The XPM Format and Library

A Tutorial

Arnaud Le Hors
January 1996

Introduction

XPM stands for X PixMap. XPM encompasses both a format and a C library that can be used to save and retrieve X pixmaps to and from various forms of storage. XPM fills a gap in the X Window System software which defines only the XBM (X BitMap) format for pixmaps that are 1-bit deep. As a result, XPM has become the de facto standard in many commercial and non-commercial applications. Moreover, the use of XPM is going to increase because Motif 2.0 includes the XPM library and because the Common Desktop Environment specifies that icons must be stored in either XBM or XPM format. At this time, several vendors distribute the XPM library as contributed software on the platforms they sell.

This tutorial describes how to use XPM to create and manipulate icons in X applications. We present several examples that use XPM to solve common pixmap-related problems. This article covers version 3 of the XPM format and version 3.4 of the XPM library.

The XPM Format

While many image formats are available, the XPM format was designed to fit the special need of storing icons. The following points make icons different from regular images:

- Icons are usually small.
- Icons may need to be drawn differently depending on whether they are being displayed on a color or monochrome screen.
- Icons may want to define symbolic colors, such as “Foreground” and “Background”, that can be changed depending on the context in which the icons are used.
- Icons can be of any shape, not just rectangular.

XPM is an ASCII format with a C syntax that allows you to include XPM files in C and C++ programs, which saves you the trouble of having to maintain and read separate files containing the definition. In addition, the ASCII format makes XPM portable and allows you to use any external compression program. Using *gzip* in this way can yield a better compression rate than the built-in GIF algorithm. This feature also means that you can take advantage of any new algorithms when they are developed.

The XPM format allows you to define colors with various default values in order to specify how they should be allocated. Thus you can control how a color image is displayed on a monochrome screen, for instance. You can also define symbolic names for colors so the default values can be overridden dynamically. This feature is especially useful in the X environment where application resources can be customized by the user. For example, defining a

Background symbolic color name lets the user modify the background color of the icon at load time. In addition, with XPM you can use the color None for “transparent” pixels. Although the X Window System does not actually provide such a feature, you can achieve this visual result in several ways. Finally, the XPM format can store hotspot coordinates. The XPM format can also store application-specific data, such as the coordinates of a connection port, as an extension to the format.

An example of what you can do with XPM is store a red bullet icon in such a way that you can use, without any duplication, the same file for building a green or blue bullet pixmap as well. With GIF you would need a different file for each color.

THE XPM LIBRARY

The XPM API consists of two sets of Xlib-level C functions and structures. The first set enables you to perform simple operations such as reading a pixmap and displaying it on a Motif PushButton or saving a pixmap to an XPM file. Basically, this set contains functions for saving and retrieving `Pixmap` and `XImage` objects to and from files, buffers (files in memory), and data (included files). The XPM library is designed to let take advantage of features in the XPM format, so you can use it to override symbolic colors and build icons with transparent color.

The second set of routines, which function at a lower level and are called by routines from the first set, provides you with advanced features for writing applications such as pixmap editors and image format converters. These routines are also useful in applications in which you want to cache XPM files. This set contains functions for saving and retrieving `XpmImage` objects to and from the same forms of storage (files, buffers, and data), plus functions to build `XImage` and `Pixmap` objects from an `XpmImage` object and vice versa.

AVAILABILITY

Since XPM is not an X Consortium standard, the XPM library may not be present on your system. If this is the case, you should get the latest version of XPM via anonymous FTP from *ftp.x.org* in */contrib/libraries*, or from *koala.inria.fr* in */pub/xpm*. You can also get it from the X11R6 contributed software package distributed by the X Consortium. The XPM library is known to compile on most systems, so follow the instructions in the *README* file and everything should work fine.

You can get support by sending any questions or suggestions related to XPM to *lehors@sophia.inria.fr*. You can subscribe to the mailing list *xpm-talk@sophia.inria.fr* by simply sending a request to *xpm-talk-request@sophia.inria.fr.* Since XPM is widely used, you can also obtain help from other users through newsgroups such as *comp.windows.x.*

The example XPM files and programs shown in this tutorial are available via anonymous FTP from *koala.inria.fr* as */pub/xpm/xpm_examples.tar.gz.*

DEFINING PIXMAPS

The first step in using XPM is to define a pixmap, which you can do using the XPM format. In version 3, a pixmap is simply an array of character strings that is composed as follows:

```
/* XPM */
static char* <pixmap_name>[] = {
"Values-string",
"Colors-strings",
"Pixels-strings",
"Extensions-strings",
};
```

Each string is surrounded by double-quotes and separated by a comma. In a string containing multiple values, the words are separated by white space, which can be composed of space and tab characters. Regular C comments can be placed anywhere outside of the character strings; the pixmap must have a header comment that contains only the word `XPM`.

The *Values-string* is a single string that must contain four integers that correspond to the width and height of the pixmap, the number of colors in the pixmap, and the number of characters per pixel in the color specification. The string can optionally specify the hotspot coordinates and the word XPMEXT if the pixmap includes an *Extensions-string*. The format of the string is as follows:

```
"width height ncolors cpp [x_hotspot y_hotspot] [XPMEXT]
```

The *Colors-strings* section contains as many strings as there are colors. Each string specifies the characters used to represent the color, the type of the color, and the actual color value. Each string has the following format:

```
"chars key value ..."
```

The *chars* value contains *cpp* characters; these characters are used to represent pixels of that color in the actual pixmap. A color specification can have multiple key/value pairs, where *value* is the specified color and *key* is a keyword describing the type of the value. XPM defines the following keys:

- m for a monochrome visual
- c for a color visual
- g4 for 4-level grayscale
- g for grayscale with more than 4 levels
- s for a symbolic name

The actual color can be specified by giving the colorname, a '#' followed by the RGB code in hexadecimal, or a '%' followed by the HSV (Hue, Saturation, Value) code.† In addition, the value None can be given as a color name to mean the transparent color.

The *Pixels-strings* section contains the actual pixmap definition. This section consists of *height* strings. Each string contains *width* * *cpp* characters. Every *cpp* length string must be one of the colors defined in the *Colors-strings* section.

The final section is the optional *Extensions-strings* section, which must be indicated in the *Values-string*. This section can contain data for multiple extensions. The data for a single extension can be a single string or multiple strings. If the data is a single string, the extension takes the following form:

```
"XPMEXT extension-name extension-data",
```

If the extension uses multiple strings, the form is as follows:

```
"XPMEXT extension-name",
"extension-data",
...
```

In either case, if this section is included, it must end with the string XPMENDEXT.

A Simple Example

The following pixmap definition shows you how to create the red bullet icon shown in Figure 1 Since this documentation is printed in black and white, the color icons in the figures appear in various shades of gray.

```
/* XPM */
static char * bullet_xpm[] = {
/* width height number_of_colors chars_per_pixel */
/* colors */
"25 25 2 1",
"X    c black",
".    c red",
/* pixels */
```

† The HSV codes are not supported by the current XPM library.

```
"XXXXXXXXXXXXXXXXXXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXX",
"XXXXXXXXXX.....XXXXXXXXXX",
"XXXXXXXX.........XXXXXXXX",
"XXXXXX.............XXXXXX",
"XXXXX...............XXXXX",
"XXXXX...............XXXXX",
"XXXX.................XXXX",
"XXXX.................XXXX",
"XXX...................XXX",
"XXX...................XXX",
"XXX...................XXX",
"XXX...................XXX",
"XXX...................XXX",
"XXXX.................XXXX",
"XXXX.................XXXX",
"XXXXX...............XXXXX",
"XXXXX...............XXXXX",
"XXXXXX.............XXXXXX",
"XXXXXXXX.........XXXXXXXX",
"XXXXXXXXXX.....XXXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXX"};
```

The comments point out the different sections of the format. The icon is a 25 by 25 pixmap that uses the colors black and red. These colors are represented by the single characters X, and . (dot), respectively. This icon is stored in a file called *bullet.xpm*, which is relevant to a discussion about using the icon with the XPM library later in this tutorial..

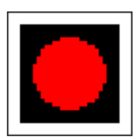

FIGURE 1: *THE RED BULLET ICON*

However, when this icon is displayed on a monochrome screen, it will appear all black, as shown in Figure 2. This happens because the color red recerts to black on a monochrome displayTo avoid this problem, you can force the color red to revert to white instead of black by changing the definition of the . (dot) character in `bullet.xpm` to the following:

```
".    c red m white",
```

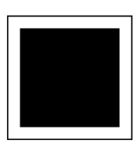

FIGURE 2: *THE RED BULLET ICON ON A MONOCHROME SCREEN*

In this color specification, we use multiple key/value pairs. We indicate that the . (dot) represents the color value `red` on a color display by using the keyword `c`, while on a monochrome display the character means `white`, because we use the `m` keyword. Figure 3 shows how the new icon looks on a monochrome display.

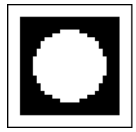

FIGURE 3: *THE NEW RED BULLET ON A MONOCHROME SCREEN*

This feature of XPM allows you to combine several icon definitions in a single file. You can use dithering in a monochrome icon and a solid color in the color version. We use this technique in the following example:

```
/* XPM */
static char * newbullet_xpm[] = {
“25 25 3 1”,
“X    c black”,
“.    c red m black”,
“     c red m white”,
“XXXXXXXXXXXXXXXXXXXXXXXXX”,
“XXXXXXXXXXXXXXXXXXXXXXXXX”,
“XXXXXXXXXXXXXXXXXXXXXXXXX”,
“XXXXXXXXXX. . .XXXXXXXXXX”,
“XXXXXXXX . . . . XXXXXXXX”,
“XXXXXX. . . . . . .XXXXXX”,
“XXXXX. . . . . . . .XXXXX”,
“XXXXX . . . . . . . XXXXX”,
“XXXX . . . . . . . . XXXX”,
“XXXX. . . . . . . . .XXXX”,
“XXX. . . . . . . . . .XXX”,
“XXX . . . . . . . . . XXX”,
“XXX. . . . . . . . . .XXX”,
“XXX . . . . . . . . . XXX”,
“XXX. . . . . . . . . .XXX”,
“XXXX. . . . . . . . .XXXX”,
“XXXX . . . . . . . . XXXX”,
“XXXXX . . . . . . . XXXXX”,
“XXXXX. . . . . . . .XXXXX”,
“XXXXXX. . . . . . .XXXXXX”,
“XXXXXXXX . . . . XXXXXXXX”,
“XXXXXXXXXX. . .XXXXXXXXXX”,
“XXXXXXXXXXXXXXXXXXXXXXXXX”,
“XXXXXXXXXXXXXXXXXXXXXXXXX”,
“XXXXXXXXXXXXXXXXXXXXXXXXX”};
```

This definition creates a red bullet icon that uses three color specifications instead of two. The red pixels have been put in two groups: one that is black on a monochrome screen and the other that is white on a monochrome screen. Thus, the bullet is red on a color screen, just like the one shown in Figure 1, and it is gray on a monochrome screen, as shown in Figure 4.

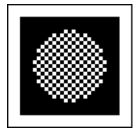

FIGURE 4: *THE DITHERED RED BULLET ICON ON A MONOCHROME SCREEN*

SYMBOLIC COLORS

With XPM, it's very easy to define a symbolic color name, such as Foreground. You change only a single line in the *bullet.xpm* file. To define a symbolic color name, use the following color specification for the color `red`:

```
".    c red s Foreground",
```

All we have done is add the `s` keyword, which means that the following value, `Foreground`, is the symbolic name of this color. As a result, you can dynamically define Foreground to be blue or any other color and get a bullet icon of any color from the same file. In our later discussion, we refer to this new icon as *bullet_symb.xpm*.

The way you specify the value of Foreground depends on the application. Later in this tutorial we describe how you can use the XPM library for this purpose. In the mean time, you can test out this functionality using the *sxpm* program, which is part of the XPM distribution. This program is designed to show XPM files. It provides you with a set of options that you can use to trigger most of the features of XPM, such as the color overriding mechanism and the transparent color. You can obtain a short description of all the supported options by entering the command *sxpm -help*, or by looking at the manual page for more information. Use the following command to set the Foreground to blue:

```
% sxpm bullet_symb.xpm -s Foreground blue
```

TRANSPARENCY

XPM can handle icons of any shape. For instance, the bullet icon would be more appealing with a circular shape instead of a square shape. One simple way to create a shaped icon is to use the color None instead of black, in order to have a transparent background. Again, you cahnge only a single line in the XPM file *bullet.xpm*. This time change the color definition for X to the following:

```
"X    c none",
```

We will call this new icon *bullet_transp.xpm*. It is illustrated in Figure 5.

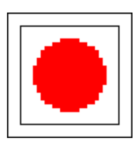

FIGURE 5: *THE RED BULLET ICON WITH A CIRCULAR SHAPE*

Note that the icon is still defined in a rectangle, but it looks like it is not. Here again, the way that the transparent color is handled depends on the application, and also, in some cases, on whether your X server provides the Shape Extension. We describe how to use the XPM library to achieve this transparent effect later in the tutorial.

EXTENSIONS

The following example shows how you can specify hotspot coordinates and private data, such as the coordinates of a connection port, in an XPM definition:

```
/* XPM */
static char * smallbullet_xpm[] = {
/* width height number_of_colors chars_per_pixel x_hotspot y_hotspot */
“5 5 2 1 2 2 XPMEXT”,
“X    c none”,
“.    c red m white s Foreground”,
“X...X“,
“.....”,
“.....”,
“.....”,
“X...X“,
“XPMEXT Ports”,
“2 0 input”,
“XPMENDEXT”};
```

The hotspot is located at the point 2, 2. The private data is included as an extension. A port is defined with its coordinates and its name in a private data format.

A COLORFUL EXAMPLE

As a final example, we want to show you a more colorful icon, to give you a feel for the power of XPM. The following XPM definition creates a pixmap of Bart Simpson, as shown in Figure 6:

```
/* XPM */
static char * bart_xpm[] = {
/* width height ncolors chars_per_pixel */
“64 64 6 1”,
/* colors */
“  m None  c None s None”,
“. m black c black”,
“X m white c #faf080”,
“o m white c white”,
“O m white c red  s Shirt”,
“+ m white c blue s Shorts”,
/* pixels */
“                                                                “,
“                           ......                               “,
“                     .. ... ..X..                               “,
“                .. . ....X..XX.X.                               “,
“            ... .X...XXX.XX.XXXX.                               “,
“             ....XXX.XXXXXXXXXXX.                               “,
“             .XX.XXXXXXXXXXXXXXX.                               “,
“             ..XXXXXXXXXXXXXXXXX.                               “,
“              .XXXXXXXXXXXXXXXXX.                               “,
“              ..XXXXXXXXXXXXXX....                              “,
“               .XXXXXXXXX...X.ooo..                             “,
“               ..XXXXXXX.ooo....o..                             “,
“                .XXXXXX.ooooo.ooo..                             “,
“                 .XXXXX.o..oo.......                            “,
“                 .XXXXX.ooooo...XXX..                           “,
“                 .XXXXXX.ooo..XXXXX.                            “,
“                  .XXXXXX...XXXXX....                           “,
“                  .XXXXXXXXXXXX...X...                          “,
“                   .XXXXXXXXXXXXXXXX...                         “,
“                   .XXXXXXXXXXXXXXXXX.                          “,
```

```
"                   .XXXXXXXXXXXXXXXX...                         ",
"                   ..XXXXX.XXXXXXX....                          ",
"                    .XXXXX.........                             ",
"                    .XXXX.XXXXXX..                              ",
"                     .XXXXXXXX...                               ",
"                     .XXXXXXXX..                                ",
"                     .XXXXXXXX.                                 ",
"                   .......XXXX.                                 ",
"                  ..OOOOO........                               ",
"                  .OOOOOOOOOOO.O..                              ",
"                  .OOOOOO.OOOO.O..                              ",
"                  .OOOOOO.OOOO.OO.                              ",
"                  .OOOOOO..OOO.OO..                             ",
"                  .OOOOOOO.OOO..O..                             ",
"                  .........OOOO..O.                             ",
"                   ..XXX.OOOOOOO.O.                             ",
"                   .XXXX.OOOOOOOO.                              ",
"                  ..XXXX.OOOOOOOO..                             ",
"                 .O.XXXX.OOOOOOOOO.                             ",
"                .OO.XXXX.OOOOOOOOO.                             ",
"               .OOO.XXXX.OOOOOOOOOO.                            ",
"               .OOO..XXX.OOOOOOOOOO.                            ",
"               .OOOO.XXX.OOOOOOOOOO.                            ",
"               .OOOO.XXX.OOOOOOOOOO.                            ",
"               ......XXXX.OOOO.......                           ",
"                .+++.XXXX......++++.                            ",
"                 .+.XX.XXXX.+++++++.                            ",
"                 .+.X.XX....+++++++.                            ",
"                 .+..XX.++++++++++.                             ",
"                 .++.X..+++++++++++.                            ",
"                 .+++++++++........                             ",
"                 ...........XXX.                                ",
"                  ..XXXX.   .XXX.                               ",
"                   ..XXX.   .XXX.                               ",
"                    .XXX.   .XXX.                               ",
"                    .XXX.   .XXX.                               ",
"                    .....   .....                               ",
"                   .oooo....oooo..                              ",
"                    ....X..X....X..                             ",
"                   .XXXX..X.XXXX..                              ",
"                   .XXXXXX..XXXXXX..                            ",
"                   .XXXXXXX.XXXXXXX.                            ",
"                   .X.X.X.X.X.X.X.X.                            ",
"                   ........X.......                             "};
```

USING PIXMAPS

Now that you've seen a number of examples of pixmaps defined with XPM, it's time to look at how you actually use them. In this section, we start with a simple example of putting an XPM icon on a Motif PushButton. Then we move on to more complex examples that demonstrate the main features of the XPM library. This section doesn't cover all of the XPM functions; for full details see the reference pages that follow this tutorial.

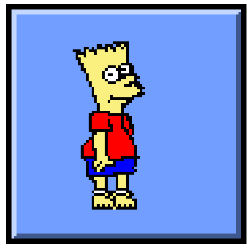

FIGURE 6: *A COLORFUL PIXMAP*

READING AN XPM FILE

In order to use an XPM icon, you must read an XPM file to create a pixmap. The following program demonstrates how to use the XPM library to read the XPM file *bullet.xpm* and create a pixmap. Once we have the pixmap, we can put it on a Motif PushButton widget as shown in Figure 7.

```
#include <stdio.h>
#include <Xm/PushB.h>
#include <X11/xpm.h>

main(int argc, char *argv[])
{
    XtAppContext app;
    Widget top, button;
    int status;
    Pixmap pixmap;

    /* create a simple hierarchy with a PushButton widget */
    top = XtAppInitialize(&app, "BulletButton", NULL, 0, &argc, argv,
                          NULL, NULL, 0);
    button = XmCreatePushButton(top, "button", NULL, 0);

    /* create a pixmap from the XPM file */
    status = XpmReadFileToPixmap(XtDisplay(top),
                                 XRootWindowOfScreen(XtScreen(top)),
                                 "bullet.xpm", &pixmap, NULL, NULL);
    if (status != XpmSuccess) {
        fprintf(stderr, "XpmError: %s\n", XpmGetErrorString(status));
        exit(1);
    }
    /* then put the pixmap onto the button */
    XtVaSetValues(button,
                  XmNlabelType, XmPIXMAP,
                  XmNlabelPixmap, pixmap,
                  NULL);
    XtManageChild(button);
    XtRealizeWidget(top);
    XtAppMainLoop(app);
}
```

The relevant function call in this example is `XpmReadFileToPixmap()`. This function creates and returns a pixmap from the specified XPM file. As with all of the main XPM functions, this function returns a status code that

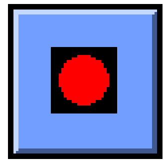

FIGURE 7: *THE RED BULLET ICON ON A PUSHBUTTON*

can be used to detect and treat a possible error. As shown in this example, you can use `XpmGetErrorString()` to build an error message. This function simply returns a character string related to the given error code.

Since the XPM functions and structures are declared in the XPM header file *xpm.h*, you must include this header file in your program. You must also link your program with the XPM library. The *button* program can be compiled with the following *Imakefile*:

```
        DEPLIBS = $(DEPXMULIB) $(DEPXTOOLLIB) $(DEPXLIB)
LOCAL_LIBRARIES = -lXm -lXpm $(XMULIB) $(XTOOLLIB) $(XLIB)
           OBJS = button.o
           SRCS = button.c

ComplexProgramTarget(button)
```

You can also use the following command with X11R5:

```
% cc -o button button.c -lXm -lXpm -lXmu -lXt -lXext -lX11
```

With X11R6, use this command instead:

```
% cc -o button button.c -lXm -lXpm -lXmu -lXt -lSM -lICE -lXext -lX11
```

USING PIXMAP ATTRIBUTES

In the example described above, the last two parameters passes to the `XpmReadFileToPixmap()` function are `NULL`. However, instead of passing `NULL` as the last parameter, you can pass a reference to an `XpmAttributes` structure to specify characteristics of the pixmap, as well as to retrieve useful data about it. You can use the `XpmAttributes` structure to specify attributes such as the colormap, the visual, and default colors. The `XpmReadFileToPixmap()` routine uses the values you specify to create the pixmap and then it fills in the `XpmAttributes` structure with information about the resulting pixmap. This structure is defined as follows:

```
typedef struct {
  unsigned long valuemask;     /* Specifies which components are defined */
  Visual *visual;              /* Specifies the visual to use */
  Colormap colormap;           /* Specifies the colormap to use */
  unsigned int depth;          /* Specifies the depth */
  unsigned int width;          /* The width of the created pixmap */
  unsigned int height;         /* The height of the created pixmap */
  unsigned int x_hotspot;      /* The x hotspot's coordinate */
  unsigned int y_hotspot;      /* The y hotspot's coordinate */
  unsigned int cpp;            /* Specifies the number of char per pixel */
  Pixel *pixels;               /* List of used color pixels */
  unsigned int npixels;        /* Number of pixels */
  XpmColorSymbol *colorsymbols;/* Array of color symbols to override */
  unsigned int numsymbols;     /* Number of symbols */
  char *rgb_fname;             /* RGB text file name */
  unsigned int nextensions;    /* Number of extensions */
  XpmExtension *extensions;    /* Array of extensions */
```

```
    int ncolors;                /* Number of colors */
    XpmColor *colorTable;       /* Color table pointer */
    char *hints_cmt;            /* Comment of the hints section */
    char *colors_cmt;           /* Comment of the colors section */
    char *pixels_cmt;           /* Comment of the pixels section */
    unsigned int mask_pixel;    /* Transparent pixel color table index */
    /* Color Allocation Directives */
    unsigned int exactColors;   /* Only use exact colors for visual */
    unsigned int closeness;     /* Allowable RGB deviation */
    unsigned int red_closeness; /* Allowable red deviation */
    unsigned int green_closeness;/* Allowable green deviation */
    unsigned int blue_closeness; /* Allowable blue deviation */
    int color_key;              /* Use colors from this color set */
    Pixel *alloc_pixels;        /* List of allocated color pixels */
    unsigned int nalloc_pixels; /* Number of allocated pixels */
} XpmAttributes;
```

As you can see, this is a rather large structure. In order to be able to specify which fields are actually set, the structure also contains a value mask. The `valuemask` field is a bitwise inclusive OR of the valid attribute mask bits. If you are familiar with Xlib, you'll recognize this technique as similar to how you handle GCs. The one difference here is that the value mask is part of the structure, instead of being passed separately to different routines. The structure is designed this way because XPM functions can modify values in the structure to let you know whether some optional data, such as hotspot coordinates, is available. For complete details on the `XpmAttributes` structure and the valid attribute masks, see the reference page for the structure.

The following code fragment shows a modified version of the previous program. In this case, an `XpmAttributes` structure is used to retrieve the pixmap size in order to set the size of the PushButton widget:

```
XpmAttributes attributes;
...
/* initialize the XpmAttributes valuemask */
attributes.valuemask = 0;

/* create a pixmap from the XPM file */
status = XpmReadFileToPixmap(XtDisplay(top),
                             XRootWindowOfScreen(XtScreen(top)),
                             "bullet.xpm", &pixmap, NULL, &attributes);
if (status != XpmSuccess) {
    fprintf(stderr, "XpmError: %s\n", XpmGetErrorString(status));
    exit(1);
}

/* resize the button to the pixmap size plus some margins,
 * and put the pixmap onto the button  */
XtVaSetValues(button,
              XmNwidth, attributes.width + 50,
              XmNheight, attributes.height + 25,
              XmNlabelType, XmPIXMAP,
              XmNlabelPixmap, pixmap,
              NULL);
```

The `XpmReadFileToPixmap()` function sets the `width` and `height` components of the given `XpmAttributes` structure. We use these values to set the size of the widget.

Note that we intialize the `valuemask` of the `XpmAttributes` structure by setting it to zero before calling `XpmReadFileToPixmap()`. This set is very important because the routine checks the structure and uses the specified fields in creating the pixmap. If `valuemask` is not set to some valid value, unpredictable errors can occur. Forgetting to initialize `valuemask` is one of the most common errors that programmers make when using the XPM library.

One common use of the `XpmAttributes` structure is to override the colors of an XPM icon. For example, if you want the PushButton to display a green bullet icon when it is armed, as shown in Figure 8, you do not need a new XPM file. All you have to do is add a symbolic color name like Foreground to your XPM file, as explained earlier, and then use this symbolic color to create a green bullet icon from the same file. The following program demonstrates this use of the XpmAttributes structure, using the *bullet_symb.xpm* pixmap defined earlier.

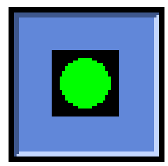

FIGURE 8: *A GREEN BULLET ICON ON AN ARMED PUSHBUTTON*

```
#include <stdio.h>
#include <Xm/PushB.h>
#include <X11/xpm.h>

main(int argc, char *argv[])
{
    XtAppContext app;
    Widget top, button;
    int status;
    Pixmap pixmap, armpixmap;
    XpmAttributes attributes;
    XpmColorSymbol symbol;

    /* create a simple hierarchy with a PushButton widget */
    top = XtAppInitialize(&app, "BulletButton", NULL, 0, &argc, argv,
                          NULL, NULL, 0);
    button = XmCreatePushButton(top, "button", NULL, 0);

    /* create a red bullet pixmap from the XPM file */
    status = XpmReadFileToPixmap(XtDisplay(top),
                                 XRootWindowOfScreen(XtScreen(top)),
                                 "bullet_symb.xpm", &pixmap, NULL, NULL);
    if (status != XpmSuccess) {
        fprintf(stderr, "XpmError: %s\n", XpmGetErrorString(status));
        exit(1);
    }
    /* set the Foreground to green */
    symbol.name = "Foreground";
    symbol.value = "green";

    attributes.colorsymbols = &symbol;
    attributes.numsymbols = 1;
    attributes.valuemask = XpmColorSymbols;

    /* create a green bullet pixmap with these attributes */
```

```
    status = XpmReadFileToPixmap(XtDisplay(top),
                                 XRootWindowOfScreen(XtScreen(top)),
                                 "bullet_symb.xpm", &armpixmap, NULL,
                                 &attributes);
    if (status != XpmSuccess) {
        fprintf(stderr, "XpmError: %s\n", XpmGetErrorString(status));
        exit(1);
    }
    /* then put the two pixmaps onto the button */
    XtVaSetValues(button,
                  XmNlabelType, XmPIXMAP,
                  XmNlabelPixmap, pixmap,
                  XmNarmPixmap, armpixmap,
                  NULL);
    XtManageChild(button);
    XtRealizeWidget(top);
    XtAppMainLoop(app);
}
```

The first step in overriding a color in an XPM icon is to set the symbolic color to an actual color using an `XpmColorSymbol` structure. This structure is defined as follows:

```
typedef struct {
    char *name;      /* Symbolic color name */
    char *value;     /* Color value */
    Pixel pixel;     /* Color pixel */
} XpmColorSymbol;
```

The `pixel` field, which is not used in the previous example, allows you to specify the color directly with its pixel value instead of its name, if you already have it.

In our case, we set Foreground to green. We then use this structure to set the `colorsymbols` field of the `XpmAttributes` structure and set the `valuemask` accordingly. The `XpmAttributes` structure is passed to `XpmReadFileToPixmap()`, which uses the structure in creating the pixmap.

USING TRANSPARENCY

There are several cases where you need transparency. One is when you simply want a bullet icon with a transparent background, so that you get an icon similar to what is shown in Figure 9.:

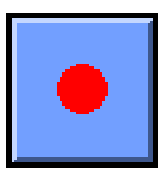

FIGURE 9: *THE BULLET ICON WITH A TRANSPARENT BACKGROUND ON A PUSHBUTTON*

One technique for achieving this effect is to use the XPM color overriding mechanism to create a pixmap with the transparent color set to the color of the underlying widget. The following code fragment demonstrates this technique using the *bullet_transp.xpm* file defined earlier:

```
    Pixel color;
    ...
    button = XmCreatePushButton(top, "button", NULL, 0);

    /* get the background color of the button */
    XtVaGetValues(button, XmNbackground, &color, NULL);

    /* set the transparent color to the background */
    symbol.name = NULL;
    symbol.value = "none";
    symbol.pixel = color;

    attributes.colorsymbols = symbols;
    attributes.numsymbols = 1;
    attributes.valuemask = XpmColorSymbols;
    status = XpmReadFileToPixmap(XtDisplay(top),
                                 XRootWindowOfScreen(XtScreen(top)),
                                 "bullet_transp.xpm", &pixmap, NULL,
                                 &attributes);
    if (status != XpmSuccess) {
        fprintf(stderr, "XpmError: %s\n", XpmGetErrorString(status));
        exit(1);

    /* put the pixmap on the button */
    XtVaSetValues(button,
                  XmNlabelType, XmPIXMAP,
                  XmNlabelPixmap, pixmap,
                  NULL);
    }
```

This is a modified version of the first program, where the red bullet pixmap is created with the color None set to the background color of the PushButton widget. When the pixmap is displayed on the widget, the bullet does not appear to have a rectangular border.

Another case where you need transparency is when you want to use a nonrectangular XPM icon to define the shape of a window. As we stated earlier, the X Window System does not actually support transparency, but most X servers provide the Shape Extension, which you can use to achieve this effect.† The following program illustrates how you can use XPM and this extension to create a Motif PushButton widget that is shaped like the bullet icon.

```
    #include <stdio.h>
    #include <Xm/PushB.h>
    #include <Xm/BulletinB.h>
    #include <X11/xpm.h>
    #include <X11/extensions/shape.h>

    main(int argc, char *argv[])

    {
        XtAppContext app;
        Widget top, bb, button;
        int status;
        Pixmap pixmap, mask;
        XpmAttributes attributes;
```

† See The X11 Nonrectangular Window Shape Extension documentation for details about this extension.

```
    XpmColorSymbol symbol;

    top = XtAppInitialize(&app, "BulletButton", NULL, 0, &argc, argv,
                          NULL, NULL, 0);
    bb = XmCreateBulletinBoard(top, "bb", NULL, 0);
    button = XmCreatePushButton(bb, "button", NULL, 0);

    /* create a red bullet pixmap from the XPM file */
    status = XpmReadFileToPixmap(XtDisplay(top),
                                 XRootWindowOfScreen(XtScreen(top)),
                                 "bullet_transp.xpm", &pixmap, &mask,
                                 NULL);
    if (status != XpmSuccess) {
        fprintf(stderr, "XpmError: %s\n", XpmGetErrorString(status));
        exit(1);
    }

    /* put the pixmap onto the button */
    XtVaSetValues(button,
                  XmNlabelType, XmPIXMAP,
                  XmNlabelPixmap, pixmap,
                  NULL);
    XtManageChild(button);
    XtManageChild(bb);
    XtRealizeWidget(top);
    /* finally reshape the button window with the mask */
    XShapeCombineMask(XtDisplay(button), XtWindow(button), ShapeBounding,
                      6, 6, mask, ShapeSet);

    XtAppMainLoop(app);
}
```

In this example, we pass a reference to another pixmap variable, `mask`, to the `XpmReadFileToPixmap()` function. Since the XPM file uses the color None, this variable is set to an additional 1-bit deep pixmap that can then be used as a shape mask for the `XShapeCombineMask()` function.

This functionality allows you store both the pixmap and its mask in an XPM file, rather than having to create two files as you would with XBM. Another situation where this feature is useful occurs when you define a cursor. Defining a cursor with XPM saves you from having to deal with two files, since both the cursor and its shape can be stored in a single XPM file.

Optimizing with the Low-level Routines

In this final example, we examine how you can optimize your program using the low-level XPM functions. If you look back at the example where we created both red and green bullet icons, it's clear that some work has been duplicated: the same XPM file is read once to create the red bullet icon and a second time to create the green one. You can avoid this unnecessary time-consuming operation by using an `XpmImage` structure, which is defined as follows:

```
typedef struct {
    unsigned int width;     /* image width */
    unsigned int height;    /* image height */
    unsigned int cpp;       /* number of characters per pixel */
```

```
    unsigned int ncolors;  /* number of colors */
    XpmColor *colorTable;  /* list of related colors */
    unsigned int *data;    /* image data */
} XpmImage;
```

With the `XpmColor` structure defined like this:

```
typedef struct {
    char *string;       /* characters string */
    char *symbolic;     /* symbolic name */
    char *m_color;      /* monochrome default */
    char *g4_color;     /* 4 level grayscale default */
    char *g_color;      /* other level grayscale default */
    char *c_color;      /* color default */
} XpmColor;
```

These structures allow you to read an XPM file and store the image definition in memory with all of the information related to the colors it contains. Once you have an XpmImage structure that contains the image definition, you can use it to create as many pixmaps as you want without having to read the XPM file multiple times. The following program demonstrates this technique by creating the red and green bullet icons reading the file *bullet_transp.xpm* only once.

```
#include <stdio.h>
#include <Xm/PushB.h>
#include <X11/xpm.h>

main(int argc, char *argv[])
{
    XtAppContext app;
    Widget top, button;
    int status;
    Pixmap pixmap, armPixmap;
    XpmAttributes attributes;
    XpmColorSymbol symbols[2];
    Pixel color;
    XpmImage xpmimage;

    top = XtAppInitialize(&app, "BulletButton", NULL, 0, &argc, argv,
                          NULL, NULL, 0);
    button = XmCreatePushButton(top, "button", NULL, 0);

    /* read the XPM file */
    status = XpmReadFileToXpmImage("bullet_transp.xpm", &xpmimage, NULL);
    if (status != XpmSuccess) {
        fprintf(stderr, "XpmError: %s\n", XpmGetErrorString(status));
        exit(1);
    }

    /* create a red bullet pixmap from the XpmImage structure */
    XtVaGetValues(button, XmNbackground, &color, NULL);

    symbols[0].name = NULL;
    symbols[0].value = "none";
    symbols[0].pixel = color;

    attributes.colorsymbols = symbols;
```

```
        attributes.numsymbols = 1;
        attributes.valuemask = XpmColorSymbols;
        status = XpmCreatePixmapFromXpmImage(XtDisplay(top),
                                    XRootWindowOfScreen(XtScreen(top)),
                                    &xpmimage, &pixmap, NULL,
                                    &attributes);
        if (status != XpmSuccess) {
            fprintf(stderr, "XpmError: %s\n", XpmGetErrorString(status));
            exit(1);
        }

        /* then, create a green pixmap from the XpmImage structure */
        XtVaGetValues(button, XmNarmColor, &color, NULL);

        symbols[1].pixel = color;
        symbols[1].name = "Foreground";
        symbols[1].value = "green";
        attributes.numsymbols = 2;
        attributes.valuemask = XpmColorSymbols;

        status = XpmCreatePixmapFromXpmImage(XtDisplay(top),
                                    XRootWindowOfScreen(XtScreen(top)),
                                    &xpmimage, &armPixmap, NULL,
                                    &attributes);
        if (status != XpmSuccess) {
            fprintf(stderr, "XpmError: %s\n", XpmGetErrorString(status));
            exit(1);
        }
        /* now put the two pixmaps onto the button */
        XtVaSetValues(button, XmNlabelType, XmPIXMAP,
                      XmNlabelPixmap, pixmap,
                      XmNarmPixmap, armPixmap,
                      NULL);

        XtManageChild(button);
        XtRealizeWidget(top);
        XtAppMainLoop(app);
    }
```

This program reads the XPM file using `XpmReadFileToXpmImage()`, which stores all of the data for the image in the specified `XpmImage` structure. Then we create the two pixmaps from the `XpmImage` structure instead of from the file, which is much faster.

Another case where you need to use the `XpmImage` structure is when you are writing a pixmap editor dedicated to the XPM format, so you can directly deal with all the data the format can store. If you are writing an image converter, you will also want to use this structure so you get a free XPM parser.

Related Software

You can find many X applications that allow you to edit and manipulate color images and to save them as XPM files. However, the program called *pixmap* is the only icon editor that is freely available and fully dedicated to XPM. It provides you with access to all the original XPM capabilities such as symbolic color names. This program is written and supported by Lionel Mallet from SIMULOG, who can be reached electronically at *mallet@simp-*

olis.fr. You can get *pixmap* via anonymous FTP from *ftp.x.org* in */contrib/applications* and from *avahi.inria.fr* in */pub/pixmap*.

Many of these image editors also allow you to convert XPM files to and from other image formats. The best way to deal with image conversion is to use the *netpbm* package which provides you with a number of programs to convert images to and from a variety of different formats, as well as programs that allow a few basic image operations. This package is based on the original *pbmplus* package and is supported by a group of programmers who can be reached through the mailing list *netpbm@fysik4.kth.se*. The *netpbm* package is freely available via anonymous FTP from f*tp.x.org* in */R5contrib*.

Finally, if you are looking for a collection of XPM icons, we suggest you to look at *Anthony's X Icon Library*, which is also freely available. It can be retrieved via anonymous FTP from *ftp.cae.wisc.edu* in */hpux9/X11R5/Graphics*. This library contains a large collection of small icons (less than 100 by 100 pixels in size) that are stored in categories. While black and white icons are stored in XBM format, all the color icons are stored in XPM format.

CONCLUSION

XPM fills a gap in the X Window System and provides you with a powerful format and library to help you build more pleasant graphical user interfaces with attractive color icons. XPM has been designed to fit the special needs of designing and using color icons with respect to the X philosophy of providing the user with customizable resources.