *DLL_CALLCONV FreeImage_GetFIFDescription(FREE_IMAGE_FORMAT fif);

返回一个说明字符串，它描述给定的插件能读和/或写的位图格式。

FreeImage_GetFIFRegExpr

DLL_API const char *DLL_CALLCONV FreeImage_GetFIFRegExpr(FREE_IMAGE_FORMAT fif);

返回一个能被正则表达式引擎使用的正则表达式字符串来标识位图。

FreeImageQt利用到了这个函数。

FreeImage_GetFIFFromFilename

DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_GetFIFFromFilename(const char *filename);

这个函数用一个文件名或文件扩展名为参数，并返回一个可以通过FREE_IMAGE_FORMAT标识符的形式读/写具有该扩展名的文件的插件。

```

// 通用图象载入器
// @param lpszPathName 指向全文件名的指针
// @param flag 可选装载标志常量
// @return 若成功返回被载入的 dib , 否则返回 NULL
FIBITMAP* GenericLoader(const char* lpszPathName, int
    flag) {
    FREE_IMAGE_FORMAT fif = FIF_UNKNOWN;
    // 检查文件签名并推断其格式目前 (没有用到第二个参数 FreeImage)
    fif = FreeImage_GetFileType(lpszPathName, 0);
    if(fif == FIF_UNKNOWN) {
        //没有签名? 试着从文件扩展名来猜测文件格式
        fif = FreeImage_GetFIFFromFilename(lpszPathName);
    }
    // 检查插件是否具有读取该格式的能力 ...
    if((fif != FIF_UNKNOWN) && FreeImage_FIFSupportsReading
        (fif)) {
        // 好, 让我们来载入文件
        FIBITMAP *dib = FreeImage_Load(fif, lpszPathName,
            flag);
        // 除非是坏的文件格式, 不然的话就大功告成了!
        return dib;
    }
    return NULL;
}

```

FreeImage_GetFIFFromFilenameU

DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_GetFIFFromFilenameU
(const wchar_t *filename);

这个函数起的作用与 *FreeImage_GetFIFFromFilename* 一样, 但它还支持 UNICODE 文件名。请注意该函数只在 32 位 Windows 操作系统下起作用, 在其他操作系统下函数不做任何事, 并返回 FIF_UNKNOWN。

FreeImage_FIFSupportsReading

DLL_API BOOL DLL_CALLCONV FreeImage_FIFSupportsReading(FREE_IMAGE_
FORMAT fif);

如果属于给定 FREE_IMAGE_FORMAT 的插件可以用来载入位图则返回 TRUE, 否则返回 FALSE。

FreeImage_FIFSupportsWriting

DLL_API BOOL DLL_CALLCONV FreeImage_FIFSupportsWriting(FREE_IMAGE_FORMAT fif);

如果属于给定FREE_IMAGE_FORMAT的插件可以用来保存位图则返回TRUE, 否则返回FALSE。

```

/** 通用图像写入器
@param dib 指向要保存的的指针 dib
@param lpszPathName 指向全文件名的指针
@param flag 可选保存标志常量
@return 若成功返回 true , 否则返回 false
*/
bool GenericWriter(FIBITMAP* dib, const char*
    lpszPathName, int flag) {
    FREE_IMAGE_FORMAT fif = FIF_UNKNOWN;
    BOOL bSuccess = FALSE;
    // 试着从文件扩展名来猜测文件格式
    fif = FreeImage_GetFIFFromFilename(lpszPathName);
    if( fif != FIF_UNKNOWN ) {
        // 检查该种格式的是否可以保存 dib
        BOOL bCanSave;
        FREE_IMAGE_TYPE image_type = FreeImage_GetImageType(
            _dib);
        if(image_type == FIT_BITMAP) {
            // 标准位图类型, 检查插件是否具有足够的写入和输出能力 ...
            WORD bpp = FreeImage_GetBPP(_dib);
            bCanSave = (FreeImage_FIFSupportsWriting(fif) &&
                FreeImage_FIFSupportsExportBPP(fif, bpp));
        } else { //特殊位图格式, 检查插件是否具有足够的输出能力
            bCanSave = FreeImage_FIFSupportsExportType(fif,
                image_type);
        }
        if(bCanSave) {
            bSuccess = FreeImage_Save(fif, _dib, lpszPathName,
                flag);
        }
    }
    return (bSuccess == TRUE) ? true : false;
}

```

FreeImage_FIFSupportsExportType

DLL_API BOOL DLL_CALLCONV FreeImage_FIFSupportsExportType(FREE_IMAGE_FORMAT fif, FREE_IMAGE_TYPE type);

如果属于给定FREE_IMAGE_FORMAT的插件能够以所需要的数据类型保存位图则返回TRUE, 否则返回FALSE。关于可以保存非标准图象的插件一览表, 请参见附录中的被支持的文件格式表。

FreeImage_FIFSupportsExportBPP

DLL_API BOOL DLL_CALLCONV FreeImage_FIFSupportsExportBPP(FREE_IMAGE_FORMAT fif, int bpp);

如果属于给定FREE_IMAGE_FORMAT的插件能够以所需要的位深度保存位图则返回TRUE, 否则返回FALSE。

```
// 建造一个字符串对序列, 它指定可用来保存文件的过滤器以用于保存文件
// 对话框 (GetSaveFileName 或 CFileDialog) 中
// @param szFilter 输入和输出参数。 szFilter 是一个字符数组, 其
// 全部长度应为 1024 或 1024 以上。
// @param bpp 要保存的图象的位深度
// @return 返回被支持的输出格式的数目
int GetSaveAsFilterString(char *szFilter, WORD bpp) {
    int i, iCount;
    char Filter[1024];
    char *token;
    szFilter[0] = '\0';
    iCount = 0;
    // 为每种格式建一个串
    for(i = 0; i < FreeImage_GetFIFCount(); i++) {
        if(FreeImage_FIFSupportsExportBPP((FREE_IMAGE_FORMAT)
            i, bpp)) {
            // 处理 PNM 文件的特殊情况
            strcpy(Filter, FreeImage_GetFormatFromFIF((
                FREE_IMAGE_FORMAT)i));
            if((bpp == 1) && (!strcmp(Filter, "PGM", 3) || !
                strcmp(Filter, "PPM", 3)))
                continue;
            if((bpp == 8) && (!strcmp(Filter, "PBM", 3) || !
                strcmp(Filter, "PPM", 3)))
                continue;
        }
    }
}
```

```

        if((bpp == 24) && (!strncmp(Filter, "PGM", 3) || !
            strncmp(Filter, "PBM", 3)))
            continue;
        // 描述
        sprintf(Filter, "%s_(%s)|",
            FreeImage_GetFIFDescription((FREE_IMAGE_FORMAT)i),
            FreeImage_GetFIFExtensionList((
                FREE_IMAGE_FORMAT)i));
        strcat(szFilter, Filter);
        // 扩展名
        strcpy(Filter, FreeImage_GetFIFExtensionList((
            FREE_IMAGE_FORMAT)i));
        token = strtok(Filter, ",");
        while(token != NULL) {
            strcat(szFilter, "*.");
            strcat(szFilter, token);
            strcat(szFilter, ";");
            // 获取下一个 token
            token = strtok(NULL, ",");
            szFilter[strlen(szFilter)-1] = '|';
            iCount++;
        }
        strcat(szFilter, "|");
        return iCount;
    }

```

FreeImage_FIFSupportsICCProfiles

DLL_API BOOL DLL_CALLCONV FreeImage_FIFSupportsICCProfiles(FREE_IMAGE_FORMAT fif);

如果属于给定FREE_IMAGE_FORMAT的插件能够载入或保存一个ICC轮廓则返回TRUE，否则返回FALSE。

```

// 确定是否提供了对轮廓的支持
if (FreeImage_FIFSupportsICCProfiles(FIF\_TIFF)) {
    // 提供了对轮廓的支持
}

```

FreeImage_RegisterLocalPlugin

```
DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_RegisterLocalPlugin(FI
_InitProc proc_address, const char *format FI_DEFAULT(0), const char *description FI_DEFAULT(0),
const char *extension FI_DEFAULT(0), const char *regexpr FI_DEFAULT(0));
```

注册一个新的插件以在FreeImage中使用。插件直接驻留在驱动FreeImage的应用程序中。第一个参数是指向对插件进行初始化的函数的一个指针。插件负责填充一个Plugin结构，并储存一个系统设定的格式标识号用以对消息进行日志管理。

```
static int s_format_id;
void stdcall
Init(Plugin *plugin, int format_id) {
    s_format_id = format_id;
    // 指向一个函数的指针，该函数返回关于位图类型的类型字符串
    // 例如载入 BMP 的一个插件返回串 "BMP"
    plugin->format_proc = Format;
    // 指向一个函数的指针，该函数返回关于位图类型的说明字符串
    // 例如载入 BMP 的插件可能返回串 "Windows or OS/2 Bitmap"
    plugin->description_proc = Description;
    // 指向一个函数的指针，该函数返回一个以逗号分隔的、对该插件有
    // 效的可能文件扩展名列表
    // 一个 JPEG 插件会返回 "jpeg, jif, jfif"
    plugin->extension_proc = Extension;
    // 指向一个用来载入位图的函数的指针
    plugin->load_proc = Load;
    // 指向一个用来保存位图的函数的指针
    plugin->save_proc = Save;
    // 指向一个函数的指针，该函数将会试着通过查看位图的前面少量字节
    // 来标识位图
    plugin->validate_proc = Validate;
}
```

FreeImage_RegisterExternalPlugin

```
DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_RegisterExternalPlugin
(const char *path, const char *format FI_DEFAULT(0), const char *description FI_DEFAULT(0),
const char *extension FI_DEFAULT(0), const char *regexpr FI_DEFAULT(0));
```

注册一个新的插件以在FreeImage中使用。插件驻留在DLL中。在功能上来说，这个函数与FreeImage_RegisterLocalPlugin是相同的，但现在FreeImage调用一个DLL中的Init函数，而不是调用应用程序中的本地函数。Init函数必须叫

做“Init”而且必须使用stdcall调用约定。

2.10 多页函数

FreeImage实现了函数集的特色,这些函数可以用来在一个多页位图格式中处理多页。目前对TIFF、ICO和GIF都支持此功能。多页API使得在一个多页位图中访问、改变其中的页或删除页、改变页顺序成为可能。所有这些都是通过一个复杂的压缩缓存机制,以插件中最小的实现代码和对内存的最小需求的代价来提供的。

?在多页API中,无论何时函数需要一个‘page’(页)参数,它都总是以0为基的(即序号从0编起)。

FreeImage_OpenMultiBitmap

DLL_API FIMULTIBITMAP * DLL_CALLCONV FreeImage_OpenMultiBitmap(FREE_IMAGE_FORMAT fif, const char *filename, BOOL create_new, BOOL read_only, BOOL keep_cache_in_memory FI_DEFAULT(FALSE), int flags FI_DEFAULT(0));

打开一个多页位图。第一个参数告诉FreeImage要打开的位图的类型,目前支持的类型有FIF_TIFF、FIF_ICO和FIF_GIF。第二个参数指定位图名。当第三个参数为TRUE时表示将创建一个新位图,而不是打开一个现有的位图。当第四个参数为TRUE时,将会以只读模式打开位图。keep_cache_in_memory参数纯粹是为了性能的考虑而设的,当它为TRUE时所有收集到的位图数据在页处理过程中保持在内存中,否则这些数据以64K字节块为单位被缓慢注入到硬盘上的一个临时文件。请注意根据位图大小和所执行的处理量大小的不同,这个临时文件可以变得相当大。这种缓冲到磁盘上的做法是明智的。最后的参数用来改变位图插件的行为,或激活插件中某种特性。每个插件有它自己的参数集。

FreeImage_CloseMultiBitmap

DLL_API BOOL DLL_CALLCONV FreeImage_CloseMultiBitmap(FIMULTIBITMAP *bitmap, int flags FI_DEFAULT(0));

关闭一个先前打开的多页位图,并在位图不是以只读模式打开的情况下,应用对位图的任何改动。

flags参数用来改变位图插件的行为,或激活插件中某种特性。每个插件有它自己的参数集(参见表2.4)。一些位图保存器可以接受参数来改变保存行为。当没有这些参数或未用到这些参数时,您可以传递0值或<TYPE_OF_BITMAP>_DEFAULT(例如TIFF_DEFAULT, ICO_DEFAULT等等)。

FreeImage_GetPageCount

DLL_API int DLL_CALLCONV FreeImage_GetPageCount(FIMULTIBITMAP *bitmap);

返回当前多页位图中现有的页数。

FreeImage_AppendPage

DLL_API void DLL_CALLCONV FreeImage_AppendPage(FIMULTIBITMAP *bitmap, FIBITMAP *data);

追加一个新页到位图末。

FreeImage_InsertPage

DLL_API void DLL_CALLCONV FreeImage_InsertPage(FIMULTIBITMAP *bitmap, int page, FIBITMAP *data);

在位图的给定位置之前插入一个新页。Page参数必须是小于当前位图中现有页数的一个数。

FreeImage_DeletePage

DLL_API void DLL_CALLCONV FreeImage_DeletePage(FIMULTIBITMAP *bitmap, int page);

在位图的给定位置删除一页。

FreeImage_LockPage

DLL_API FIBITMAP * DLL_CALLCONV FreeImage_LockPage(FIMULTIBITMAP *bitmap, int page);

在内存中锁定位图页以供编辑。现在可以把位图页保存到另一个文件，或插入到另一个多页位图中。对位图页操作完毕后，必须调用FreeImage_UnlockPage来将位图页归还到位图中，并应用对位图页的任何改动。

?用FreeImage_Unload来解除对一个锁定页的锁定是被禁止的：您必须使用FreeImage_UnlockPage来代替。

FreeImage_UnlockPage

DLL_API void DLL_CALLCONV FreeImage_UnlockPage(FIMULTIBITMAP *bitmap, FIBITMAP *page, BOOL changed);

解除先前对一个位图页的锁定，并将位图页归还到多页引擎中。当最后的参数为TRUE时位图页被打上已改动标记，并且新位图页被应用到多页位图中。

FreeImage_MovePage

DLL_API BOOL DLL_CALLCONV FreeImage_MovePage(FIMULTIBITMAP *bitmap, int target, int source);

将源页移动到目标页的位置。成功时返回TRUE，失败时返回FALSE。

FreeImage_GetLockedPageNumbers

DLL_API BOOL DLL_CALLCONV FreeImage_GetLockedPageNumbers(FIMULTIBITMAP *bitmap, int *pages, int *count);

返回由当前内存中锁定页的页码组成的一个数组。当pages参数为NULL时，数组的大小在count变量中返回。然后您就可以根据需要的大小来给数组分配内存，并再次调用FreeImage_GetLockedPageNumbers来定位(populate)数组。

2.11 内存输入/输出流

内存输入/输出例程使用FreeImageIO结构的一种特殊版本，目的是向/从内存流中保存/载入FIBITMAP图象，就象您对文件流所做的那样。内存文件流(由FreeImage内部管理)支持从文件中载入和保存FIBITMAP。

FreeImage_OpenMemory

DLL_API FIMEMORY *DLL_CALLCONV FreeImage_OpenMemory(BYTE *data FI_DEFAULT(0), DWORD size_in_bytes FI_DEFAULT(0));

打开一个内存流。函数返回一个指向已打开的文件流的指针。当以默认值(0)来调用该函数时，它打开一个内存流来进行读/写访问，这个流支持从内存文件中载入和保存FIBITMAP(由FreeImage内部管理)，它还支持在内存文件中寻址和定位。

这个函数还可以用来保护由驱动FreeImage的应用程序提供的一个内存缓冲块，包含图象数据的一个缓冲块是作为函数的data变参(缓冲区首址)和size_in_bytes变参(缓冲区以字节为单位的大小)给出的。由FreeImage保护的内存块是只读的，可以从其中载入图象，但不能向其中保存图象。

FreeImage_CloseMemory

DLL_API void DLL_CALLCONV FreeImage_CloseMemory(FIMEMORY *stream);

关闭并释放(free)一个内存流。当流被FreeImage管理时，内存文件被销毁(destroyed)。否则(被封装的缓冲块)，它的析构(destruction)任务留给了驱动FreeImage的应用程序。

!一旦对内存流操作完毕，永远要调用这个函数(不管流是如何打开的)，否则会有内存泄漏。

FreeImage_LoadFromMemory

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_LoadFromMemory(FREE_IMAGE_FORMAT fif, FIMEMORY *stream, int flags FI_DEFAULT(0));

这个函数对内存流做**FreeImage_Load**对文件流所做的事情。**FreeImage_LoadFromMemory**对一个位图进行解码、为其分配内存然后将其作为FIBITMAP返回。第一个参数定义要载入的位图的类型，例如当传递FIF_BMP时，一个BMP文件被载入内存(可能的常量一览表见 2.1)。第二个参数告知FreeImage它必须要解码的内存流。最后的参数用来改变位图插件的行为，或激活插件中某种特性，每个插件有它自己的参数集。

一些位图载入器可以接受参数来改变载入行为(参见表 2.3)。当没有参数或未用到参数时,可以向函数传递0值或<位图类型>_DEFAULT(如BMP_DEFAULT、ICO_DEFAULT等等)。

```
void testLoadMemIO(const char *lpszPathName) {
    struct stat buf;
    int result;
    // 将数据与 lpszPathName 相关联
    result = stat(lpszPathName, &buf);
    if(result == 0) { // 为一个内存缓冲块分配内存并载入临时数据
        BYTE *mem_buffer = (BYTE*)malloc(buf.st_size * sizeof
            (BYTE));
        if(mem_buffer) {
            FILE *stream = fopen(lpszPathName, "rb");
            if(stream) {
                fread(mem_buffer, sizeof(BYTE), buf.st_size,
                    stream);
                fclose(stream);
                // 将二进制数据与一个内存流相关联
                HMEMORY *hmem = FreeImage_OpenMemory(mem_buffer,
                    buf.st_size);
                // 获取文件类型
                FREEIMAGEFORMAT fif =
                    FreeImage_GetFileTypeFromMemory(hmem, 0);
                // 从内存流中载入一幅图象
                FIBITMAP *check = FreeImage_LoadFromMemory(fif,
                    hmem, 0);
                // 象普通文件那样保存
                FreeImage_Save(FIF_PNG, check, "blob.png",
                    PNG_DEFAULT);
                FreeImage_Unload(check);
                // 关闭流
                FreeImage_CloseMemory(hmem);
            }
        } // 用户负责释放数据所占用内存
        free(mem_buffer);
    }
}
```


FreeImage_SaveToMemory

```
DLL_API BOOL DLL_CALLCONV FreeImage_SaveToMemory(FREE_IMAGE_FORMAT
fif, FIBITMAP *dib, FIMEMORY *stream, int flags FI_DEFAULT(0));
```

这个函数对内存流做*FreeImage_Save*对文件流所做的事情。*FreeImage_SaveToMemory*将先前载入的一个FIBITMAP保存到一个由FreeImage管理的内存文件中。第一个参数定义要保存的位图的类型，例如当传递FIF_BMP时，一个BMP文件被保存(可能的常量一览表见 2.1)。第二个参数是必须要在其中保存位图的内存流。当内存指针指向内存文件的起始时，现有的数据被覆盖。另外(Otherwise)，您可以在同样的流中保存多页。

请注意，一些位图保存插件对能保存的位图类型有限制。例如，JPEG插件只能保存24位和8位灰度位图。最后的参数用来改变位图插件的行为，或激活插件中某种特性，每个插件有它自己的参数集。

一些位图保存器可以接受参数来改变保存行为(参见表 2.4)。当没有参数或未用到参数时，可以向函数传递0值或<位图类型>_DEFAULT(如BMP_DEFAULT、ICO_DEFAULT等等)。

```
void testSaveMemIO(const char *lpszPathName) {
    FIMEMORY *hmem = NULL;
    // 载入一个普通文件并对其进行解码
    FREE_IMAGE_FORMAT fif = FreeImage_GetFileType(
        lpszPathName);
    FIBITMAP *dib = FreeImage_Load(fif, lpszPathName, 0);
    // 打开一个内存流
    hmem = FreeImage_OpenMemory();
    // 对图象进行编码并将其保存到内存中
    FreeImage_SaveToMemory(fif, dib, hmem, 0);
    // hmem 包含内存中以 fif 格式储存的数据，内存占用空间大小
    // 等于 file_size
    long file_size = FreeImage_TellMemory(hmem);
    printf("File_size: %ld\n", file_size);
    // 在内存中载入一幅图象也容易。移动文件读写指针到内存流的起始
    FreeImage_SeekMemory(hmem, 0L, SEEK_SET);
    // 获取文件类型
    FREE_IMAGE_FORMAT mem_fif =
        FreeImage_GetFileTypeFromMemory(hmem, 0);
    // 在内存句柄载入一幅图象
    FIBITMAP *check = FreeImage_LoadFromMemory(mem_fif,
        hmem, 0);
    // 把它保存为普通文件
```

```

FreeImage_Save(FIF_PNG, check, "dump.png", PNG_DEFAULT);
// 确保流已关闭, 因为 FreeImage_SaveToMemory 会引起内部的
// 内存分配, 这是释放这块内存的唯一办法
FreeImage_CloseMemory(hmem);
FreeImage_Unload(check);
FreeImage_Unload(dib);
}

```

FreeImage_AcquireMemory

DLL_API BOOL DLL_CALLCONV FreeImage_AcquireMemory(FIMEMORY *stream, BYTE **data, DWORD *size_in_bytes);

提供对一个内存流的直接缓冲访问。在入口, *stream* 是目标内存流, 返回值 *data* 是指向内存缓冲区的指针, 返回值 *size_in_bytes* 是缓冲区以字节为单位的大小。如果调用成功, 函数返回 TRUE, 否则返回 FALSE。

!当内存流由 FreeImage 内部管理时, 一旦您调用 *FreeImage_SaveToMemory*, 则 *FreeImage_AcquireMemory* 返回的数据指针可能会变为无效。

```

void testAcquireMemIO(const char *lpszPathName) {
    FIMEMORY *hmem = NULL;
    // 载入一个普通文件
    FREEIMAGEFORMAT fif = FreeImage_GetFileType(
        lpszPathName);
    FIBITMAP *dib = FreeImage_Load(fif, lpszPathName, 0);
    // 打开一个内存流并为其分配内存
    hmem = FreeImage_OpenMemory();
    // 将图象保存到一个内存流中
    FreeImage_SaveToMemory(FIF_PNG, dib, hmem, PNG_DEFAULT);
    ;
    FreeImage_Unload(dib);
    // 从内存流中获取缓冲区
    BYTE *mem_buffer = NULL;
    DWORD size_in_bytes = 0;
    FreeImage_AcquireMemory(hmem, &mem_buffer, &
        size_in_bytes);
    // 将缓冲区内容保存到一个文件流中
    FILE *stream = fopen("buffer.png", "wb");
    if(stream) {

```

```

        fwrite(mem_buffer, sizeof(BYTE), size_in_bytes,
               stream);
        fclose(stream);
    }
    // 关闭内存流并释放其占用的内存空间
    FreeImage_CloseMemory(hmem);
}

```

FreeImage_TellMemory

DLL_API long DLL_CALLCONV FreeImage_TellMemory(FIMEMORY *stream);

获取一个内存指针的当前位置。在入口, *stream* 是目标内存流, 如果调用成功, 函数返回文件的读写指针的当前位置, 否则返回-1。

FreeImage_SeekMemory

DLL_API BOOL DLL_CALLCONV FreeImage_SeekMemory(FIMEMORY *stream, long

offset, int origin);

将内存读写指针移动到指定位置。参数如下:

stream 指向目标内存流的指针

offset 从 *origin* 开始的字节数

origin 初始位置

如果调用成功, 函数返回TRUE, 否则返回FALSE。

FreeImage_SeekMemory 函数将与流 *stream* 相关联的内存文件指针(如果存在的话)移动到一个与 *origin* 相距 *offset* 个字节的新位置。下一个对流的操作在新位置进行。在由 *FreeImage* 管理的一个流上, 下一个操作可以是读操作, 也可以是写操作。变参 *origin* 必须是以下常量之一, 它是在 *STDIO.H* 中定义的(在 *FreeImage.h* 中也有定义):

SEEK_CUR 文件读写指针的当前位置

SEEK_END 文件末

SEEK_SET 文件头

2.12 压缩函数

FreeImage 使用了很多第三方开源库, 以便载入和保存复杂的图象格式。在这些库当中, 象 *ZLib* 这样的库处理内存缓冲区的压缩和解压。因为这个特性可能在很多应用程序中很有用而不仅仅用于图象压缩, *FreeImage* 提供了对这些库的主要功能的一个接口。

目前被支持的只有ZLib压缩。其他压缩算法可能会随着FreeImage将来的发行版本而添加进来。

FreeImage_ZLibCompress

DLL_API DWORD DLL_CALLCONV FreeImage_ZLibCompress(BYTE *target, DWORD target_size, BYTE *source, DWORD source_size);

使用ZLib库将源缓冲区压缩到一个目标缓冲区。在入口处, *target_size* 是目标缓冲区的全部大小, 它必须至少比(*source_size*+12)个字节大0.1%。

函数返回被压缩的缓冲区的实际大小, 或者如果出错的话返回0。

```
BYTE *data = NULL;
DWORD original_size = 0;
// ...
// 压缩数据
DWORD compressed_size = (DWORD)((double) original_size +
    (0.1 * (double) original_size) + 12);
BYTE *compressed_data = (BYTE*)malloc(compressed_size *
    sizeof(BYTE));
compressed_size = FreeImage_ZLibCompress(compressed_data,
    compressed_size, data, original_size);
// 将数据写入磁盘
fwrite(&original_size, sizeof(DWORD), 1, stream);
fwrite(&compressed_size, sizeof(DWORD), 1, stream);
fwrite(compressed_data, sizeof(BYTE), compressed_size,
    stream);
free(compressed_data);
```

FreeImage_ZLibUncompress

DLL_API DWORD DLL_CALLCONV FreeImage_ZLibUncompress(BYTE *target, DWORD target_size, BYTE *source, DWORD source_size);

用ZLib库将源缓冲区解压到一个目标缓冲区。在入口处, *target_size* 是目标缓冲区的全部大小, 它必须足够大以容纳全部解压后的数据。解压后的数据的大小必须在先前由压缩器保存, 并通过这个压缩库范围之外的某些机制传送给解压器。

函数返回被解压的缓冲区的实际大小, 或者如果出错的话返回0。

```

BYTE *data = NULL;
DWORD original_size = 0, compressed_size = 0;
// ...
// 从磁盘上读数据
fread(&original_size, sizeof(DWORD), 1, stream);
fread(&compressed_size, sizeof(DWORD), 1, stream);
data = (BYTE*)malloc(original_size * sizeof(BYTE));
compressed_data = (BYTE*)malloc(compressed_size * sizeof(
    BYTE));
fread(compressed_data, sizeof(BYTE), compressed_size,
    stream);
// 对数据进行解压
DWORD size = 0;
size = FreeImage_ZLibUncompress(data, original_size,
    compressed_data, compressed_size);
assert(size == original_size);
free(compressed_data);

```

FreeImage_ZLibGZip

DLLAPI DWORD DLL_CALLCONV FreeImage_ZLibGZip(BYTE *target, DWORD target_size, BYTE *source, DWORD source_size);

使用ZLib库将源缓冲区压缩到一个目标缓冲区。在入口处, *target_size* 是目标缓冲区的全部大小, 它必须至少比(*source_size*+24)个字节大0.1%。如果函数调用成功, 则目标缓冲区包含一个GZIP兼容的布局。

```

BYTE *data = NULL;
DWORD original_size = 0;
// ...
data = (BYTE*)malloc(original_size * sizeof(BYTE));
// ...
// 初始化大小是原大小加上额外的 8 gzip-header
DWORD compressed_size = (DWORD)((double) original_size +
    (0.1 * (double) original_size) + 24);
BYTE *compressed_data = (BYTE*)malloc(compressed_size *
    sizeof(BYTE));
compressed_size = FreeImage_ZLibGZip(compressed_data,
    compressed_size, data, original_size);
// 现在 compressed_data 包含 'compressed_size' 个字节

```

```
// 的 GZIP 数据
// 将数据写入一个流
// ...
free(compressed_data);
```

这个函数与内存输入/输出函数一起使用时很有用，如果某人用这个来压缩一些东西来在因特网上发送经过gzip压缩的数据的话(这里一个简单的zip布局将不会被许可)。将一个CRC32构造器与现有的zip压缩函数一起使用(参见FreeImage_ZLibCRC32函数)可以获得定制的或更复杂的布局。

FreeImage_ZLibCRC32

```
DLL_API DWORD DLL_CALLCONV FreeImage_ZLibCRC32(DWORD crc, BYTE *
source, DWORD source_size);
```

用ZLib库从source(其字节大小为给定的source_size)中更新一个运行中的crc，并返回更新后的crc(译者注：crc即循环冗余校验)。

如果source为NULL，则该函数返回crc所需要的初始化值，否则返回新的crc值。

FreeImage_ZlibGUnzip

```
DLL_API DWORD DLL_CALLCONV FreeImage_ZlibGUnzip(BYTE *target, DWORD
target_size, BYTE *source, DWORD source_size);
```

用ZLib库将一个经过gzip压缩的源缓冲区解压到一个目标缓冲区。在入口处，target_size是目标缓冲区的全部大小，它必须足够大以容纳全部解压后的数据。解压后的数据的大小必须在先前由压缩器保存，并通过这个压缩库范围之外的某些机制传送给解压器。

函数返回被解压的缓冲区的实际大小，或者如果出错的话返回0。

2.13 帮助函数

FreeImage_IsLittleEndian

```
DLL_API BOOL DLL_CALLCONV FreeImage_IsLittleEndian();
```

如果运行FreeImage的操作系统平台使用Little Endian约定(Intel处理器)，则该函数返回TRUE；而如果使用的是Big Endian约定，则函数返回FALSE(Motorola处理器)。

FreeImage_LookupX11Color

```
DLL_API BOOL DLL_CALLCONV FreeImage_LookupX11Color(const char *szColor,
BYTE *nRed, BYTE *nGreen, BYTE *nBlue);
```

将一个X11颜色名转换为一个相应的RGB值。在入口处，szColor是颜色名。在输出处，nRed、nGreen和nBlue为在[0..255]范围内的颜色分量。如果函数调用成功则返回TRUE，否则返回FALSE。

```
BYTE red , green , blue ;
BOOL bResult ;
bResult = FreeImage_LookupX11Color("papaya_whip" , &red , &
    green , &blue) ;
if(bResult) {
    assert((red == 255) && (green == 239) && (blue == 213))
        ;
}
```

FreeImage_LookupSVGColor

DLL_API BOOL DLL_CALLCONV FreeImage_LookupSVGColor(const char *szColor,
BYTE *nRed, BYTE *nGreen, BYTE *nBlue);

将一个SVG颜色名转换为一个相应的RGB值。在入口处，szColor是颜色名。在输出处，nRed、nGreen和nBlue为在[0..255]范围内的颜色分量。如果函数调用成功则返回TRUE，否则返回FALSE。

```
BYTE red , green , blue ;
BOOL bResult ;
bResult = FreeImage_LookupSVGColor("lemonchiffon" , &red ,
    &green , &blue) ;
if(bResult) {
    assert((red == 255) && (green == 250) && (blue == 205))
        ;
}
```

参考文献

Scalable Vector Graphics (SVG) 1.1 Specification.

[Online]<http://www.w3.org/TR/SVG/types.html>

第三章 元数据函数参考

3.1 介绍

元数据或曰“关于数据的数据”描述诸如图象之类的数据的内容、品质、状态和其他特征。在FreeImage中，元数据是以关键词、自由文本或其他数据类型的方式与一幅图象相联系的信息，这个信息可以是相对直接的信息，诸如作者、创建日期或一个资源(resource)的主题内容等等，它还可以是更复杂和更不易于定义的(例如拍摄环境、用于记录位置的GPS信息等等)。

元数据的储存和获取通常遵循一个标准或规范，用来描述图象的元数据标准的例子有NAA、EXIF、GeoTIFF或Adobe XMP。但是标准并不总是被应用的，很多图象格式使用它们自己的专有方式来储存元数据，要么是以简单的文本字符串、要么是以一种更复杂的方式来储存(例如Adobe Photoshop使用的8BIM)。

虽然很多标准的或专有的格式被用来描述带有元数据的图象，FreeImage给您提供了一个简单的接口来处理所有与图象相关的信息。

FreeImage标签(Tag)

FreeImage使用一种称为标签(Tag)的结构来储存元数据信息。标签的概念起源于TIFF规范，并且因为它的普遍性而被广泛地用于在文件中储存元数据信息。

FreeImage提供了标准TIFF或Exif标签结构的一个增强版本，这个版本的描述如下：

表 3.1:FreeImage FITAG结构

域名	数据类型	描述
key	指向一个C串(char *)的指针	标签域名(在一个元数据模型的内部是唯一的)
description	指向一个C串(char *)的指针	若有标签则为标签的描述, 否则为NULL
id	WORD(16位无符号整数)	若有标签则为标签的ID号, 否则为0
type	WORD (16位无符号整数)	标签的数据类型(参见下面的FREE_IMAGE_MDTYPE)
count	DWORD (32位无符号整数)	标签中类型分量(type components)的数目
length	DWORD (32位无符号整数)	标签值以字节计的长度
value	指向一个32位值的指针(void *)	标签值

给定一个元数据模型(例如Exif、Exif GPS、IPTC/NAA), 标签的键值(或标签域名)在一个元数据模型内部是唯一的。这种唯一性让FreeImage可用该键值来在一个哈希表中给标签建立索引以加速对标签的访问。无论何时在一个元数据模型里储存一个标签, 您就会因而需要同时给该标签提供一个唯一的键值。

一个FreeImage标签可以用来储存任意类型的数据(例如串、整数、双精度数、有理数等等)。表 3.2 中给出了一个完整的FreeImage支持的数据类型列表。例如, 当标签数据类型指示为双精度数类型, 而标签数目为8时, 那么标签数据是一个由8个双精度数组成的数组类型, 它的长度应该是64个字节(8 x sizeof(double))。如果签数据类型指示为有理数类型而长度为48个字节, 那么在标签中有(48字节/ (2 x 4字节)) = 6个有理数。

对于ASCII串, 一个ASCII标签入口的count部分的值包含NULL。

表 3.2:FreeImage标签数据类型

标签数据	类型描述
0 = FIDT_NOTYPE	Placeholder (不使用这个类型)
1 = FIDT_BYTE	8位无符号整数
2 = FIDT_ASCII	包含一个7位ASCII码的8位字节;最后一个字节必须为NUL(二进制0)
3 = FIDT_SHORT	16位(2字节)无符号整数
4 = FIDT_LONG	32位(4字节)无符号整数
5 = FIDT_RATIONAL	两个长整数:前者代表分数中的分子;后者为分母
6 = FIDT_SBYTE	一个8位有符号(补码(twos-complement))整数
7 = FIDT_UNDEFINED	一个8位字节, 根据其域定义的不同可包含任意类型
8 = FIDT_SSHORT	一个16位(2字节)有符号(补码)整数
9 = FIDT_SLONG	一个32位(4字节)有符号(补码)整数
10 = FIDT_SRATIONAL	两个加长整数:前者代表分数中的分子;后者为分母
11 = FIDT_FLOAT	单精度(4字节)IEEE格式
12 = FIDT_DOUBLE	双精度(8字节)IEEE格式
13 = FIDT_IFD	FIDT_IFD数据类型与LONG相同,但它是用于储存位移(offsets)
14 = FIDT_PALETTE	32位(4字节)RGBQUAD

FreeImage元数据模型

当前被FreeImage库识别的元数据模型, 与能载入或保存相应元数据的FreeImage插件一起列于表 3.3 中。

这些元数据模型在附录中有更详细的描述(参见[FreeImage元数据模型](#))。

表 3.3:FreeImage支持的元数据模型

元数据模型 / FIF	FIF_JPEG	FIF_TIFF	FIF_PNG	FIF_GIF
0=FIMD_COMMENTS	R/W	-	R/W	R/W
1=FIMD_EXIF_MAIN	R	-	-	-
2=FIMD_EXIF_EXIF	R	-	-	-
3=FIMD_EXIF_GPS	R	-	-	-
4=FIMD_EXIF_MAKERNOTE	R	-	-	-
5=FIMD_EXIF_INTEROP	R	-	-	-
6=FIMD_IPTC	R	R	-	-
7=FIMD_XMP	R/W	R/W	R/W	-
8=FIMD_GEOTIFF	-	R/W	-	-
9=FIMD_ANIMATION	-	-	-	R/W
10=FIMD_CUSTOM	-	-	-	-

R=读,W=写,-=尚未实现

3.2 标签的创建和销毁

FreeImage_CreateTag

DLL_API FITAG *DLL_CALLCONV FreeImage_CreateTag();

为一个新的FITAG对象分配内存。在不再使用这个对象时，必须调用FreeImage_DeleteTag来销毁它。

?只有当您使用FreeImage_SetMetadata函数时才需要标签创建和销毁函数。

FreeImage_DeleteTag

DLL_API void DLL_CALLCONV FreeImage_DeleteTag(FITAG *tag);

删除一个先前为其分配了内存的FITAG对象。

FreeImage_CloneTag

DLL_API FITAG *DLL_CALLCONV FreeImage_CloneTag(FITAG *tag);

创建并返回一个FITAG对象的一份拷贝。在不再使用这个拷贝时，必须调用FreeImage_DeleteTag来销毁它。

3.3 标签访问器

FreeImage_GetTagKey

DLL_API const char *DLL_CALLCONV FreeImage_GetTagKey(FITAG *tag);

返回标签域名(在一个元数据模型内唯一)。

FreeImage_GetTagDescription

DLL_API const char *DLL_CALLCONV FreeImage_GetTagDescription(FITAG *tag);

如果存在标签说明则将其返回，否则返回NULL。

FreeImage_GetTagID

DLL_API WORD DLL_CALLCONV FreeImage_GetTagID(FITAG *tag);

如果存在标签的ID(标识号)则将其返回，否则返回0。

FreeImage_GetTagType

DLL_API FREE_IMAGE_MDTYPE DLL_CALLCONV FreeImage_GetTagType(FITAG

*tag);

返回标签数据类型(已知的数据类型列表见表 3.2)。

FreeImage_GetTagCount

DLL_API DWORD DLL_CALLCONV FreeImage_GetTagCount(FITAG *tag);

返回标签中分量的数目(以标签类型的单位计算)。例如，当标签数据类型指明为一个双精度数(即一个FIDT_DOUBLE类型)且标签数目为8时，则标签值是一个由8个双精度数组成的数组。

FreeImage_GetTagLength

DLL_API DWORD DLL_CALLCONV FreeImage_GetTagLength(FITAG *tag);

以字节为单位返回标签值的长度。

FreeImage_GetTagValue

DLL_API const void *DLL_CALLCONV FreeImage_GetTagValue(FITAG *tag);

返回标签值。根据调用FreeImage_GetTagType和FreeImage_GetTagCount的不同结果，用户自己已经合适于完成正确地解释这些字节的任务。

FreeImage_SetTagKey

DLL_API BOOL DLL_CALLCONV FreeImage_SetTagKey(FITAG *tag, const char *key);

设置标签域名(这是永远需要的，在一个元数据模型内必须是唯一的)。如果调用成功该函数返回TRUE，否则返回FALSE。

FreeImage_SetTagDescription

DLL_API BOOL DLL_CALLCONV FreeImage_SetTagDescription(FITAG *tag, const

char *description);

设置(通常是可选的)标签说明。如果调用成功该函数返回TRUE，否则返回FALSE。

?标签说明从来不储存在文件中。FreeImage管理维护内部的一个关于所有已知标签的表，如果这些标签的说明存在，则连说明一起都在表内。无论何时您读取一个已知标签时，FreeImage可以通过使用 *FreeImage_GetTagDescription* 来

给出标签说明(假设标签对库来说是已知的)。然而,您在储存一个标签时却永远不需要提供一个标签说明。

FreeImage_SetTagID

DLL_API BOOL DLL_CALLCONV FreeImage_SetTagID(FITAG *tag, WORD id);
 设置(通常是可选的)标签的ID(标识号)。如果调用成功该函数返回TRUE,否则返回FALSE。

FreeImage_SetTagType

DLL_API BOOL DLL_CALLCONV FreeImage_SetTagType(FITAG *tag, FREE_IMAGE_MDTYPE type);
 设置标签数据类型(这永远是必要的,现有的数据类型列表见表 3.2)。如果调用成功该函数返回TRUE,否则返回FALSE。

FreeImage_SetTagCount

DLL_API BOOL DLL_CALLCONV FreeImage_SetTagCount(FITAG *tag, DWORD count);
 设置标签中的数据个数(这永远是必要的,以标签类型的单位来表示)。如果调用成功该函数返回TRUE,否则返回FALSE。

FreeImage_SetTagLength

DLL_API BOOL DLL_CALLCONV FreeImage_SetTagLength(FITAG *tag, DWORD length);
 设置标签值的以字节为单位的长度(这永远是必要的)。如果调用成功该函数返回TRUE,否则返回FALSE。

FreeImage_SetTagValue

DLL_API BOOL DLL_CALLCONV FreeImage_SetTagValue(FITAG *tag, const void *value);
 设置标签值(这永远是必要的)。如果调用成功该函数返回TRUE,否则返回FALSE。

!在标签数据类型、标签数目和标签长度等值被填写后必须调用该函数。否则您将不可能成功地调用 *FreeImage_SetMetadata*。

3.4 元数据迭代子

FreeImage_FindFirstMetadata

DLL_API FIMETADATA *DLL_CALLCONV FreeImage_FindFirstMetadata(FREE_IMAGE_MDMODEL model, FIBITMAP *dib, FITAG **tag);
 提供了与在 *model* 变参中指定的元数据模型相匹配的标签的第一个实例的信息。

如果调用成功，FreeImage_FindFirstMetadata返回一个标识与`model`说明相匹配的标签组的唯一的句柄，该句柄可以在接下来对FreeImage_FindNextMetadata或FreeImage_FindCloseMetadata的调用中使用。

当输入dib中不存在元数据模型时，FreeImage_FindFirstMetadata返回NULL。

FreeImage_FindNextMetadata

DLL_API BOOL DLL_CALLCONV FreeImage_FindNextMetadata(FIMETADATA *mdhandle, FITAG **tag);

寻找下一个与先前对FreeImage_FindFirstMetadata的调用中的`model`变参相匹配的标签，如果它存在的话。然后，相应地改变标签对象的内容。

如果调用成功，FreeImage_FindNextMetadata返回TRUE，否则返回FALSE，表示找不到相匹配的标签。

FreeImage_FindCloseMetadata

DLL_API void DLL_CALLCONV FreeImage_FindCloseMetadata(FIMETADATA *mdhandle);

关闭指定的元数据搜索句柄并释放相关资源。

```
// 本段代码假设已载入由一个叫做"dib" 的变量代表的位图
FITAG *tag = NULL;
FIMETADATA *mdhandle = NULL;
mdhandle = FreeImage_FindFirstMetadata(FIMD_EXIF_MAIN,
    dib, &tag);
if(mdhandle) {
    do {
        // 处理标签
        printf("%s\n", FreeImage_GetTagKey(tag));
        // ...
    } while(FreeImage_FindNextMetadata(mdhandle, &tag));
}
FreeImage_FindCloseMetadata(mdhandle);
```

3.5 元数据访问器

FreeImage_GetMetadata

DLL_API BOOL DLL_CALLCONV FreeImage_GetMetadata(FREE_IMAGE_MDMODEL model, FIBITMAP *dib, const char *key, FITAG **tag);

获取与一个dib相连的元数据。在入口处, *model*是要寻找的元数据模型, *dib*是包含元数据的图象, *key*是元数据域名(在一个元数据模型内唯一), 而*tag*是由函数返回的一个FITAG结构。

当搜索的标签不存在时, 标签对象保持不变, 函数返回FALSE, 否则数返回TRUE并且标签对象被填(populated)以元数据信息。

```
// 本段代码假设已载入由一个叫做 "dib" 的变量代表的位图
// 获取相机模型
FITAG *tagMake = NULL;
FreeImage_GetMetadata(FIMD_EXIF_MAIN, dib, "Make", &
    tagMake);
if(tagMake != NULL) {
    // 这里我们知道(根据 Exif 规范) tagMake 是一个 C 串
    printf("Camera_model: %s\n", (char*)
        FreeImage_GetTagValue(tagMake));
}
```

?当由*FreeImage_GetMetadata*或元数据迭代函数返回的一个标签被改动时, 所作改动将会被应用到与位图相连的相应标签。保存位图将因而也同时保存改动过的标签(假设FreeImage库可以保存相应的元数据模型)。

FreeImage_SetMetadata

```
DLL_API BOOL DLL_CALLCONV FreeImage_SetMetadata(FREE_IMAGE_MDMODEL
model, FIBITMAP *dib, const char *key, FITAG *tag);
```

将一个新的FreeImage标签与一个dib相连。在入口处, *model*是用来保存标签的元数据模型, *dib*是目标图象, *key*是标签域名, 而*tag*是要与之相连的FreeImage标签。

若tag为NULL则元数据被删除。

若key和tag均为NULL则元数据模型被删除。

如果调用成功, 函数返回TRUE, 否则返回FALSE。

?FreeImage用来对一个标签进行索引的标签域名(或标签键值)由元数据模型规范给定(例如EXIF规范或Adobe XMP规范)。

```

char *xmp_profile = NULL;
DWORD profile_size = 0;
// ...
// 以下假设在一个 (null 结束的) 变量 "xmp_profile" 中储存了
// 一个 XML 包, 其大小已由 "profile_size" 给定并包含 NULL 值
// 创建一个标签
FITAG *tag = FreeImage_CreateTag();
if(tag) {
    // 填写标签成员
    FreeImage_SetTagKey(tag, "XMLPacket");
    FreeImage_SetTagLength(tag, profile_size);
    FreeImage_SetTagCount(tag, profile_size);
    FreeImage_SetTagType(tag, FIDT_ASCII);
    // 标签值必须在填写标签数据类型、标签数和标签长度之后储存
    FreeImage_SetTagValue(tag, xmp_profile);
    // 储存标签
    FreeImage_SetMetadata(FIMD_XMP, dib,
        FreeImage_GetTagKey(tag), tag);
    // 销毁标签
    FreeImage_DeleteTag(tag);
}

```

3.6 元数据帮助函数

FreeImage_GetMetadataCount

DLL_API unsigned DLL_CALLCONV FreeImage_GetMetadataCount(FREE_IMAGE_
MDMODEL model, FIBITMAP *dib);

返回在与输入dib相连的元数据模型`model`中包含的标签数。

```

unsigned count;
if(count = FreeImage \_GetMetadataCount(FIMD_EXIF_GPS,
    dib)) {
    // 处理 GPS 数据
}

```


FreeImage_TagToString

DLL_API const char* DLL_CALLCONV FreeImage_TagToString(FREE_IMAGE_ MDMODEL model, FITAG *tag, char *Make FI_DEFAULT(NULL));

将一个FreeImage标签结构转换为一个代表被解译的标签值的串。标签值是根据元数据模型规范被解译的,例如,设想一个从FIMD_EXIF_EXIF元数据模型中分离出来的标签,其标识号(ID)为0x9209而键值为“Flash”,则标签值为0x0005,函数将返回“Strobe return light not detected”。

在入口处, *model*是我们从其中分离出标签的元数据模型, *tag*是要解译的FreeImage标签,而*Make*是相机模型。这最后的参数目前没有被FreeImage库用到,但在将来会用它来解译相机制作者标注(FIMD_EXIF_MAKERNOTE元数据模型)。

```
// 本段代码假设已载入由一个叫做"dib" 的变量代表的位图
FITAG *tag = NULL;
FIMETADATA *mdhandle = NULL;
mdhandle = FreeImage_FindFirstMetadata(model, dib, &tag);
if(mdhandle) {
    do {
        // 将标签值转换为一个串
        const char *value = FreeImage_TagToString(model, tag);
        // 打印标签, 注意大多数标签没有说明, 特别是当没有提供元数据规范时
        if(FreeImage_GetTagDescription(tag)) {
            cout << "key:_:" << FreeImage_GetTagKey(tag) << ";_
                value:_:" << value << ";_description:_:" <<
                FreeImage_GetTagDescription(tag) << "\n";
        } else {
            cout << "key:_:" << FreeImage_GetTagKey(tag) << ";_
                value:_:" << value << ";description:_:(none)\n";
        }
    } while(FreeImage_FindNextMetadata(mdhandle, &tag));
}
FreeImage_FindCloseMetadata(mdhandle);
```

第四章 工具包函数参考

4.1 旋转和翻转

FreeImage_RotateClassic

1 8 24 32

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_RotateClassic(FIBITMAP *dib,
double angle);

该函数用三种剪切(shear)来旋转一个1位、8位灰度位图或一个24位、32位彩色图象，旋转角度由angle参数以度为单位指定。旋转是围绕图象中心进行的。被旋转的图象保持源图象的大小和长宽比(通常目标图象比较大)，所以应该在将一幅图象旋转90°、180°或270°时使用该函数。

```
// 本段代码假设已载入由一个叫做" dib " 的变量代表的位图
// 执行一个 90 ° 旋转 (CCW 旋转 )
FIBITMAP *rotated = FreeImage_RotateClassic( dib , 90 );
```

!对于1位位图，旋转限于90° 的整数倍角度(例如 - 90、90、180、270)，对于其他角度返回一个NULL值。

?附录中给出了该函数的一个演示(参见[旋转函数的使用](#))。

参考文献

Paeth A., A Fast Algorithm for General Raster Rotation. Graphics Gems, p. 179, Andrew Glassner editor, Academic Press, 1990. Yariv E., High quality image rotation (rotate by shear).

[Online] <http://www.codeproject.com/bitmap/rotatebyshear.asp>

FreeImage_RotateEx

8 24 32

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_RotateEx(FIBITMAP *dib, double
angle, double x_shift, double y_shift, double x_origin, double y_origin, BOOL use_mask);

这个函数用一种3次(立方)B样条来对一幅8位灰度位图、24位或32位图象执行旋转和/或平移(translation)。被旋转的图象将会具有和源图象相同的宽度和高度,因此该函数更适合于计算机显示和自动化(robotics)。

旋转角度由angle参数以度为单位指定。水平和垂直平移(以像素为单位)由x_shift参数和y_shift参数指定。旋转围绕着由x_origin和y_origin(同样以像素为单位)指定的中心进行。当use_mask被设为TRUE时,与图象无关的部分被设为黑色,否则用反射技术来填充无关的像素。

```
// 本段代码假设已载入由一个叫做"dib"的变量代表的位图
// 将图象围绕着图象区域中心进行旋转
double x_orig = FreeImage_GetWidth(dib) / (double)2;
double y_orig = FreeImage_GetHeight(dib) / (double)2;
// 使用掩码执行一个15°旋转(无平移)CCW
FIBITMAP *rotated = FreeImage_RotateEx(dib, 15, 0, 0,
    x_orig, y_orig, TRUE);
```

?附录中给出了该函数的一个演示(参见[旋转函数的使用](#))。

参考文献

Philippe Thévenaz, Spline interpolation, a C source code implementation.

[Online] <http://bigwww.epfl.ch/thevenaz/>

Unser M., Splines: A Perfect Fit for Signal and Image Processing. IEEE Signal Processing Magazine, vol. 16, no. 6, pp. 22-38, November 1999.

Unser M., Aldroubi A., Eden M., B-Spline Signal Processing: Part I—Theory. IEEE Transactions on Signal Processing, vol. 41, no. 2, pp. 821-832, February 1993.

Unser M., Aldroubi A., Eden M., B-Spline Signal Processing: Part II—Efficient Design and Applications. IEEE Transactions on Signal Processing, vol. 41, no. 2, pp. 834-848, February 1993.

FreeImage_FlipHorizontal

DLL_API BOOL DLL_CALLCONV FreeImage_FlipHorizontal(FIBITMAP *dib);
沿垂直轴将输入dib水平翻转。

FreeImage_FlipVertical

DLL_API BOOL DLL_CALLCONV FreeImage_FlipVertical(FIBITMAP *dib);
沿水平轴将输入dib垂直翻转。

FreeImage_JPEGTransform

```
DLL_API BOOL DLL_CALLCONV FreeImage_JPEGTransform(const char *src_file, const
char *dst_file, FREE_IMAGE_JPEG_OPERATION operation, BOOL perfect
FI_DEFAULT(FALSE));
```

对一个JPEG文件执行无损旋转或翻转。在入口处，*src_file*是源JPEG文件而*dst_file*是目的JPEG文件。对源文件和目的文件使用同一个文件是允许的：源文件将被变换(transform)并被覆盖。*operation*参数指定要应用的变换的种类。以下是可能的变换：

表 4.1:FREE_IMAGE_JPEG_OPERATION常量

操作	描述
FIJPEG_OP_NONE	无变换(不做任何事)
FIJPEG_OP_FLIP_H	水平翻转
FIJPEG_OP_FLIP_V	垂直翻转
FIJPEG_OP_TRANSPOSE	沿从左上方轴横跨到右下方轴的方向变换
FIJPEG_OP_TRANSVERSE	沿从右上方轴横跨到左下方轴的方向变换
FIJPEG_OP_ROTATE_90	90度顺时针旋转
FIJPEG_OP_ROTATE_180	180度旋转
FIJPEG_OP_ROTATE_270	270度顺时针旋转(或90度ccw)

FreeImage_JPEGTransform通过对压缩过的数据(DCT系数)进行重排来运行，甚至不需要对图象进行完全解码。因此，它的变换是无损的：根本没有图象质量的降损(degradation)，而如果您用FreeImage_Load接着用FreeImage_Save来实现同样的转换的话，情况可就不是这样了。

FIJPEG_OP_TRANSPOSE变换没有关于图象尺寸的限制。如果图象的尺寸不是iMCU大小(通常是8或16像素)的倍数的话，其他变换的操作就很古怪了，因为它们只能以所期望的方式对整个DCT系数数据块进行变换。

函数的默认行为是这样设计的：当对尺寸为奇数的图象进行变换时，丢弃任何未经变换的边缘像素，而不是在一个变换后的图象的右边和/或底部边缘有着外观奇怪的小长条。很明显，这种变换在应用于尺寸为奇数的图象时是不可逆的，所以严格说来这种操作并不是无损的。

为了避免有损变换，可将*perfect*参数设置为TRUE。使用该参数时，任何不可逆的变换都被避免，并扔出一个错误消息(可以用*FreeImage_SetOutputMessage*来把它记录在日志中)，函数将返回FALSE。

4.2 过采样/减像素采样

FreeImage_Rescale

8 24 32

16 48 64

```
DLL_API FIBITMAP * DLL_CALLCONV FreeImage_Rescale(FIBITMAP *dib, int
dst_width, int dst_height, FREE_IMAGE_FILTER filter);
```

该函数执行一幅灰度或RGB(A)图象的重采样(或按比例绘制、缩放)以使其宽度和高度变为所期望的值。当不能处理位深度或没有足够的内存时(这可能发生于图象非常大的情况下)返回NULL值。

重采样牵涉到图象像素大小的改变(因而也牵涉到图象显示大小的改变)。当您进行减像素采样(减少像素点数)时, 信息被从图象中删除; 当您进行过采样(或增加像素点数)时, 新的像素点被基于现存像素点的颜色值添加进来。像素点如何被添加或被删除, 取决于您指定的一个插值滤镜。

以下滤镜可以用作重采样滤镜:

表 4.2:IMAGE_FILTER常量

滤镜标志	描述
FILTER_BOX	箱形(Box), 脉冲,傅立叶窗,1阶 (常量) B样条
FILTER_BILINEAR	双线性(Bilinear)滤镜
FILTER_BSPLINE	4阶(立方)B样条
FILTER_BICUBIC	Mitchell-Netravali双参数立方(two-param cubic)滤镜
FILTER_CATMULLROM	Catmull-Rom样条,Overhauser样条
FILTER_LANCZOS3	Lanczos-windowed sinc滤镜

?在附录中给出了关于这些滤镜的一些提示(参见选择正确的重采样滤镜)。

参考文献

Paul Heckbert, C code to zoom raster images up or down, with nice filtering. UC Berkeley, August 1989.[Online]

<http://www-2.cs.cmu.edu/afs/cs.cmu.edu/Web/People/ph/heckbert.html>

Hou H.S., Andrews H.C., Cubic Splines for Image Interpolation and Digital Filtering. IEEE Trans. Acoustics, Speech, and Signal Proc., vol. ASSP-26, no. 6, pp. 508-517, Dec. 1978.

Glassner A.S., Principles of digital image synthesis. Morgan Kaufmann Publishers, Inc, San Francisco, Vol. 2, 1995.

Mitchell Don P., Netravali Arun N., Reconstruction filters in computer graphics. In John Dill, editor, Computer Graphics (SIGGRAPH '88 Proceedings), Vol. 22, No. 4, pp. 221-228, August 1988.

Keys R.G., Cubic Convolution Interpolation for Digital Image Processing. IEEE

Trans. Acoustics, Speech, and Signal Processing, vol. 29, no. 6, pp. 1153-1160, Dec. 1981.

4.3 颜色处理

FreeImage使用RGB(A)色彩模型来代表内存中的彩色图象。一个8位灰度图象具有单独通道，通常称为黑白通道。一个24位图象由三个8位通道组成，分别为红色、绿色和蓝色通道。对于32位图象，第四个通道—叫做alpha通道—被用来创建和储存掩码，掩码让您可以对一幅图象的指定部分进行处理、分离和保护。与其他通道不同，alpha通道在物理意义上来说并不传达颜色信息。

在FreeImage中使用的颜色处理函数让您可以修改一个指定通道的直方图，这种变换称为点操作，它可以用于一幅图象的亮度、对比度或gamma值的调整以执行图象优化(例如直方图均衡，非线性调整)，甚至还可以用来翻转(invert)一幅图象或给图象设置阈值。

目前在FreeImage中定义了以下通道：

表 4.3:FREE_IMAGE_COLOR_CHANNEL常量

通道标志	描述
FICC_RGB	函数应用于红色、绿色和蓝色通道
FICC_RED	函数仅应用于红色通道
FICC_GREEN	函数仅应用于绿色通道
FICC_BLUE	函数仅应用于蓝色通道
FICC_ALPHA	函数仅应用于alpha通道
FICC_BLACK	函数应用于黑白通道
FICC_REAL	复图象: 函数应用于实部
FICC_IMAG	复图象: 函数应用于虚部
FICC_MAG	复图象: 函数应用于模
FICC_PHASE	复图象: 函数应用于辐角(phase)

FreeImage_AdjustCurve

8 24 32

DLL_API BOOL DLL_CALLCONV FreeImage_AdjustCurve(FIBITMAP *dib, BYTE *LUT, FREE_IMAGE_COLOR_CHANNEL channel);

在一个8位、24位或32位图象上根据一个查找表(Lookup table (LUT))中的值来执行直方图变换，该变换根据以下等式改变一个或多个通道：

$$channel(x, y) = LUT[channel(x, y)]$$

'LUT'的大小假设为256。要对其进行变换的颜色通道由channel参数指定。变换是这样进行的：

- 8位位图：如果图象有一个调色板，则LUT被应用到该调色板，否则LUT被应用到灰度值。channel参数未被用到。
- 24位位图：如果通道等于FICC_RGB，则同样的LUT被应用到每个颜色平面(R、G、和B)，否则LUT只被应用到指定的通道(R、G、B或A)。

如果调用成功，函数返回TRUE，否则返回FALSE(例如当无法处理源dib的位深度时)。

FreeImage_AdjustGamma

8 24 32

DLL_API BOOL DLL_CALLCONV FreeImage_AdjustGamma(FIBITMAP *dib, double gamma);

在一幅8位、24位或32位图象上执行gamma校正。gamma参数代表要使用的gamma值(gamma >0)，1.0代表图象原样不变，小于1则图象变深，大于1则图象变浅。

如果调用成功，函数返回TRUE；当gamma小于或等于0、或无法处理源dib的位深度时返回FALSE。

FreeImage_AdjustBrightness

8 24 32

DLL_API BOOL DLL_CALLCONV FreeImage_AdjustBrightness(FIBITMAP *dib, double percentage);

将一幅8位、24位或32位图象的亮度作一定数量的调整，该数量由percentage参数给定，其值为[-100..100]之间的一个数。0值意味着无变化，小于0使图象变深而大于0使图象变浅。

如果调用成功，函数返回TRUE，否则返回FALSE(例如当无法处理源dib的位深度时)。

FreeImage_AdjustContrast

8 24 32

DLL_API BOOL DLL_CALLCONV FreeImage_AdjustContrast(FIBITMAP *dib, double percentage);

将一幅8位、24位或32位图象的对比度作一定数量的调整，该数量由percentage参数给定，其值为[-100..100]之间的一个数。0值意味着无变化，小于0使图象对比度减小而大于0使图象对比度增加。

如果调用成功，函数返回TRUE，否则返回FALSE(例如当无法处理源dib的位深度时)。

FreeImage_Invert

1 4 8 16 24 32

DLL_API BOOL DLL_CALLCONV FreeImage_Invert(FIBITMAP *dib);

反转(Invert)每个像素数据。

FreeImage_GetHistogram

8 24 32

DLL_API BOOL DLL_CALLCONV FreeImage_GetHistogram(FIBITMAP *dib, DWORD *histo, FREE_IMAGE_COLOR_CHANNEL channel FI_DEFAULT(FICC_BLACK));

计算图象直方图。对于24位和32位图象，直方图可以从红色通道、绿色通道、蓝色通道和黑白通道中计算出来。对于8位图象，直方图从黑白通道中计算出来。其他位深度不被支持(函数不做任何事情且返回FALSE)。histo参数必须由驱动FreeImage的应用程序分配内存，假设它的大小等于256。

4.4 通道处理

FreeImage_GetChannel

24 32

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_GetChannel(FIBITMAP *dib, FREE_IMAGE_COLOR_CHANNEL channel);

获取一幅8位、24位或32位图象的红色、绿色、蓝色或alpha通道。dib是待处理的输入位图，而channel是要分离的颜色通道。如果函数调用成功，则返回分离出来的通道，否则返回NULL。

FreeImage_SetChannel

24 32

DLL_API BOOL DLL_CALLCONV FreeImage_SetChannel(FIBITMAP *dib, FIBITMAP *dib8, FREE_IMAGE_COLOR_CHANNEL channel);

将一个8位dib插入一幅24位或32位图象中。dib8和dib必须具有相同的宽度和高度。dib是要修改的目标图象(24位或32位)，dib8是要插入的图象，而channel是要替换的颜色通道。如果调用成功，函数返回TRUE，否则返回FALSE。

FreeImage_GetComplexChannel

2x64

DLL_API FIBITMAP * DLL_CALLCONV FreeImage_GetComplexChannel(FIBITMAP *src, FREE_IMAGE_COLOR_CHANNEL channel);

获取一幅复图象(类型为FIT_COMPLEX的图象)的实部、虚部、模或辐角。若函数调用成功,则将分离出来的通道作为一个FIT.DOUBLE图象返回,否则返回NULL。

FreeImage_SetComplexChannel

2x64

DLL_API BOOL DLL_CALLCONV FreeImage_SetComplexChannel(FIBITMAP *dst, FIBITMAP *src, FREE_IMAGE_COLOR_CHANNEL channel);

设置一幅复图象(类型为FIT_COMPLEX的图象)的实部、虚部、模或辐角。src和dst都必须具有相同的宽度和高度。在入口处,dst是待修改的图象(类型为FIT_COMPLEX的图象),而src是要替换的通道(类型为FIT.DOUBLE的图象)。若函数调用成功,则返回TRUE,否则返回NULL。

4.5 复制/粘贴/合成例程

FreeImage_Copy

DLL_API FIBITMAP *DLL_CALLCONV FreeImage_Copy(FIBITMAP *dib, int left, int top, int right, int bottom);

复制当前dib图象的一部分。由(left, top, right, bottom)参数定义的矩形首先被规范化,这样左边的坐标比右边的小而顶端坐标比底部的小。然后,返回的位图被定义为宽度等于(right - left)而高度等于(bottom - top)。

函数的参数如下:

left:指定被剪切(cropped)矩形的左边位置。

top:指定被剪切矩形的顶端位置。

right:指定被剪切矩形的右边位置。

bottom:指定被剪切矩形的底部位置。

若调用成功,函数返回子图象,否则返回NULL。

FreeImage_Paste

1 4 8 16 24 32

DLL_API BOOL DLL_CALLCONV FreeImage_Paste(FIBITMAP *dst, FIBITMAP *src, int left, int top, int alpha);

将一个子图象与当前dib图象进行Alpha合成或合并。dst图象的位深度必须大于等于src图象的位深度。对src的上面部分的处理是在内部进行的，并不对src作修改。被支持的dst的位深度等于1、4、8、16、24或32。函数的参数如下：

dst :目标图象。

src : 源图象。

left :指定子图象的左边位置。

top : 指定子图象的顶端位置。

alpha : Alpha合成因子。若alpha=0..255, 则对源图象和目标图象进行Alpha合成; 若alpha > 255, 则源图象被合并到目标图象。

若调用成功, 函数返回TRUE, 否则返回NULL。

FreeImage_Composite

8 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_Composite(FIBITMAP *fg, BOOL
useFileBkg FI_DEFAULT(FALSE), RGBQUAD *appBkColor FI_DEFAULT(NULL), FIBITMAP
*bg
FI_DEFAULT(NULL));
```

该函数将一个有透明前景的图象与一个单色背景或背景图象进行混合。在入口处, fg定义了前景图象和透明掩码(对于8位dib作为透明度表、或对于32位dib作为一个alpha通道, 隐含于前景图象中)。

合成样本值的计算式是:

$$\text{output} = \alpha * \text{foreground} + (1 - \alpha) * \text{background}$$

其中alpha和输入及输出样本值由0到1范围内的小数表达。对于彩色图象, 计算是对R、G和B(红色、绿色和蓝色)样本分别单独进行的。

以下伪代码演示了其他参数的内部使用:

```
if (useFileBkg && FreeImage_HasBackgroundColor (fg)) {
    // 将文件背景色用作单独背景色
} else { // 没有文件背景色 ... 使用应用程序的背景色?
    if (appBkColor) { // 将应用程序的背景色用作单独背景色
    }
    // 没有应用程序背景色 ... 使用一幅背景图象?
    else if (bg) { // 将 bg 用作背景图象
        // bg 必须是一个与 fg 有着相同宽度和高度的 24 位图象
    }
    else { // 缺省的情况, 用一个棋盘作背景图象
    }
}
```

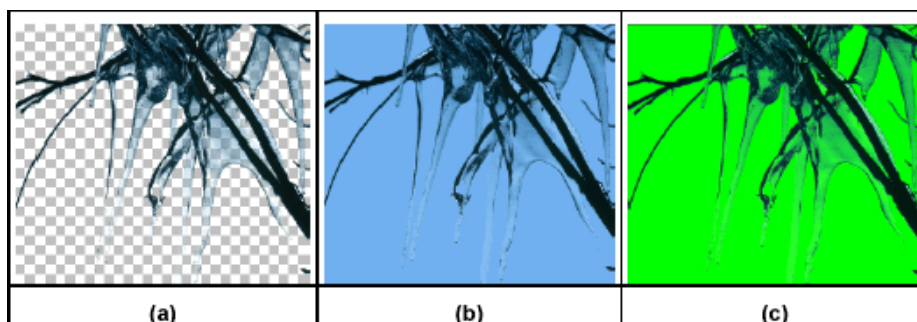


图 4.1:FreeImage_Composite 函数演示

图4.1给出了FreeImage_Composite函数的一个演示。这个简单的图象是一个有着鲜蓝色文件背景的8位透明PNG图象。每幅图是用以下调用生成的：

```
FIBITMAP *fg = FreeImage_Load(FIF_PNG, "test.png",
    PNG_DEFAULT);
// image (a) : 使用一个检查板背景
FIBITMAP *display_dib_a = FreeImage_Composite(fg);
// image (b) : 使用图象文件背景, 如果存在的话
FIBITMAP *display_dib_b = FreeImage_Composite(fg, TRUE);
// image (c) : 使用一个用户指定的背景
RGBQUAD appColor = { 0, 255, 0, 0 };
FIBITMAP *display_dib_c = FreeImage_Composite(fg, FALSE,
    &appColor);
```

参考文献

Portable Network Graphics (PNG) Specification (Second Edition).

[Online]<http://www.w3.org/TR/PNG/>

表 4.4:FreeImage允许的位图文件格式

FORMAT	DESCRIPTION	EXTENSIONS	SUPPORT LOADING	SUPPORT WRITING	SUPPORT OPTIONS	SUPPORT TRANSPARENCY		SUPPORT ICC PROFILES	EXPORTED BITDEPTH								EXPORTED TYPE								
						8	32		1	4	8	16	24	32	FTL_BITMAP	FTL_UINT16	FTL_UINT16	FTL_UINT32	FTL_INT32	FTL_FLOAT	FTL_DOUBLE	FTL_COMPLEX	FTL_RGB16	FTL_RGBA16	FTL_RGBF
FIF_BMP	Windows 或 OS/2 位图	bmp	•	•	•	-	•	-	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
FIF_CUT	Dr. Halo	cut	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
FIF_DDS	DirectX Surface	dds	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
FIF_GIF	Graphics Interchange Format	gif	•	•	•	•	-	-	-	•	-	-	-	•	-	-	-	-	-	-	-	-	-	-	-
FIF_HDR	High Dynamic Range	hdr	•	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	•	-
FIF_ICO	Windows Icon	ico	•	•	•	-	•	-	•	•	•	•	•	•	•	•	-	-	-	-	-	-	-	-	-
FIF_JFF	IFF Interleaved Bitmap	iff_lbm	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
FIF_JNG	JPEG Network Graphics	jng	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
FIF_JPEG	JPEG - JFIF Compliant	jpg,jif,jpeg,jpe	•	•	•	-	•	-	-	•	-	•	-	•	-	-	-	-	-	-	-	-	-	-	-
FIF_KOALA	C64 Koala Graphics	koa	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
FIF_MNG	Multiple Network Graphics	mng	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
FIF_PBM	Portable Bitmap (ASCII)	pbm	•	•	•	-	-	-	-	-	-	-	-	•	-	-	-	-	-	-	-	-	-	-	-
FIF_PBMRAW	Portable Bitmap (RAW)	pbm	•	•	•	-	-	-	•	-	-	-	-	•	-	-	-	-	-	-	-	-	-	-	-
FIF_PCD	Kodak PhotoCD	pcd	•	-	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
FIF_PCX	Zsoft Paintbrush	pcx	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
FIF_PGM	Portable Greymap (ASCII)	pgm	•	•	•	-	-	-	-	•	-	-	-	•	-	-	-	-	-	-	-	-	-	-	-
FIF_PGMRAW	Portable Greymap (RAW)	pgm	•	•	•	-	-	-	-	•	-	-	-	•	-	-	-	-	-	-	-	-	-	-	-
FIF_PNG	Portable Network Graphics	png	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
FIF_PPM	Portable Pixelmap (ASCII)	ppm	•	•	•	-	-	-	-	-	-	-	-	•	-	-	-	-	-	-	-	-	-	-	-
FIF_PPMRAW	Portable Pixelmap (RAW)	ppm	•	•	•	-	-	-	-	-	-	-	-	•	-	-	-	-	-	-	-	-	-	-	-
FIF_PSD	Adobe Photoshop	psd	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
FIF_RAS	Sun Raster Image	ras	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
FIF_TARGA	Truevision Targa	tga,targa	•	•	•	•	•	-	•	•	•	•	•	•	•	-	-	-	-	-	-	-	-	-	-
FIF_TIFF	Tagged Image File Format	tif,tiff	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	-
FIF_WBMP	Wireless Bitmap	wap,wbmp,wbm	•	•	-	-	-	•	-	-	-	-	-	•	-	-	-	-	-	-	-	-	-	-	-
FIF_XBM	X11 Bitmap Format	xbm	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
FIF_XPM	X11 Pixmap Format	xpm	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

图例	
Yes :	•
No :	-

附录 A 选择正确的重采样滤镜

重采样滤镜的效果高度依赖于待重定尺寸的图象的物理特性。然而可以证明，以下的提示有助于判定使用何种滤镜。

A.1 箱形(Box)滤镜

从计算的角度看，箱形缩放(Box scaling)是最简单、最快捷的缩放算法。人们起了各种不同的名字来表示这种简单的算法核心，这些名字包括箱形滤波、取样并保持(sample-and-hold)函数、脉冲函数、傅立叶窗、1阶(常量)B样条和最近像素算法。这种技术通过像素复制实现了放大，而通过稀疏点采样实现了对图象的缩小。对于大尺寸图象，箱形插值产生块状外观(blocky appearance)的图象，另外还可能产生最多为1个半像素的平移误差。这些问题使得这种技术不适合于对重建图象的精度有要求的情况。

A.2 双线性(Bilinear)滤镜

双线性是速度第二快的缩放函数。它使用线性插值法来确定输出图象。双线性缩放以中等代价给大多数缩放因子相对小(4X或更小)的应用提供了适当好的结果。虽然如此，通常人们还需要更高的保真度，因此提出了更复杂的滤波器。

A.3 B样条滤镜

B样条滤镜产生最光滑的输出,但却容易掩盖细节(but tends to smooth over fine details)。该函数需要与Mitchell和Netravali的Bicubic滤镜相同的处理时间。对于需要最光滑输出的应用，B样条滤镜是推荐的算法。

A.4 二维立方(Bicubic)滤镜

Mitchell和Netravali的二维立方滤镜是高级的参数化缩放滤镜。它用一个三次函数来产生非常光滑的输出而同时又保留动态范围和锐度(sharpness)。二维

立方滤镜花费的处理时间大约为双线性滤镜的两倍。这种滤镜可以用于任何缩放应用中，特别是在缩放因子为2X或更高的情况下。

A.5 Catmull-Rom滤镜

在使用Mitchell-Netravali滤镜时您必须设置b和c 两个参数，使得 $b + 2c = 1$ ，以便使用数字最精确的滤镜。二维立方滤镜使用默认值($b = 1/3, c = 1/3$)，这是Mitchell和Netravali在营造人类最舒适视觉的主观测试时推荐的值。当 $b = 0$ 时c取得最大值 $c = 0.5$ ，这就是Catmull-Rom样条，也是对于追求锐度的情况的好建议。Catmull-Rom滤镜被普遍认为是最佳立方(三次)插值滤镜而得到采纳。

A.6 Lanczos滤镜

Lanczos算法使用基于sinc函数的滤镜。这是理论上最准确的滤镜，对于不具有锐度转换的照片图象能产生最佳输出。然而因为走样的缘故，Lanczos滤镜将会产生波纹现象，特别是对于文本块。Lanczos算法需要花费的处理时间为双线性算法的三倍。除非是在极少数使用有带宽限制、不具有清晰边沿的照片图象的应用中，否则不推荐使用Lanczos算法。

附录 B 各种重采样方法的比较

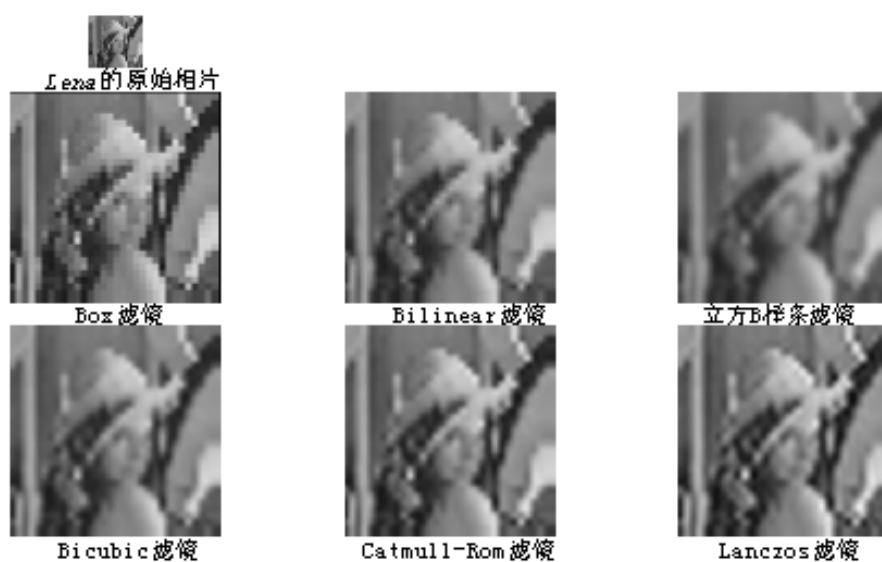


图 B.1:不同重采样滤镜在Lena的相片从32x32增加到400%时的比较

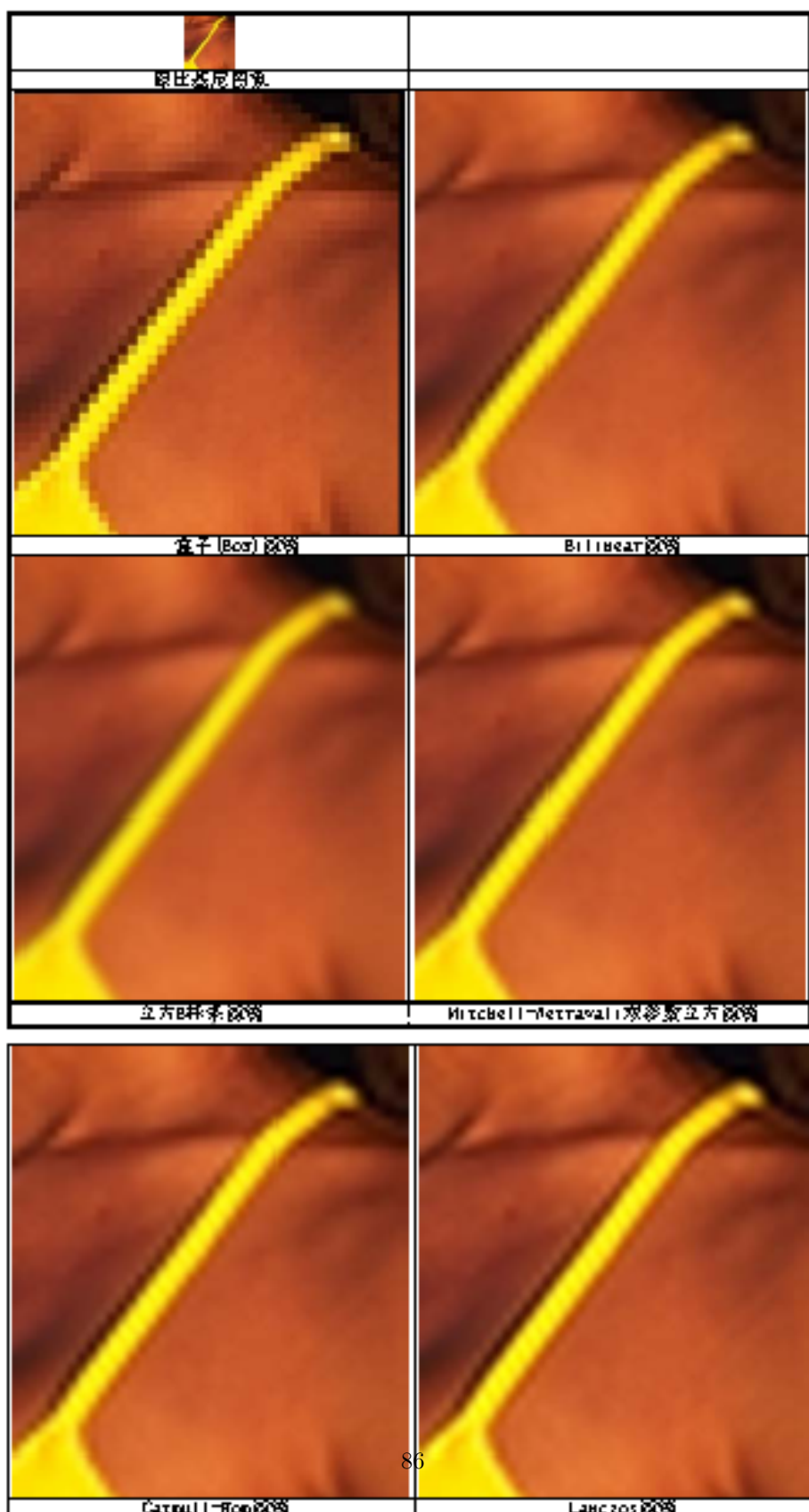


图 B.2: 不同重采样滤镜在比基尼相片从40x60增加到800%时的比较

附录 C 使用旋转函数

C.1 FreeImage_RotateClassic

下面的图演示了使用FreeImage_RotateClassic来将一幅图象旋转45° 的结果。请注意，旋转后的图象比原来的图象要大。



图 C.1:用FreeImage_RotateClassic将鸚鵡相片旋转45°

图 C.2 中展示了同一幅图旋转 90° 的情况。这次旋转后的图象和原来的图象有着同样大小。



图 C.2:用FreeImage_RotateClassic将鹦鹉相片旋转 90°

C.2 FreeImage_RotateEx

图 C.3 展示了可以用FreeImage_RotateEx函数得到的一些结果。

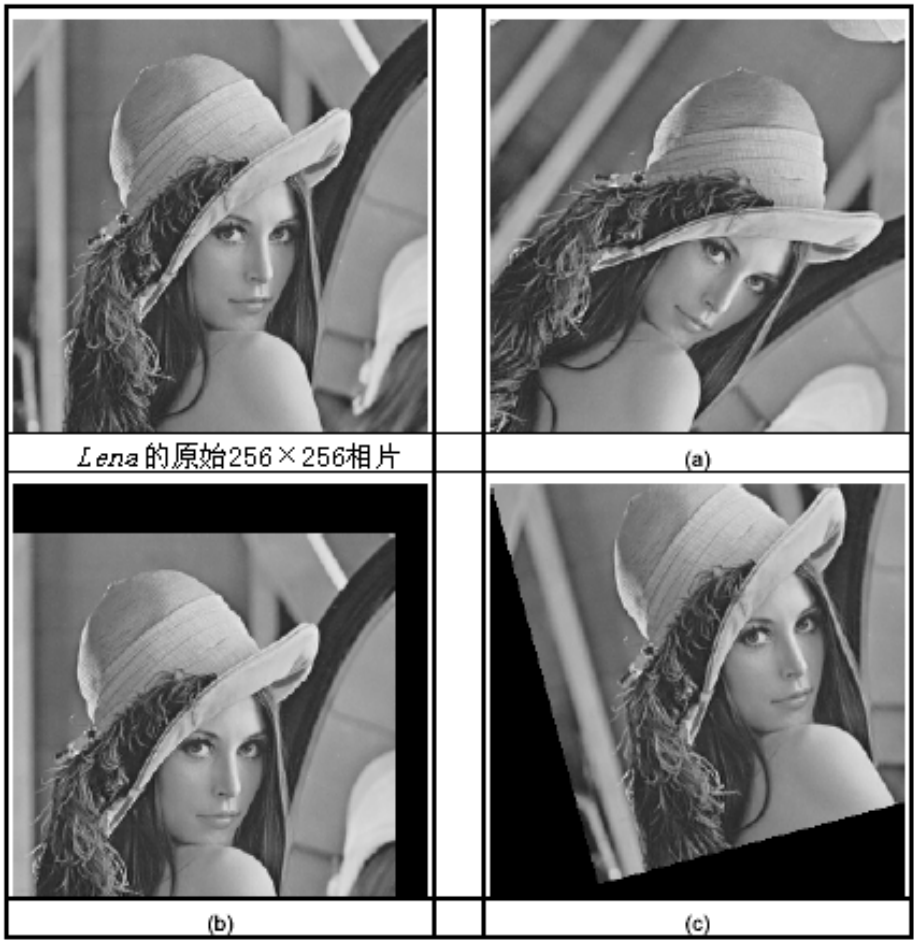


图 C.3:演示FreeImage_RotateEx用法的一些例子

(a) : 由一种任意变换(未用掩码)形成的图象。该图象已绕任意原点旋转一些角度,同时为了更好地测量,添加了额外的平移。注意对反射数据的影响(函数允许通过掩码把外推数据剔除出去,如果您想这么做的话)。

```
FIBITMAP *dst = FreeImage_RotateEx(src , angle , x_shift ,
    y_shift , x_origin , y_origin , FALSE);
```

(b) : 用下面代码做简单的整数平移形成的图象 :

```
FIBITMAP *dst = FreeImage_RotateEx(src, 0, -20, 30, 0, 0,  
    TRUE);
```

这次我们把`use_mask`参数设为TRUE, 以便通过掩码把图象中无关的部分剔除掉。

(c): 围绕左上角旋转移形成的图象:

```
FIBITMAP *dst = FreeImage\RotateEx(src, 15, 0, 0, 0, 0,  
    TRUE);
```

附录 D FreeImage元数据模型

D.1 FIMD_COMMENTS

这个模型用来储存图象的注释或图象的关键词。

JPEG格式支持单独的用户注释串，该串可以用标签域名”Comment”来设置。

通过将任意关键词用作标签域名，**PNG**格式支持任意多个用户注释串，每个关键词与元数据一起被保存和载入。

通过将任意关键词用作标签域名，**GIF**格式支持任意多个用户注释串，关键词不随元数据保存。载入时，每个注释与叫做”CommentX”的标签键相连，这里X是范围从0到N-1的一个数，而N是在GIF文件中的注释数。

D.2 FIMD_EXIF_*

这些模型用来载入储存于JPEG图象中的Exif元数据。以下是支持的子模型：

FIMD_EXIF_MAIN

这是Exif-TIFF元数据，即TIFF和Exif文件都共有的元数据。

FIMD_EXIF_EXIF

这个模型代表Exif规范元数据。

FIMD_EXIF_GPS

这个模型代表Exif GPS 元数据，这种元数据是Exif标准的一部分。

FIMD_EXIF_MAKERNOTE

Exif 制造商标志(maker notes)是由照相机制造商添加的元数据。对于这些元数据并没有公共的规范，每个制造商使用它自己的规范来命名标签域。

目前FreeImage库支持以下的制造商：Canon, Casio, Fujifilm, Kyocera, Minolta, Nikon (type 1, type 2 和 type 3), Olympus, Panasonic 以及 Pentax。

FIMD_EXIF_INTEROP

这个模型代表Exif公用(interoperability)元数据。

注：Exif规范可以在下面的网址上下载：<http://www.exif.org>

D.3 FIMD_IPTC

这个模型代表信息交换模型(IIM), 又叫做IPTC/NAA元数据模型, 最先由IPTC和美国报业协会(Newspaper Association of America, 即NAA)定义(见<http://www.iptc.org/IIM/>)。

在Adobe Photoshop中, 这个模型得到了广泛的使用, 但无论是IPTC抑或是Adobe现在已经不提倡对它的支持了, 因为它已经被XMP标准所取代。

D.4 FIMD_XMP

FIMD XMP 代表一个单独的Adobe XML包, 它是由标签域名"XMLPacket"来索引的。

Adobe XMP标准的描述是在下面的网址:

<http://www.adobe.com/products/xmp/main.html>

D.5 FIMD_GEOTIFF

这个模型代表GeoTIFF元数据标准, 用来向TIFF文件添加georeferencing(地理参考)信息。

GeoTIFF规范可以在下面的网址找到:

<http://www.remotesensing.org/geotiff/geotiff.html>

D.6 FIMD_ANIMATION

这个模型用来载入和保存与一个GIF或MNG动画文件相联的动画元数据。FIMD_ANIMATION模型支持的元数据已由FreeImage定义。目前该模型仅被GIF插件支持。

FIMD_ANIMATION规范在后面的附录中描述。

D.7 FIMD_CUSTOM

FIMD_CUSTOM是一个容器(placeholder)元数据模型, 可以用来储存用户自定义(user specific)的元数据。例如它可以用来储存可能在您编写的定制插件中使用的元数据。

附录 E FIMD_ANIMATION元数据模型规范

动画元数据模型是一个用来载入和保存与动画文件(诸如GIF或MNG文件)相联的动画元数据的通用模型。这个模型支持的元数据是特地为FreeImage库定义的(这不是一个诸如Exif之类的元数据标准)。

当向一个动画文件中保存动画元数据时, FreeImage元数据是被透明地传送到一个给定的插件规范所需要的元数据中的。另一方面, 当载入一个动画文件时, 文件的动画元数据是透明地传送到FreeImage动画元数据模型中的。

目前, 该模型仅被GIF插件支持。

以下是该模型所支持的元数据:

与单页相关联或与一个多页动画文件的第0页相关联的标签					
标签名	域名	标签号		类型	数目
		十进制	十六进制		
Logical width	LogicalWidth	1	0x0001	FIDT_SHORT	1
Logical height	LogicalHeight	2	0x0002	FIDT_SHORT	1
Global palette	GlobalPalette	3	0x0003	FIDT_PALETTE	任意
Loop	Loop	4	0x0004	FIDT_LONG	1
与一个单页或多页动画文件的每页(包括第0页)相关联的标签					
标签名	域名	标签号		类型	数目
		十进制	十六进制		
Frame left	FrameLeft	4097	0x1001	FIDT_SHORT	1
Frame top	FrameTop	4098	0x1002	FIDT_SHORT	1
No local palette	NoLocalPalette	4099	0x1003	FIDT_BYTE	1
Interlaced	Interlaced	4100	0x1004	FIDT_BYTE	1
Frame time	FrameTime	4101	0x1005	FIDT_LONG	1
Frame disposal method	DisposalMethod	4102	0x1006	FIDT_BYTE	1

注：下列值被DisposalMethod标签支持：

GIF_DISPOSAL_UNSPECIFIED = 0

GIF_DISPOSAL_LEAVE = 1

GIF_DISPOSAL_BACKGROUND = 2

GIF_DISPOSAL_PREVIOUS = 3

与单页或一个多页动画文件的第0页相关联的标签

LogicalWidth

0-65535范围内的、显示每页的整个画布宽度

Tag = 1 (0001.H)

Type = FIDT_SHORT

Count = 1

Save Default = dib宽度

Load: 总存在于文件中并被设置

LogicalHeight

0-65535范围内的、显示每页的整个画布高度

Tag = 2 (0002.H)

Type = FIDT_SHORT

Count = 1

Save Default = dib高度

Load: 总存在于文件中并被设置

GlobalPalette

”全局”调色板的RGBQUAD数据，该”全局”调色板可以应用于所有不具有本地调色板(最大可达256 x FIDT_PALETTE)的图象。

Tag = 3 (0003.H)

Type = FIDT_PALETTE

Count = 0 至 256

Save Default = no global palette

保存注解: 向下舍入到最近的2的幂

Load: 如果存在于文件中则设置，如果文件没有全局调色板则不设置

附注(只针对GIF)

调色板大小必须为2、4、8、16、32、64、128或256，或没有全局调色板(0)。

如果您指定一个具有count=127的元数据，则因为它向下舍入的缘故，只有最前面的64个会被使用，并且插件将会把GIF文件头中的全局调色板的大小设置为6位。

Loop

0-65536范围内(0=不定)的动画播放次数。

Tag: 4 (0004.H)

Type: FIDT_LONG

Count: 1

Save Default = 0 (播放次数不确定)

保存注解:

特定于GIF文件而言, NETSCAPE2.0应用扩展代表动画重复的次数, 于是, 1次重复意味着2个循环(从头到尾播放动画两遍), 65535是可以储存的最大重复次数值, 它被解释为65536个循环。

Load: 元数据总是被设置为0-65536范围内的一个值(若扩展存在于文件中则设置为0或2-65536, 若扩展不存在于文件中则设置为1)

与单页或多页动画文件的每页(包括第0页)相关联的标签

FrameLeft

显示图象的逻辑画布区的x方向位移(0-65535)。

Tag = 4097 (1001.H)

Type = FIDT_SHORT

Count = 1

Save Default = 0

Load: 总存在于文件中并被设置

FrameTop

显示图象的逻辑画布区的y方向位移(0-65535)。

Tag = 4098 (1002.H)

Type = FIDT_SHORT

Count = 1

Save Default = 0

Load: 总存在于文件中并被设置

NoLocalPalette

一个标志—压缩保存与dib相连的调色板的标志(强制dib使用全局调色板)。本地调色板是由一个页所使用的调色板, 该调色板数据不是通过元数据设置的(象全局调色板), 因为它是与dib相连的。

Tag = 4099 (1003.H)

Type = FIDT_BYTE

Count = 1

Save Default = 0 (意味着, 是的, 保存本地调色板数据)

Load: 总存在于文件中并被设置

Interlaced

告知图象是否应该被交替地保存(stored interlaced)。

Tag = 4100 (1004.H)

Type = FIDT_BYTE

Count = 1

Save Default = 0

Load: 总存在于文件中并被设置

FrameTime

以毫秒计的每帧播放时间(只针对GIF文件—文件中保存的值为百分之一秒)。

Tag = 4101 (1005.H)

Type = FIDT_LONG

Count = 1

Save Default = 100ms (只针对GIF文件—保存的值为10cs)

保存注解:

对于GIF文件,以毫秒指定的值被向下舍入,例如129ms(129毫秒)被保存为12cs。IE5/IE6具有一个最小默认值100ms, Mozilla/Firefox/Netscape 6+/Opera具有最小值20ms和默认值100ms—如果指定的值小于20ms或缺少GCE的话。Netscape 4 具有最小值10ms—如果指定0ms的话取最小值,但如果缺少GCE的话将使用0ms这个值。GIF插件总是向GIF文件中写入一个GCE扩展,并且它总是使用GIF89a。

Load: 总是被设置,如果不存在于文件中则设置为0。

DisposalMethod

显示图象后在逻辑画布上要做的事情。

Tag = 4102 (1006.H)

Type = FIDT_BYTE

Count = 1

Save Default = GIF_DISPOSAL_BACKGROUND (恢复到背景色,背景色是透明的,具有0 alpha值)

保存注解:

GIF_DISPOSAL_UNSPECIFIED 可能与GIF_DISPOSAL_LEAVE所做的事情相同,不应该使用这个值。

GIF_DISPOSAL_LEAVE将会把图象留在原处,整个或部分地被下一个图象重画。

GIF_DISPOSAL_BACKGROUND将会以背景色填充图象帧所使用的区域。

GIF_DISPOSAL_PREVIOUS将会在重画图象之前把逻辑画布返回到上次状态。

Load: 总是被设置,如果不存在于文件中则设置为GIF_DISPOSAL_LEAVE。

附注(只针对GIF)

透明度对所有页都提供单独支持,使用透明度表中第一个完全透明的索引,而透明度表中其他的索引则将会是完全不透明。

背景色只为第0页储存和使用，但需要设置全局调色板以便恰当地使用它。

GIF_PLAYBACK载入选项(参见表 2.3)将会通过播放从第0页到指定页的形式，把单独的页作为一个具有透明度的32bpp图象来载入，载入时遵循透明覆盖方法和gif布置(disposal)方法。请注意它并不是以一个可播放的方式来实际播放图象。它在内部从第0页到所请求的页”播放”图象，返回一个单独的静止图象，这是图象帧的真正外观。

GIF_LOAD256载入选项由GIF_PLAYBACK内部使用，但用户同样可以使用。它只是防止许多移位操作以及2色和16色图象带来的烦人的事情。

使用FIMD_ANIMATION元数据模型

该模型对用FreeImage来生成GIF动画文件是很有用的，网络浏览器稍后会播放这些动画文件。这些元数据用来保存(和载入)GIF文件为定义一个动画而支持的各种不同选项。

最简单的例子不需要改变任何元数据。仅仅以create_new=TRUE的方式打开一个多页GIF文件，并开始向其中添加动画页即可。

在您关闭多页图象时，生成的GIF动画文件将会永久不停地播放，以1/10秒钟(100毫秒)的间隔显示每页。

GIF文件的每一页有它自己的调色板并填充整个逻辑区域。

用户最容易陷入的障碍是向多页位图中添加比他们添加的第一页还大的页，因为没有设置特定的元数据，逻辑画布区域将会被设置为与第一页大小相同。而如果您有一帧超出了画布区的话，则这种情况是GIF规范在技术上未定义的(不允许的)。(IE/Firefox将会简单地使图象变大，变成与最大帧所需要的大小相同的尺寸；Opera将会砍掉图象超出逻辑区之外的任何部分)。

```
// 假设我们有一个已经是 8bpp 的 dib 数组且图象大小全都相同
// 还有一些叫做 fps 的浮点数代表每秒钟播放的帧数
FIMULTIBITMAP *multi = FreeImage_OpenMultiBitmap(FIF_GIF,
    "output.gif", TRUE, FALSE);
DWORD dwFrameTime = (DWORD)((1000.0f / fps) + 0.5f);
for(int i = 0; i < 10; i++) {
    // 清除 dib 使用的任何动画元数据, 因为我们将添加自己的元数据
    FreeImage_SetMetadata(FIMD_ANIMATION, dib[i], NULL,
        NULL);
    // 向 dib[i] 添加动画标签
    FITAG *tag = FreeImage_CreateTag();
    if(tag) {
        FreeImage_SetTagKey(tag, "FrameTime");
        FreeImage_SetTagType(tag, FIDT_LONG);
        FreeImage_SetTagCount(tag, 1);
        FreeImage_SetTagLength(tag, 4);
        FreeImage_SetTagValue(tag, &dwFrameTime);
        FreeImage_SetMetadata(FIMD_ANIMATION, dib[i],
            FreeImage_GetTagKey(tag), tag);
        FreeImage_DeleteTag(tag);
    }
    FreeImage_AppendPage(multi, dib[i]);
}
FreeImage_CloseMultiBitmap(multi);
```