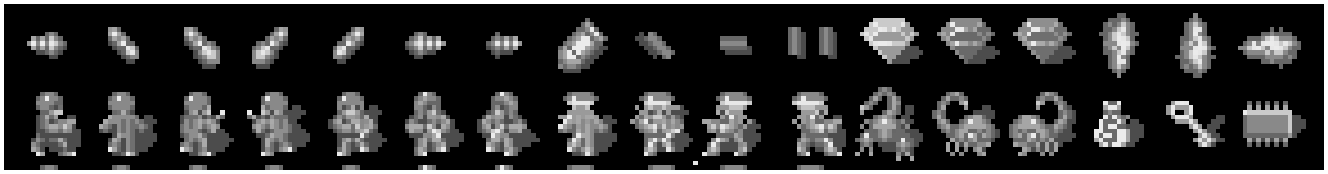


Chapter 3

# Image Compression and Graphic File Formats

**In this chapter, you'll learn about:**

- **Common image compression schemes**
- **Essential graphic file formats**
- **Important graphic file formats**
- **File format suitability for arcade game graphics**
- **Caveats for working with graphic files**
- **Specific graphic file format recommendations**

## Image Compression

Graphic images can consume a lot of disk space, particularly when they're saved at high screen resolutions and/or color depths. To see what I mean, let's calculate the file size required by images saved at common screen resolutions and color depths by applying this simple formula:

```
                    H x V x C
File Size (KB) = --------------
                    8 x 1024

H represents the number of horizontal pixels
V represents the number of vertical pixels
C represents the color depth
```

TABLE 3-1: Examples of Different Graphic Image File Sizes

| *Screen Resolution* | *Color Depth* | *File Size (in KB)* |
|---|---|---|
| 320x200 | 4 bits/pixel | 32 |
| 320x200 | 8 bits/pixel | 64 |
| 320x240 | 8 bits/pixel | 75 |
| 640x480 | 4 bits/pixel | 150 |
| 640x480 | 8 bits/pixel | 300 |
| 640x480 | 16 bits/pixel | 600 |
| 640x480 | 24 bits/pixel | 900 |
| 800x600 | 8 bits/pixel | 468.75 |
| 800x600 | 16 bits/pixel | 937.5 |
| 800x600 | 24 bits/pixel | 1406.25 |

As you can see from Table 3-1, graphics files, particularly those at higher screen resolutions and color depths, can grow to be quite large. When designing arcade game graphics, you can easily wind up with dozens, if not hundreds, of such files. Even with today's multi-gigabyte hard drives, storing them can consume a fair amount of disk space. What's more, large graphics files tend to be more difficult to work with than smaller ones as they take longer to load, save, and manipulate than their smaller counterparts. Need to put a bunch on a floppy? Forget about it. Want to e-mail one to a friend in New Zealand? Don't even bother unless you both have access to ISDN, DSL, or cable modems.

To make more efficient use of available disk space, most graphics files are *compressed*. Compression makes use of complex mathematical equations to dramatically reduce the sizes of images. There are several methods of image compression available, however, they all tend to fall into two broad categories: *lossless compression* and *lossy compression*.

## Lossless Compression

Lossless compression technologies involve no data loss. The original information of each file is stored intact and can be completely recovered from its compressed state at the time it is decompressed or extracted. Lossless compression is usually implemented using one of two different methods: *statistical* or *dictionary-based*.

The statistical method uses a three-step process to do its dirty work. First it reads in a single byte of data. Next, it calculates the probability of that byte's appearance in the data. Finally, it encodes the byte by replacing it with a symbolic token that acts as a kind of shorthand version of the data. When the data is later decoded, the symbol is replaced with the original byte. The statistical method can achieve very high compression levels if the data contains many repeating values.

The dictionary-based method uses a somewhat different concept. It reads in several bytes of data and looks for groups of symbols that appear in a dictionary. If a match is found, a pointer into the dictionary is made. The more matches found in the image data, the higher the level of compression that can be achieved.

Lossless compression is important to us because it maintains the *integrity*, or the quality, of the file's information.

There are several lossless compression algorithms commonly used on graphic images. These include:

- RLE compression
- Packbits compression
- LZ77 compression
- LZW compression

### RLE Compression

Run-length encoding, or RLE, compression works on images by analyzing their graphic data sequentially from left to right and top to bottom. It compares the value of each byte of information with the value of the previous byte. Each new data value is recorded into a "packet" of two bytes where the first byte contains the number of times the value is repeated, and the second packet contains the actual value. The bytes in the packet are known as the "run count" and "run value," respectively.

In situations where a graphic image contains many repeat values, the compression ratio will be very high. For example, if every byte in a 1000-byte image was identical, its size could be reduced to 20 bytes, giving it a 50:1 compression ratio. An image with lots of different values won't compress very well, and in some cases can actually become larger than the original. For example, if all bytes in an image

are different from each other, the image's size will double, because two bytes are used to store each byte in the image.

RLE compression exists in several forms, all of which work essentially the same way but differ somewhat in terms of efficiency. Because it works well for most types of images, RLE is the most common lossless image compression technology and is used by such popular graphics formats as PCX and BMP.

## Packbits Compression

Packbits is considered an RLE compression scheme because it also looks for "runs," or repeated values, and calculates their number, or overall "length." The repeated bytes it finds are then "packed" together, hence the name. Compared to RLE, images that use packbits typically offer superior compression rates. Yet despite this, packbits compression is used by only a few file formats, namely by LBM and TGA.

## LZ77 Compression

LZ77 (Lempel, Ziv, 77) compression was developed in 1977. It works by keeping track of the last *n* bytes of the data seen, and when it encounters a data sequence that has already been analyzed, it records its position along with the length of the sequence.

LZ77 is used as the primary compression scheme for the PNG image format. In addition, all popular file compression schemes, such as ARJ, LHA, ZIP, and ZOO, are based on the LZ77 algorithm.

## LZW Compression

The LZW (Lempel, Ziv, Welsh) compression scheme does its work by looking for patterns of data and assigning codes to them. It works best on images with large areas of solid colors. Images that contain many different colors and patterns aren't very good candidates for this type of compression.

LZW is a "dictionary-based" compression technology. Its "dictionary" includes a special table that contains a code for each possible value contained in an image. For example, if the LZW compression is used on an 8-bit image, the LZW dictionary will be initialized with codes for up to 256 possible values. As the image data is read, new values are added to the table as each unique pattern of data is found. LZW dictionaries aren't actually saved along with the compressed data. Instead, the dictionary is reconstructed as part of the image decompression process.

LZW is most often associated with GIF (Graphics Interchange Format) image files. However, it is used for other types of graphics files, including TIFF (Tagged

Image File Format), and the basic technology serves as the foundation for many of the ZIP file compression algorithms.

## Lossy Compression

Lossy compression techniques all involve some sort of information loss. Data that is compressed using a lossy compression scheme usually can't be restored back to its original, uncompressed state. However, in exchange for this inaccurate reconstruction, lossy compression is capable of achieving very high compression rates.

**NOTE:** When using a lossy compression scheme, repeatedly saving an image will cause that image's quality to degrade.

There are two lossy compression methods in general use. These are:

- Color reduction
- JPEG

### Color Reduction

Color reduction is a special process that allows you to create files with fewer colors than contained in the original image. For example, you can reduce a 24-bit (with millions of colors) image down to an 8-bit (256 colors) one. The end result will produce a new file that is substantially smaller than the original, as indicated by Table 3-1. Whenever you reduce the number of colors in an image, there is always some image information that is lost.

There are a couple of color reduction techniques in use. The most basic form is called *remapping*. This simply means that when an image is color reduced, the resulting image will have the best color reproduction possible. Every pixel contained in it will be replaced with colors available in the new palette. Unfortunately, the results from remapping are usually less than satisfactory.

The other color reduction process is called *color dithering*. Color dithering works much like mixing two shades of paint to produce a third color. Instead of mixing paint, however, it blends patterns of different colors to simulate the presence of colors not in the current palette. As a result, color dithering can usually produce better results than remapping, especially when reducing an image from 256 colors down to 16. Most graphics programs support both methods of color reduction, although the resulting image quality varies widely between them.

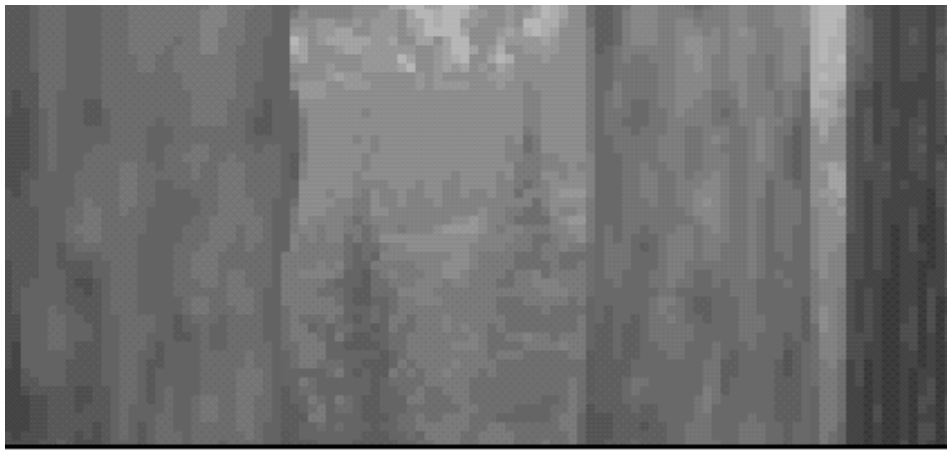
The primary advantage of color reduction is that it's relatively fast and usually produces good results on most types of images. Color reduction is almost always used in conjunction with one of the lossless compression techniques to reduce

image file size even further. Both color reduction and dithering are discussed in more detail in Chapter 8.

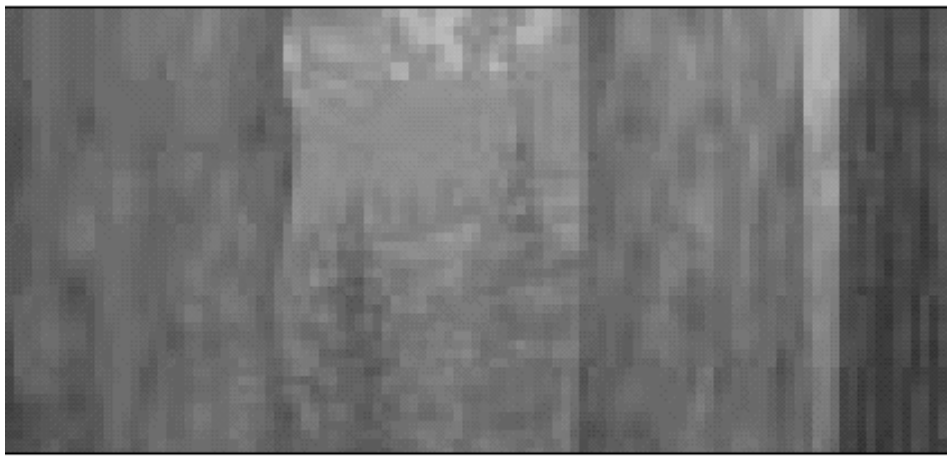
## JPEG

JPEG is a lossy compression technique that is used for compressing photo-realistic images with millions of colors in them. For more information on JPEG and how it works, see the JPEG entry under the Important Graphic File Formats section of this chapter.

Figure 3-1 compares the two compression schemes. Notice the artifacting exhibited by lossy compression compared to lossless compression.



Lossless Compression



Lossy Compression

FIGURE 3-1: Lossless vs. Lossy Compression Comparison

# Essential Graphic File Formats

Computers use file formats to negotiate the exchange of information between different applications and systems. Graphics are no different in this regard. Once you spend time designing game artwork, you'll want to save what you create for future editing or display. Many graphics editors offer several choices of file formats to save your pictures in. However, only a few are actually useful to arcade game graphics designers. I've taken the liberty of identifying and describing them here.

## BMP (Bitmap)

**Native Platform(s):** Windows 3.1, 95, 98, NT 4.0, and 2000

**File Extension(s):** .BMP, .bmp

BMP is the standard graphics format used by the various incarnations of Microsoft's Windows operating systems (i.e., Windows 3.1, 95, 98, and NT). The BMP format comes in two flavors, RGB and RLE.

The BMP RGB format, as its name implies, saves images as RGB data (red, green, blue) which is a representation of what's actually generated on the computer's display. This allows you to accurately save and exchange images with thousands or millions of colors in them. The RGB format doesn't support image compression.

The BMP RLE format stores image data using one of the RLE compression schemes. Unlike the BMP RGB format, this allows you to take advantage of compression and substantially reduce the overall size of your graphic images. Besides compression, the main difference between the BMP RGB and BMP RLE formats is that the BMP RLE format is limited to saving images with a maximum of 256 colors.

The Windows BMP format is definitely one of the best choices when it comes to saving your game graphics, especially on the Windows platform. This is simply due to the fact that virtually every Windows graphics package can read and write BMP files. In addition, the BMP format enjoys a good level of cross-platform compatibility, as Apple *QuickTime* (versions 2.0 and better) and most major Macintosh graphics packages have the ability to read and write both BMP format variations.

**NOTE:** Although BMP is the standard Windows graphics file format, some Windows graphics programs still don't support RLE flavored BMP files. To avoid any unpleasant compatibility problems down the road, make sure all of the graphics software you use works with BMP RLE files. Otherwise, just to be safe, always save your files in the BMP RGB format.

TABLE 3-2: BMP Characteristics

| *BMP Variant* | *Screen Resolutions* | *Compression* | *Lossy* | *Color Depths Supported* | *Platform Compatibility* |
|---|---|---|---|---|---|
| RGB | Any | No | No | 1, 4, 8, or 24 bits | Windows, Macintosh* |
| RLE | Any | Yes, RLE | No | 1, 4, 8, or 24 bits | Windows, Macintosh* |

* Compatible with *QuickTime* and most popular graphics packages.

**NOTE:** Windows also supports two related image formats called *DIB*, or device-independent bitmap, and *RLE*, or run-length encoding (simply an RLE compressed DIB file with an .RLE file extension). While a number of graphics packages can read and write these formats, there are still plenty of graphics programs that won't. Therefore, I generally don't recommend using these formats unless you know your graphics software can actually support them.

## GIF (Graphics Interchange Format)

**Native Platform(s):** N/A

**File Extension(s):** .GIF, .gif

The GIF format was developed by the CompuServe Information Service during the mid-1980s as a means to pass graphic files between different computers. Until GIF was developed, graphic formats were usually limited to being used on the same types of machines. Since that time, the format has evolved and has established itself as a universal graphics standard.

There are two versions of the GIF format in use: *87a* and *89a*. GIF 87a was the original implementation of the format. Although still used, it has been largely replaced by the later 89a format. Technically speaking, there is little difference between the two, although the 89a format does offer some additional features. These include interlacing, the ability to store multiple images within a single image, and transparency. *Interlacing* enables images to appear in stages while they load. Multiple frames can be used to produce simple, frame-based animation effects. These images are frequently referred to as *animated GIFs*. *Transparency* allows the background of an image to show through, making it seem as if an image is free-floating. Neither version of the format can be used to store images with more than 256 colors.

Animated GIF files are particularly interesting to the arcade game artist. They are essentially compiled sequences of images and are frequently used in online games as game sprites. In fact, I used them to create all of my game animations for my work at ZapSpot. However, be careful when using them in your own game projects

because once you create an animated GIF image, it's very difficult to extract the individual frames of such animations for additional editing and manipulation. Therefore, animated GIFs can best be thought of as a non-reusable image format.

**NOTE:** While both versions of the format are widely used, some older graphics programs don't know how to deal with GIF 89a files. Therefore, if you opt to store your images in the GIF format, always try to save your images as GIF 87a to ensure maximum compatibility with different applications and platforms.

By definition, files stored in the GIF format are always compressed. Yet, unlike most of the other formats described here, they utilize the LZW compression scheme. This puts them at a slight advantage over other formats as images compressed using LZW generally offer better compression ratios, especially when used on images that contain large areas of similar colors.

The GIF format has seen widespread use on the Internet and World Wide Web where it has become the graphics format of choice. Yet, surprisingly, it hasn't seen much use in game development, despite being perfectly suited for the role. Due to its near universal popularity, virtually all graphics packages support the GIF format. In fact, it's probably one of the safest and reliable methods for exchanging graphics between different computers and operating systems.

One potential problem with using GIF files to store your game graphics is how some graphics programs will optimize the color palette. While this has no adverse effect on how the image actually appears on-screen, it can cause problems when you try to edit the file later as the image colors won't be arranged in exactly the same order as when you first created the file. What's more, due to the optimization, many of the colors of the image's original color palette will actually be removed. This situation is particularly true of graphics packages that emphasize GIF use for the World Wide Web. Make sure your software doesn't do this by testing it with non-critical files before you start using the GIF format to store your important artwork. To be safe, consider saving your images as GIF only once they are finalized or simply use a different file format to store your images altogether.

**NOTE:** Since 1994, there have been major legal issues involved with the LZW compression scheme used by the GIF format. The Unisys Corporation owns the patents to the technology and now requires any vendor which develops software that uses the GIF format to acquire an expensive license. This situation may serve to reduce the popularity of GIF in future generations of graphics software.

**NOTE:** As the GIF format is one of the most universal of all graphic formats, it's also a natural choice for storing graphics for cross-platform, Java-based arcade games.

TABLE 3-3: GIF Characteristics

| *GIF Variant* | *Screen Resolutions* | *Compression* | *Lossy* | *Color Depths Supported* | *Platform Compatibility* |
|---|---|---|---|---|---|
| GIF 87a | Any | Yes | No | 1, 2, 3, 4, 5, 6, 7, or 8 bits | DOS, Macintosh, Windows, Linux, Java |
| GIF 89a | Any | Yes | No | 1, 2, 3, 4, 5, 6, 7, or 8 bits | DOS, Macintosh, Windows, Linux, Java |

## IFF (Interchange File Format)

**Native Platform(s):** Commodore Amiga, Atari ST, DOS

**File Extension(s):** .IFF, .iff, .LBM, .lbm

The IFF format originated on the Commodore Amiga around 1985. It was developed by Electronic Arts for use with their powerful *Deluxe Paint* graphics package. The format's versatility quickly made it a standard on that platform. Later, the IFF format made its debut on both DOS (c. 1988) and the Atari ST (c. 1990) when versions of the *Deluxe Paint* program were released for these systems.

There are two versions of the IFF format still in everyday use: IFF and LBM. The basic IFF format is compatible with several different systems including the Amiga, Atari ST, and the PC. The LBM format, on the other hand, is native to PC and its version of the *Deluxe Paint* program. LBM is actually a subset of the IFF format called PBM. However, any graphics program compatible with IFF files can usually handle LBM files and vice versa.

Both IFF formats support compressed and uncompressed images. When dealing with compressed images, IFF usually employs RLE compression, while LBM files usually use packbits compression.

It's been my experience that some programs have problems reading LBM files created with the PC version of *Deluxe Paint* and vice versa. This is probably due to the fact that Electronic Arts never officially documented how the LBM packbits compression was actually implemented in their software. Therefore, there's a wide degree of variation in how different programs interpret and write these files. Because of this, my advice is to not use LBM for your images. Rather, use the PCX format instead. Given this issue, it's surprising to find that *Deluxe Paint* has no problems with the IFF files generated by other programs.

TABLE 3-4: IFF Characteristics

| *IFF Variant* | *Screen Resolutions* | *Compression* | *Lossy* | *Color Depths Supported* | *Platform Compatibility* |
|---|---|---|---|---|---|
| IFF | Any | Yes, RLE | No | 1, 4, 5, 6, 8, or 24 bits | Atari ST, Amiga, DOS, Windows*, Macintosh* |
| LBM | Any | Yes, packbits | No | 1, 4, or 8 bits | DOS |

* Denotes limited compatibility, usually obtained through graphic conversion software.

## PCX

**Native Platform(s):** DOS

**File Extension(s):** .PCX, .pcx

The PCX format has been around for a long time. ZSoft developed it in the mid-1980s for its popular *PC Paintbrush*. Because of its longevity, PCX is almost as ubiquitous as the GIF format and is even more widely supported. Even so, PCX is most at home on DOS systems where over the years it has established itself as the de facto graphics standard for that platform.

The PCX format is quite flexible and can store images with color depths of 1, 2, 4, 8, and even 24 bits. PCX files also use an RLE compression scheme, which makes PCX the recommended file format for any DOS related game and graphics development. However, because BMP is the Windows standard graphics format, I suggest that you use BMP instead of PCX when working with Windows applications if you have a choice. At this time, PCX enjoys only limited support on the Macintosh platform. Although most of the major Macintosh graphics packages can read and write the format, *QuickTime* doesn't currently support it, thus limiting its overall compatibility on that platform.

**NOTE:** Several versions of the PCX graphics format (i.e., version 1, 3, and 5) actually exist. However, versions below 5 are seriously outdated and do not support color depths greater than 4 bits. Therefore, I have opted not to discuss them here.

Even with the PCX format's widespread popularity, it's far from perfect. While the format is well documented, it isn't always well implemented. I've observed various discrepancies with how different programs interpret and display PCX files. Despite this issue, PCX is a good alternative to BMP, especially when working with programs or development environments that don't support the BMP format.

PCX can actually be considered an unofficial Windows graphics format because it's supported by the *MS Paint* application bundled with every version of Windows since version 3.1. Incidentally, *MS Paint* is actually a stripped-down version of the *PC Paintbrush* program.

TABLE 3-5: PCX Characteristics

| *PCX Variant* | *Screen Resolutions* | *Compression* | *Lossy* | *Color Depths Supported* | *Platform Compatibility* |
|---|---|---|---|---|---|
| Version 5 | Any | Yes, RLE | No | 1, 2, 4, 8, or 24 bits | DOS, Windows, Macintosh* |

* Denotes limited compatibility, usually obtained through graphic conversion software.

## PICT (Picture)

**Native Platform(s):** Apple Macintosh

**File Extension(s):** .PICT, .pict, .PCT, .pct

Apple developed the PICT file format way back in 1984, when it introduced the original Macintosh computer. PICT is the standard graphics format used by Macintosh computers. Because the format is internal to the Macintosh's operating system, it can be assumed that all Macintosh graphics programs can read and/or write legally generated PICT files without any problems.

There have been two major versions of the PICT format introduced, PICT 1 and PICT 2. PICT 1 was the original version of the format. It supported images that contained up to eight colors and has long been considered obsolete. PICT 2 was introduced several years after PICT 1. It's the current version of the format and is capable of supporting images that contain as many as 16,777,216 colors. A few minor variations of the format exist, however, these are largely proprietary to certain applications and are rarely used otherwise.

The PICT format is interesting in that it's capable of storing both vector and bitmap data within the same file. However, the vast majority of the images saved in the PICT format are actually just straight bitmaps. As with most of the formats described here, PICT utilizes a form of RLE compression in order to keep images reasonably small and compact when stored on disk.

**NOTE:** If a Macintosh user has *QuickTime* 2.0 or better installed on his or her machine, it's possible to save PICT files using whatever compression library *QuickTime* supports, including JPEG.

**NOTE:** PICT files are viewable through the Windows desktop if there's a version of *QuickTime* 2.5 or better installed.

TABLE 3-6: PICT Characteristics

| *PICT Variant* | *Screen Resolutions* | *Compression* | *Lossy* | *Color Depths Supported* | *Platform Compatibility* |
|---|---|---|---|---|---|
| PICT 2 | Any | Yes, RLE or *QuickTime* | No | 1, 4, 8, 24, or 32 bits+ | Macintosh, Windows* |

* Denotes limited compatibility, usually obtained through graphic conversion software. The Windows version of *QuickTime* allows them to be viewable, however.

\+ A color depth of 32 bits is actually 24 bits with an 8-bit alpha channel used to store transparency information.

# Important Graphic File Formats

The graphic file formats mentioned in this part of the chapter impose some sort of limitation on your artwork, either in terms of overall system/application compatibility or image quality. This does not mean they are useless, however. They all have some use or potential use in game graphics development and it's more than likely that you'll come across one or more of them in your projects.

## FLIC

**Native Platform(s):** DOS

**File Extension(s):** .FLI, .fli, .FLC, .flc

Programmer Jim Kent (author of *Autodesk Animator* for DOS and *Aegis Animator* for the Atari ST) developed the original FLIC file format in 1993 as an attempt to create a standard animation playback format. Since that time, the format has seen widespread use by DOS game developers.

There are two variations of the FLIC format in use: FLI and FLIC. FLI files are the older of the two and are limited to animations with a 320x200 resolution. The later and more common FLIC format supports several additional features and extensions. Animations in both versions are limited to a maximum of 4,000 frames.

All FLIC files are capable of displaying up to 256 colors per frame of animation and can utilize a variety of compression algorithms including RLE and packbits.

The FLIC format was a widely used predecessor of the Windows AVI movie format and is still commonly used in game development circles. However, as an animation format, FLIC files are best suited for large animations such as those

featured in game title sequences or cut scenes. For smaller animations, other formats such as animated GIFs are a better choice.

**NOTE:** FLI files limit their color values to VGA 0-63 levels or a maximum of 64 shades per color. The FLIC format was later modified to accommodate SVGA 0-255 color values or a maximum of 256 shades per color. Please refer to Chapter 9 for more information on the specific implications of these differences.

TABLE 3-7: FLI/FLIC Characteristics

| *FLIC Variant* | *Screen Resolutions* | *Compression* | *Lossy* | *Color Depths Supported* | *Platform Compatibility* |
|---|---|---|---|---|---|
| FLI | Limited to a maximum of 320x200 | Yes, RLE | No | 8 bit (color) | DOS, Windows* |
| FLIC | 320x200, 640x480, 800x600 | Yes, RLE, packbits | No | 8 bit (color) or 16+ or 24 bit+ | DOS, Windows* |

* Denotes limited compatibility, usually obtained through graphic conversion software.
\+ Not part of the official file format but supported by extended variations of the format.

## JPEG (Joint Photographic Experts Group)

**Native Platform(s):** N/A
**File Extension(s):** .JPG, .jpg

JPEG is an image compression mechanism and not a file format in itself. There are actually several classes of JPEG compression but the most popular is known as JFIF. However, in everyday use, the terms JPEG and JFIF are synonymous.

JPEG uses a lossy compression scheme that is designed to take advantage of the limitations of the human eye. Any image detail that is not visually perceived by the eye is lost when stored as a JPEG. As a result, JPEG is capable of providing images with incredible compression ratios, usually anywhere between 5:1 and 15:1. JPEG is also unique among the compression techniques mentioned here because it allows you to trade off image quality to gain more compression. The higher the level of compression, the lower the quality of the image and vice versa. The most common side effect of using JPEG compression is that images often display noticeable artifacting and blurring. JPEG works best on highly detailed images, such as photographs, as these types of images have more content to lose. Also, any image that uses JPEG compression is automatically promoted to 24-bit color depth, even if it was originally created at a lower color depth.

In addition to JFIF, there are two other variations of JPEG compression, progressive JFIF and lossless JFIF. Progressive JFIF (also called progressive JPEG) stores an image in overlapping layers. When a progressive JFIF image loads, it displays its content in stages. Each layer appears gradually until the whole image is rendered. This makes progressive JFIF useful for displaying large images in situations where bandwidth might be low, i.e., over the Internet or on slow computers. Lossless JFIF stores images without using lossy compression. Therefore, images stored this way do not suffer from the visual problems traditionally associated with JPEG compression. Yet, at the same time, lossless JFIF images also don't enjoy the high image compression rates offered by standard JFIF.

Using JPEG compression for arcade game graphics is generally not a good idea. First, it's lousy for artwork that contains sharp details (which are commonly found in sprites and menu screens) since the colors of different objects tend to bleed into each other. Second, it promotes all images to 24 bit regardless of their original color depth, making them useless for palette-based images. And, finally, images stored as JPEGs can't be resaved without further degrading their image quality. This makes it nearly impossible to edit them without destroying the integrity of the original image.

TABLE 3-8: JPEG Characteristics

| *JPEG Variant* | *Screen Resolutions* | *Compression* | *Lossy* | *Color Depths Supported* | *Platform Compatibility* |
|---|---|---|---|---|---|
| JFIF | Any | Yes, JPEG | Yes | 8 bit (grayscale) or 24 bits | DOS, Macintosh, Windows, Linux, Java |
| Progressive JFIF | Any | Yes, JPEG | Yes | 8 bit (grayscale) or 24 bits | DOS, Macintosh, Windows, Linux, Java |
| Lossless JFIF | Any | No | No | 8 bit (grayscale) or 24 bits | DOS, Macintosh, Windows, Linux, Java |

## PNG

**Native Platform(s):** N/A

**File Extension(s):** .PNG, .png

The PNG format evolved out of the ashes of the 1994 Unisys patent dispute over the LZW compression used in the GIF format when Thomas Boutell (along with others) developed the PNG format specification as an alternative and eventual replacement for the GIF format. Although until recently support for PNG has been slow in coming, it's now on the verge of becoming a new standard as more and more graphics tools are being updated to support it.

Like GIF files, PNG files support interlacing and transparency information (up to 254 levels). However, the PNG format also introduces several improvements over GIF and the other file formats mentioned here. These include built-in gamma correction, support for 8-bit color, 8-bit grayscale, true color images, and built-in file integrity checking. PNG's internal gamma correction ensures that graphic images saved as PNG files will retain a consistent gamma level regardless of the platform on which they're displayed. Its ability to store both 8-bit and 24-bit images makes it extremely useful for game-oriented graphics. Finally, its built-in file integrity checking will help minimize problems with file corruption and the like.

The PNG format uses the LZ77 compression algorithm that gives it better compression ratios than many of the other file formats discussed here.

TABLE 3-9: PNG Characteristics

| *PNG Variant* | *Screen Resolutions* | *Compression* | *Lossy* | *Color Depths Supported* | *Platform Compatibility* |
|---|---|---|---|---|---|
| PNG | Any | Yes, ZIP | No | 8-bit color, 8-bit grayscale, or 24 bits | DOS, Macintosh, Windows, Linux, Java |

## PSD (Photoshop)

**Native Platform(s):** Macintosh, Windows 3.1, 95, 98, NT 4.0, and 2000

**File Extension(s):** .PSD, .psd

The PSD format is the native file format used by Adobe's *Photoshop* program. It can store images with up to 24-bit color, and the current version supports features such as layers and transparency. PSD files can be either uncompressed or compressed using RLE compression.

Although *Photoshop* enjoys an extremely large following among Web, multimedia, and game developers worldwide, it's also an expensive program. This fact means that few low-end graphics packages are currently compatible with the PSD format. Therefore, unless you have specific need to use it, it is recommended that you avoid saving your images in PSD format.

TABLE 3-10: PSD Characteristics

| *PSD Variant* | *Screen Resolutions* | *Compression* | *Lossy* | *Color Depths Supported* | *Platform Compatibility* |
|---|---|---|---|---|---|
| PSD 5.0 | Any | Yes, RLE | No | 8 or 24 bits | Macintosh, Windows |
| PSD 3.0 | Any | Yes, RLE | No | 8 or 24 bits | Macintosh, Windows, Linux+ |
| PSD 2.0/2.5 | Any | Yes, RLE | No | 8 or 24 bits | Macintosh*, Windows* |

* Denotes limited compatibility, usually obtained through graphic conversion software.
\+ Linux systems can read PSD files when using the free *GIMP* image editor software that is frequently bundled with many Linux installations.

**NOTE:** Most of the applications outside of *Photoshop* that support the PSD file format are only compatible with the older, PSD 2.0/2.5 variations. PSD versions below 2.5 don't support layers or transparency information. Also, PSD 5.0 files store text as editable layers. This information might be lost when loading such files into older versions of *Photoshop* or other applications. Keep these issues in mind when exchanging files with applications that claim PSD compatibility.

## PSP (Paint Shop Pro)

**Native Platform(s):** Windows 3.1, 95, 98, NT 4.0, and 2000

**File Extension(s):** .PSP, .psp

PSP is the native file format of Jasc's popular *Paint Shop Pro* program. As with most of the file formats described here, the PSP format can store graphics images at a variety of color depths, including 24-bit color.

The PSP format can also store images in either compressed or uncompressed form. Compressed PSP images use either RLE or LZ77 (a variation of LZW) compression schemes. Version 5 of the PSP format introduced the ability to store *Photoshop*-like layer and transparency information with images as well.

Although *Paint Shop Pro* enjoys a sizeable following among many Windows users, few other graphics programs are actually compatible with the PSP format. Therefore, it's not recommended that you save your images as PSP files unless you intend to work exclusively with that program.

TABLE 3-11: PSP Characteristics

| *PSP Variant* | *Screen Resolutions* | *Compression* | *Lossy* | *Color Depths Supported* | *Platform Compatibility* |
|---|---|---|---|---|---|
| PSP 3.0/4.0 | Any | Yes, RLE or LZ77 | No | 8 or 24 bits | Windows* |
| PSP 5.0/6.0 | Any | Yes, RLE or LZ77 | No | 8 or 24 bits | Windows* |

* Denotes limited compatibility, usually obtained through graphic conversion software.

**NOTE:** Older versions of *Paintshop Pro* aren't fully compatible with the newer versions of the PSP format.

## TGA (Targa)

**Native Platform(s):** N/A

**File Extension(s):** .TGA, .tga

Truevision, Inc. (now Pinnacle) originally developed the Targa format in 1985 to use with its line of professional video capture products. However, since that time, the TGA format has seen many refinements and considerable use as an output format for many high-end painting and 3D programs.

TGA files can contain 8, 16, 24, or 32 bits of color information per pixel and can store images using a number of compression schemes including RLE and LZW. Although a technically solid graphics format, TGA's lack of support among many lower-end graphics applications makes it difficult to recommend for your graphics projects.

TABLE 3-12: TGA Characteristics

| *TGA Variant* | *Screen Resolutions* | *Compression* | *Lossy* | *Color Depths Supported* | *Platform Compatibility* |
|---|---|---|---|---|---|
| N/A | Any | No | No | 8, 16, 24, or 32 bits | DOS, Macintosh*, Windows* |
| N/A | Any | Yes, RLE, packbits, and LZW | No | 8, 16, 24, or 32 bits | DOS, Macintosh*, Windows* |

* Denotes limited compatibility, usually obtained through graphic conversion software.

## TIFF (Tagged Image File Format)

The Tagged Image File Format, or TIFF as it is commonly known, was designed with the intention of becoming the standard graphics file format. As such, it was engineered from the start to handle virtually any contingency that one might encounter. This provides TIFF files with the flexibility to support monochrome

and color images as well as several methods of compression including packbits and LZW.

Despite the good intentions of TIFF's designers, the format's flexibility has actually encouraged application developers to develop variations of the format that don't entirely stick to the published format specification. As a result, no single application can now claim universal TIFF compatibility and there will always be some version of the format that will cause problems with different graphics software.

Although TIFF is capable of storing almost any kind of graphic image, it's most commonly associated with multi-megabyte high-resolution scanned images.

TABLE 3-13: TIFF Characteristics

| *TIFF Variant* | *Screen Resolutions* | *Compression* | *Lossy* | *Color Depths Supported* | *Platform Compatibility* |
|---|---|---|---|---|---|
| TIFF | Any | Yes, LZW, packbits | No | 2, 4, 8, 16, or 24 bits | DOS, Macintosh, Windows, Linux, Java |

## XPM (X PixMap)

**Native Platform(s):** UNIX, Linux

**File Extension(s):** .XPM, .xpm

XPM is the de facto graphics standard for the X Window system running on UNIX and Linux computers. They are generated by a variety of applications including *Xpaint* and *GIMP.* XPM files can contain monochrome and color images. They are most frequently used to store icon data for X Window systems.

Unlike the other formats described in this chapter, XPM files do not use compression. Instead, images are saved as ASCII, C language source files. While this makes them versatile and easy to transmit through e-mail, images stored in the XPM format tend to be extremely large when compared to other graphic file formats.

Unfortunately, very few graphics packages outside the UNIX and Linux platforms can correctly handle XPM format files. This makes them difficult to exchange between other platforms.

TABLE 3-14: XPM Characteristics

| *XPM Variant* | *Screen Resolutions* | *Compression* | *Lossy* | *Color Depths Supported* | *Platform Compatibility* |
|---|---|---|---|---|---|
| XPM | Any | No | No | 2, 4, 8, 16, or 24 bits | Linux, UNIX |

## File Format Suitability for Arcade Game Graphics

Table 3-15 summarizes the overall suitability of each of the graphic file formats discussed here for storing various types of game related objects.

TABLE 3-15: File Format Suitability for Arcade Game Artwork

| *File Format* | *Title Screens* | *Menu Screens* | *Sprites* | *Backgrounds* |
|---|---|---|---|---|
| BMP | ✓ | ✓ | ✓ | ✓ |
| GIF | ✓ | ✓ | ✓ | ✓ |
| IFF/LBM | ✓ | ✓ | ✓ | ✓ |
| JPEG | ✓ | | | ✓ |
| PCX | ✓ | ✓ | ✓ | ✓ |
| PICT | ✓ | ✓ | ✓ | ✓ |
| PNG | ✓ | ✓ | ✓ | ✓ |
| PSD | ✓ | ✓ | ✓ | ✓ |
| PSP | ✓ | ✓ | ✓ | ✓ |
| TGA | ✓ | ✓ | ✓ | ✓ |
| TIFF | ✓ | ✓ | ✓ | ✓ |
| XPM | ✓ | ✓ | | |

**NOTE:** Neither the PSD nor PSP can be directly used in game graphics. They typically need to be "flattened" or optimized to discard extra information such as layers and transparency data.

**NOTE:** I left any mention of the FLI/FLIC format out of this table because it's primarily an animation format and not a general-purpose graphics file format. As such, it really isn't up to storing the static images required for game development. However, it is a useful animation playback format.

Figures 3-2 through 3-4 provide examples of the different types of arcade game artwork covered in Table 3-15.

Table 3-16 summarizes the overall suitability of the different graphic file formats described throughout this chapter.


FIGURE 3-2: Title/Menu Screen Example

FIGURE 3-3: Background Screen Example

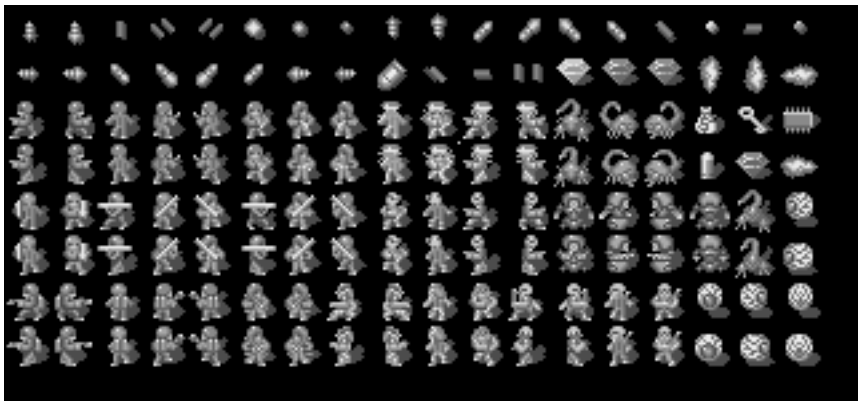


FIGURE 3-4: Sprite Screen Example

TABLE 3-16: File Format Suitability Comments

| *File Format* | *Comments* |
|---|---|
| BMP | ■ The standard graphics file format for all versions of Windows.<br>■ A good choice for exchanging graphics files between Windows and the Macintosh (provided that a version of QuickTime is installed on the Macintosh).<br>■ Not recommended for use with DOS, as very few DOS applications can read or write this format. |
| FLI/FLIC | ■ The best overall animation file format to use with DOS systems.<br>■ Can be used with Windows as long as you're sure your software is compatible with it. Fortunately, a fair number of Windows graphics packages are.<br>■ Definitely not recommended for the Macintosh due to extremely limited compatibility with software on that platform. |
| GIF | ■ The safest overall graphics format for reliably exchanging files between DOS, Windows, Macintosh, and Java.<br>■ The best overall graphics format for storing game objects for Java-based games. |
| IFF/LBM | ■ The best format to exchange graphics files between the Amiga, Atari ST, and DOS systems should you need to.<br>■ Generally not recommended for use on Windows unless you commonly work with programs that you know support it.<br>■ Has very limited support on the Macintosh and is not recommended for use on that platform. |
| JPEG | ■ Offers extremely high image compression rates; however, there's a trade-off between image quality and the level of compression.<br>■ Offers excellent compression for photographic images but generally not for game-oriented artwork.<br>■ Potentially useful for game title screens if they contain photo-realistic details. |
| PCX | ■ The best overall graphics format to use with DOS systems. Widely considered the de facto DOS graphics standard.<br>■ Not recommended for use with Windows unless you commonly work with programs that you know support it. However, it's the best format to use if you commonly find yourself exchanging files between the Windows and DOS platforms.<br>■ Has relatively limited support on the Macintosh and is not recommended for use on that platform except to exchange files with DOS systems. |
| PICT | ■ The standard graphics format on the Macintosh platform. As such, compatibility is assumed to be universal between Macintosh graphics applications.<br>■ Not recommended for use with DOS, as very few DOS graphics programs are compatible with it.<br>■ Has very limited use on Windows systems except to exchange graphics files with Macintosh systems. |

| *File Format* | *Comments* |
|---|---|
| PNG | ■ Not currently recommended for use in DOS, Windows, or Macintosh game development due to limited support among graphics tools and development tools at this time. However, this situation is expected to change in the very near future.<br>■ Potentially a very good file format for Java-based games. |
| PSD | ■ Offers good compatibility between higher-end Macintosh and Windows graphics applications.<br>■ Not recommend for DOS systems due to extremely limited support on that platform. |
| PSP | ■ Not recommended for exchanging files on DOS or Macintosh systems due to nonexistent application support on these platforms.<br>■ Offers only limited support among Windows graphics applications. Watch out! |
| TGA | ■ Not recommended for use with DOS, Windows, or the Macintosh unless you commonly work with programs that you know will support it. Usually, only expensive, high-end video and 3D animation programs do. |
| TIFF | ■ Although compatible with all major platforms, due to all of the format variations that exist, it is not a very reliable format for exchanging files between different systems. |
| XPM | ■ Only recommended for Linux and UNIX systems. Do not use it to exchange files with other platforms due to spotty compatibility with applications on other platforms.<br>■ Offers no compression so XPM files are excessively large, making it less than ideal for game-oriented artwork. |

## File Format Compression Savings

Tables 3-17 and 3-18 show how the various formats described here compare with each other when compressing different types of images. For testing purposes, I compressed two sample images that represent the two extremes: a photo-realistic title screen/menu screen and a typical arcade game sprite screen with large areas of flat color. Both images were originally saved as 640x480 Windows RGB BMP files and had file sizes of 900 KB each. I then calculated the results and indicated the file compression savings as both KB and percentages.

TABLE 3-17: File Format Compression Savings—Title Screen/Menu Screen

| *File Format (compression)* | *Final Image Characteristics* | *Original File Size* | *Compressed File Size* | *Total Savings* |
|---|---|---|---|---|
| BMP (RLE) | 640x480, 8 bits/pixel | 900 KB | 348 KB | 552 KB (61%) |
| GIF (LZW) | 640x480, 8 bits/pixel | 900 KB | 168 KB | 732 KB (81%) |
| IFF/LBM (RLE) | 640x480, 8 bits/pixel | 900 KB | 216 KB | 684 KB (76%) |

| *File Format (compression)* | *Final Image Characteristics* | *Original File Size* | *Compressed File Size* | *Total Savings* |
|---|---|---|---|---|
| JPEG (JFIF at 66% quality) | 640x480, 24 bits/pixel | 900 KB | 116 KB | 784 KB (87%) |
| PCX (RLE) | 640x480, 8 bits/pixel | 900 KB | 284 KB | 616 KB (68%) |
| PICT (RLE) | 640x480, 8 bits/pixel | 900 KB | 312 KB | 588 KB (65%) |
| PNG (LZ77) | 640x480, 8 bits/pixel | 900 KB | 160 KB | 740 KB (82%) |
| PSD (RLE) | 640x480, 8 bits/pixel | 900 KB | 308 KB | 592 KB (65%) |
| PSP (LZW) | 640x480, 8 bits/pixel | 900 KB | 168 KB | 732 KB (81%) |
| TGA (RLE) | 640x480, 8 bits/pixel | 900 KB | 320 KB | 580 KB (64%) |
| TIFF (LZW) | 640x480, 8 bits/pixel | 900 KB | 168 KB | 732 KB (81%) |
| XPM (ASCII) | 640x480, 8 bits/pixel | 900 KB | 680 KB | 220 KB (24%) |

TABLE 3-18: Format Compression Savings—Sprite Screen

| *File Format (compression)* | *Final Image Characteristics* | *Original File Size* | *Compressed File Size* | *Total Savings* |
|---|---|---|---|---|
| BMP (RLE) | 640x480, 8 bits/pixel | 900 KB | 124 KB | 776 KB (86%) |
| GIF (LZW) | 640x480, 8 bits/pixel | 900 KB | 84 KB | 816 KB (90%) |
| IFF/LBM (RLE) | 640x480, 8 bits/pixel | 900 KB | 212 KB | 688 KB (76%) |
| JPEG (JFIF at 66% quality) | 640x480, 24 bits/pixel | 900 KB | 100 KB | 800 KB (89%) |
| PCX (RLE) | 640x480, 8 bits/pixel | 900 KB | 120 KB | 780 KB (87%) |
| PICT (RLE) | 640x480, 8 bits/pixel | 900 KB | 128 KB | 772 KB (85%) |
| PNG (LZ77) | 640x480, 8 bits/pixel | 900 KB | 76 KB | 824 KB (92%) |
| PSD (RLE) | 640x480, 8 bits/pixel | 900 KB | 128 KB | 772 KB (85%) |
| PSP (LZW) | 640x480, 8 bits/pixel | 900 KB | 84 KB | 816 KB (90%) |
| TGA (RLE) | 640x480, 8 bits/pixel | 900 KB | 124 KB | 776 KB (86%) |
| TIFF (LZW) | 640x480, 8 bits/pixel | 900 KB | 84 KB | 816 KB (90%) |
| XPM (ASCII) | 640x480, 8 bits/pixel | 900 KB | 356 KB | 544 KB (60%) |

**NOTE:** I didn't make any mention of the FLI/FLIC format in Tables 3-17 and 3-18 because it's primarily an animation format and not a general-purpose graphics file format. Mentioning it here would be like comparing apples and oranges.

Overall, the results fell into line with my expectations but added a few surprises as well. Not surprisingly, JPEG offered the best compression rate on the photo-realistic title screen, averaging a staggering 87% compression saving. The LZ77-based PNG format produced the second highest savings with 82%. The

LZW-based file formats such as GIF, PSP, and TIFF followed close behind, averaging 81%. Meanwhile, RLE-based file formats such as BMP, IFF, PICT, PCX, PSD, and TGA all pulled up the rear, averaging only a 66.5% reduction in file size. What was interesting was the fact that the LZW compression schemes yielded identical savings while the RLE compression schemes seemed to vary quite a bit in the savings they offered.

However, the results were a lot closer when it came to dealing with the flat color sprite screen. Here, the PNG format yielded the highest compression rate at 92%. GIF, PSP, TGA, and TIFF followed right behind, averaging 90%. And this time, the other formats weren't too far apart, averaging 84%. This time, JPEG produced a file with an 89% compression saving, but the image's quality was dramatically and negatively affected.

**NOTE:** Different graphics software will produce different results. This is because some programs offer better file compression logic than others do. As a result, your compression savings may very well differ from my findings.

Despite these impressive numbers, don't fall into the trap of using a particular graphic file format just because it offers a great compression rate. A file format is only as good as the program that can read it. Unless you have special or unusual needs, you should always consider a file format's compatibility with different systems and applications over its ability to compress files.

## Caveats for Working with Graphic File Formats

Although we use them every day and they have proven a reliable mechanism for exchanging data between machines, graphics files are still imperfect. Things can sometimes go wrong. The two most common problems are:

- File corruption
- Incompatible versions

### File Corruption

Compression schemes do their work by essentially scrambling and rearranging the contents of a file. The vast majority of the time they work well and the compressed file decompresses without any problems or glitches. However, every now and then, you may come across a file that gets corrupted while being compressed and/or saved.

File corruption can rear its ugly head for any number of reasons. For instance, there might be a bug in how a certain program implemented a particular file-

compression routine or a system crash might have caused the file save process to abort before all of the pertinent information was written to disk.

Therefore, in order to minimize the likelihood of this happening in the future, I recommend that you always save at least two versions of every important graphics file you create. Also, avoid making changes to important files while in an unstable environment. In other words, don't continue to work on your file if your system or graphics program is about to crash. You're only asking for problems. In addition, I strongly suggest that you follow the advice I give in Chapter 4 and regularly back up your files.

## Incompatible Versions

A number of the file formats mentioned here can be used across different computer platforms with little or no additional manipulation. However, occasionally you will encounter a graphics file that should import correctly into a certain program and just doesn't. Often, the file isn't corrupt, but rather, you've discovered an unpleasant side effect of progress: incompatible file format versions.

Yes, that's right. As software packages evolve and improve, they often introduce new file formats and don't always retain 100% backward compatibility with older versions of their file formats. The problem is compounded by the fact that many third-party software vendors are slow to update their software to reflect these changes. As a result, it's entirely possible for you to save your files in a particular version of a file format that one program supports and another doesn't!

The best and easiest way to cope with this problem is to simply pay attention when saving or exporting your images. Often, the software you're using will provide one or more save options. Among other things, these options will allow you to determine the version that your file is saved as. In addition, most graphics programs will inform you when you've chosen an outdated or obscure version of a file format that might cause compatibility problems with other software. Heed its advice.

Just to drill the point home, here are some examples of common graphic file formats that might cause you grief and some solutions on how to deal with them:

- **BMP**—Always save using the RGB subformat. This ensures universal compatibility with different applications, as RLE compression isn't supported by all BMP-compatible software.
- **GIF**—Choose 87a as it is supported by all GIF-compatible software, especially when using older software.
- **IFF/LBM**—Always save the file as compressed (RLE or packbits). Some programs will allow you to save uncompressed IFF/LBM files but many programs aren't compatible with them.

- **PCX**—Always choose version 5 unless you plan to use your files with 14-year-old software or want to lose most of your artwork's color information!
- **PSD**—Choose version 2.0 or 2.5 if you're not using layers.
- **TIFF**—Always choose uncompressed or LZW. Also, be sure to indicate the appropriate *byte order*, Intel or Motorola, in order to suit the target platform for the file. Doing both of these things can greatly reduce the compatibility issues associated with this particular file format.

# Graphic File Format Recommendations

Unfortunately, I can't recommend one all-inclusive file format simply because everyone has his or her own unique requirements. Therefore, in order for you to determine the best file format(s) to use, you should look closely at such game development related issues as:

- Graphics applications
- Development environment
- Artwork type
- Operating systems and platform

## Graphics Applications

The graphics applications you use can have a major influence on which format you ultimately select. Ask yourself these simple questions: What file formats are the graphics programs I use compatible with? Can they read or write files in the formats I need? What programs will I eventually use? Weigh your answers and choose a file format accordingly.

**NOTE:** Don't decide these issues solely on compatibility with your preferred graphics applications. As you'll see in Chapter 4, there's an entire class of image conversion software that can convert graphics files from one format to another. In fact, to make things easier for you, I've included some of the better programs of this type on the book's accompanying CD-ROM. You can find more information by referencing Appendix B of this book.

## Development Environment

Your development environment can also have a significant influence on which format you eventually decide to go with. For example, your favorite C graphics library might only work with PCX files while another might work with all popular file formats. The last thing you want is a hard drive of image files that won't work with your favorite development tools. Therefore, you need to find out what file

format restrictions your particular development environment places on you and then select the appropriate format.

Generally speaking, I've found that:

- Multimedia authoring tools such as Macromedia's *Director* and the Clickteam's *The Games Factory* tend to offer the best built-in support for graphic file formats.
- Visual programming languages such as Delphi and Visual Basic tend to offer support for different graphic formats via third-party controls, components, and DLLs. However, they will typically also support at least one or more common graphic formats internally as well.
- Character-oriented languages such as C, Assembler, and Pascal tend to offer the poorest native support and often need one or more external graphics libraries to provide the appropriate compatibility with the different file formats discussed in this chapter.

## Artwork Type

The type of artwork you're designing can also play an important role in the selection process. As you saw in Tables 3-15 and 3-16, certain file formats work better on some types of graphics files than others. You need to take this into account and consider your available options accordingly. For example, if you're designing game sprites, you'll definitely want to use a lossless compression format rather than a lossy one so you can preserve the original integrity of your artwork. Similarly, if you are working with photo-realistic artwork, a file format that offers good compression and that better supports 24-bit images may be the way to go.

## Operating System and Platform

The need to remain compatible with different computer platforms and operating systems also impacts your decision. For example, if you're trying to port the game you're working on to the Macintosh, then PICT might be a good solution. Alternatively, if you need to maintain compatibility with your favorite DOS graphics editor, BMP might not be the best choice. Ultimately, the choice is an easy one to make as long as you think about your needs and requirements logically and then weigh your options carefully. Refer to Table 3-19 for suggestions on which file formats to use on which platforms.

TABLE 3-19: Recommended File Format by Platform

| *File Format* | *DOS* | *Windows 3.1, 95, 98, NT 4.0, and 2000* | *Macintosh* | *Linux* | *Java* |
|---|---|---|---|---|---|
| BMP | ✓ | ✓ | | ✓ | |
| FLI/FLIC | ✓ | ✓ | | | |
| GIF 87a/89a | ✓ | ✓ | ✓ | ✓ | ✓ |
| IFF/LBM | ✓ | ✓ | | | |
| JPEG | ✓ | | ✓ | ✓ | ✓ |
| PCX | ✓ | ✓ | | ✓ | |
| PICT | | | ✓ | | |
| PNG | ✓ | ✓ | ✓ | ✓ | ✓ |
| PSD | | ✓ | ✓ | | |
| PSP | | ✓ | | | |
| TGA | ✓ | ✓ | ✓ | ✓ | |
| TIFF | ✓ | ✓ | ✓ | ✓ | ✓ |
| XPM | | | | ✓ | |