XPM
The X PixMap Format

Arnaud Le Hors & Colas Nahaboo
lehors@sophia.inria.fr
colas@sophia.inria.fr

BULL Research FRANCE – Sophia Antipolis

November 1991

1 Introduction: Why another image format?

As many X programmers, when we began to try to include color images in our X programs, we were faced with a plethora of image formats to choose from. We thought that each of these formats had been designed to handle big (typically more than 600x400 pixels), colorful (with at least 8 bit colormaps) images. However, we soon found that we had different needs, and thus these image formats were somewhat inadequate. We thus designed the XPM (X PixMap) icon format, and refined it thanks to the international X community. We feel that it is mature enough now to be proposed to the X consortium to be included in the standard MIT tape. This paper describes this third version of the XPM format.

2 Why is there no standard image format?

At first glance, one may think that a single image format could encompass all the needs of computer graphics professionals. The problem is that in most cases, images must be handled very efficiently, and thus the image format chosen often closely reflected the underlying hardware, as is the case for most formats coming from the PC world (e.g. GIF, IFF's ILBM, Mac resources). In the academic world, there is nearly one format per research team. The situation has stabilized, the formats can be grouped into three classes:

1. the working format: the one that is most adapted to the application or graphics hardware and software. In our X world, this can be thought of as the XWD format (X Window Dump, which is derived from the memory structure XImage). Rasterfiles on Sun workstations and GIF files on PCs are example of formats reflecting the underlying hardware.

2. the storage format: one that is often chosen for its built-in picture compression facility. Gif (and perhaps Tiff in the future) seems to play this role nowadays.

3. the interchange format: the one chosen to ease the writing of conversion programs between this format and any other. One good example of this pivotal format is Jeff Posanker's **pmbplus** package.

3 The characteristics of icons

What we really need is an **icon** format rather than an **image** format. In a typical application one often want a lot a small images, which can be either elements of the decoration (for a 3D look), buttons to press on, or "road signs" to convey information. These images have different characteristics such as being small, numerous, stylized, and customizable.

Also an icon is composed of several objects associated with single colors. If you consider for instance a 3D button you may define five objects: front, top, bottom, left, and right faces (figure 1.1). We need to be able to specify the way the colors associated with the different objects composing the icon are changed depending on the type of visual it is displayed on. For instance, on a color visual we may have a 3D button with white left and top faces, a gray front face, and black bottom and right faces. Displayed with the same colors on a black & white visual it would be: white left and top faces, a white front face, and black bottom and right faces (figure 1.2). This is not satisfactory because we would rather have: a white front face, and black side faces (figure 1.3).

We also need to be able to override built-in colors in order to let the user change them. For instance if we define an application interface with 3D buttons, we need to let the user redefine the colors used to keep the 3D look if the application is customised to be within green or blue shades.

In addition we want to have transparent colors to be able to define non-rectangular icons. One application of this is to define cursors. Indeed so far cursors are two colors only and are defined with two bitmaps: the source specifying the foreground and the background colors, and the mask specifying the shape (these are stored in two different files). This could be handled by a single icon in which the outer pixels are defined as transparent. Furthermore having cursors defined with such icons would provide support for future multicolor cursors.

With cursors comes the need to have hotspot information associated with the icon, but this is not the only case for which such an information can be useful. In particular, we may want to make animation sequences by putting several icons, one after the other, at the same place. To do so we need to have a hotspot related to the icons to know how to place them. For example to animate a warning sign by growing and reducing it, one needs to have a hotspot to easily center the icons whatever their size.

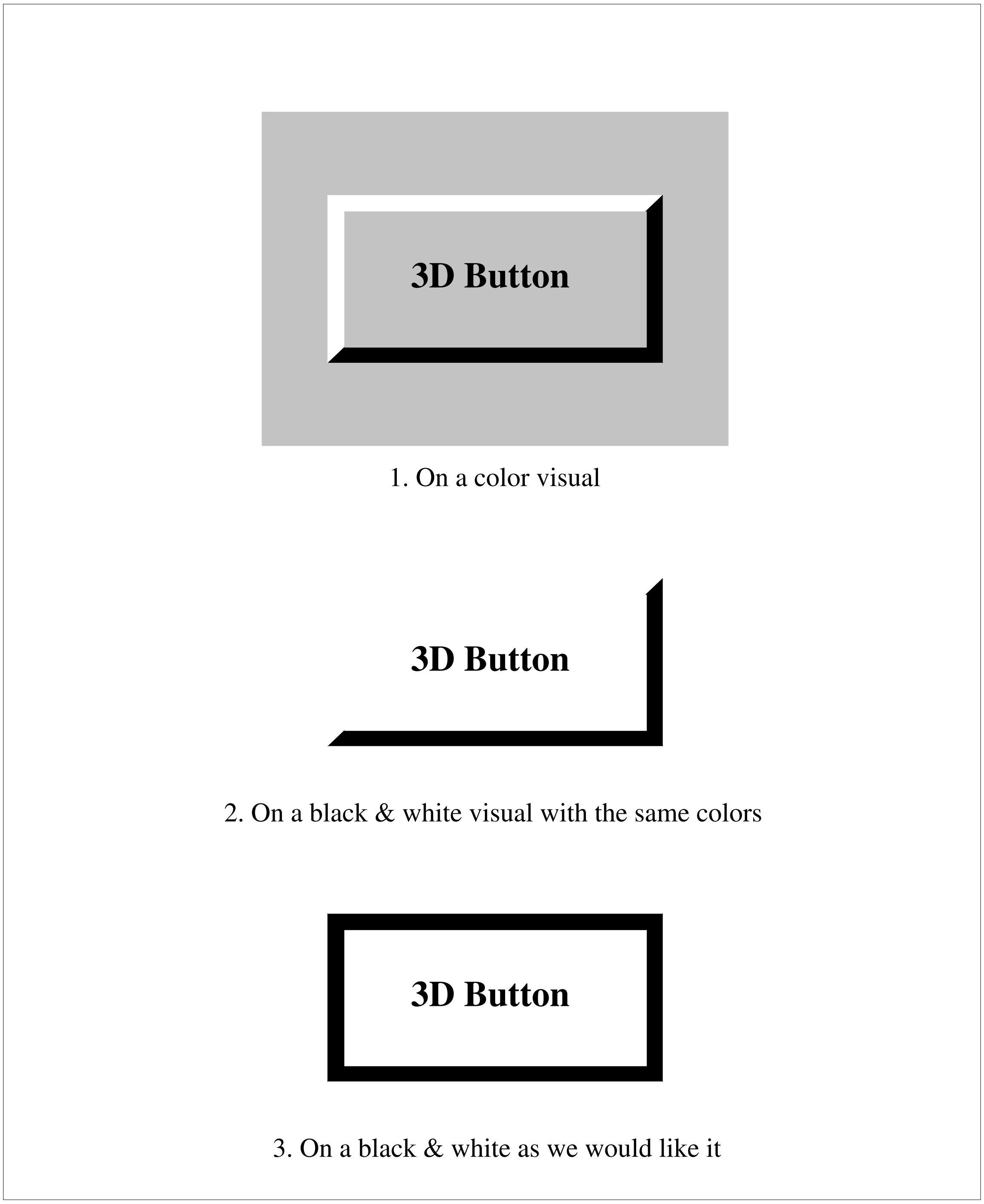


Figure 1: A 3D look button

4 The XPM format

The **XPM** format presents a C syntax, in order to provide the ability to include **XPM** files in C and C++ programs. It is in fact an array of strings composed of six different sections as follows:

```
<Header line>
<Declaration and Beginning of Assignment line>
<Values>
<Colors>
<Pixels>
<Extensions>
<End of Assignment>
```

The `<Header line>` is a comment containing the keyword XPM as follows:

/* XPM */

The words are separated by a white space which can be composed of space and tabulation characters.

The `<Declaration and Beginning of Assignment line>` must end by a newline character and is composed as follows:

static char* `<variable_name>`[] = {

The `<Values>` section is a string containing four or six integers in base 10 that correspond to: the pixmap width and height, the number of colors, the number of characters per pixel (so there is no limit on the number of colors), and, optionally the hotspot coordinates.

`<width> <height> <ncolors> <cpp> [<x_hotspot> <y_hotspot>]`

The `Colors` section contains as many strings as there are colors, and each string is as follows:

`<chars>` {`<key> <color>`}+

Where `<chars>` is the `<chars_per_pixel>` length string (not surrounded by anything) representing the pixels, `<color>` is the specified color, and `<key>` is a keyword describing in which context this color should be used. Currently the keys may have the following values:

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

Colors can be specified by giving the colorname, a # foolwed by the RGB code, or a % followed by the HSV code. The symbolic name provides the ability of specifying the colors at load time and not to hard-code them in the file. Also the string **None** can be given as a colorname to mean “transparent”. Transparency is handled by providing a masking bitmap in addition to the pixmap.

The `<Pixels>` section is composed by `<height>` strings of `<width>` * `<chars_per_pixel>` characters, where every `<chars_per_pixel>` length string must be one of the previously defined groups in the `<Colors>` section.

Then follows the `<Extensions>` section which is empty so far but may contained several strings in future.

Finally the `<End of Assignment>` section ends the format with a closing brace “}”.

Below is an example which is the XPM file of a plaid pixmap. This is a 22x22 pixmap, with 4 colors and 2 characters per pixel. The hotspot coordinates are (0, 0). There are symbols and default colors for color and monochrome visuals.

```
/* XPM */
static char * plaid[] = {
/* plaid pixmap
 * width height ncolors chars_per_pixel */
"22 22 4 2 0 0",
/* colors */
"   c red       m white  s light_color ",
"Y  c green     m black  s lines_in_mix ",
"+  c yellow    m white  s lines_in_dark ",
"x              m black  s dark_color ",
/* pixels */
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"Y Y Y Y Y x Y Y Y Y Y + x + x + x + x + x + ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"          x           x   x   x Y x   x   x ",
"          x             x   x   Y   x   x   ",
"          x           x   x   x Y x   x   x ",
"          x             x   x   Y   x   x   ",
"          x           x   x   x Y x   x   x ",
"x x x x x x x x x x x x x x x x x x x x x x ",
"          x           x   x   x Y x   x   x ",
```

```
"          x             x   x   Y   x   x   ",
"          x           x   x   x Y x   x   x ",
"          x             x   x   Y   x   x   ",
"          x           x   x   x Y x   x   x "
} ;
```

5 Advantages and limitations

The major advantage of the XPM format compared to the other image formats is, of course, that it handles the peculiar icon needs we have described earlier. Below is an XPM file defining a 3D button where the symbol top&left for the top and left faces has two default colors: white for color visual and black for monochrome, which gives us the desired effect (see figures 1.1 & 1.3).

```
/* XPM */
static char * 3D-button [] = {
"30 20 3 1",
"  s top\&left     c white m black",
"X s bottom\&right c black",
". s front         c gray",
"                              ",
"                             X",
"  ..........................XX",
"  ..........................XX",
"  ..........................XX",
"  ..........................XX",
"  ..........................XX",
"  ..........................XX",
"  ..........................XX",
"  ..........................XX",
"  ..........................XX",
"  ..........................XX",
"  ..........................XX",
"  ..........................XX",
"  ..........................XX",
"  ..........................XX",
"  ..........................XX",
"  ..........................XX",
" XXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"};
```

However, this format still has some limitations. There is, for example, no provision for resizing an icon although we can think of cases for which this would be useful. This happens typically when the icon is associated to another object, which can be resized. For instance if a sign is associated with a button, the sign should be resized in the same way as the button. The resize operation of course can be done by the application with some pixmap scaling routine but, in fact, as colors are specified for the different visuals, the way we want an icon to be resized may be not obvious. Indeed, instead of just scaling it, which can give poor results, one wants to modify it in order to keep it as clear as possible. For example one may add colors to have better anti-aliasing. In the same way big fonts are not obtained by scaling small fonts (figure 2). But we think that it is a problem which can only be addressed by higher level format such as PostScript or TeX's Metafont system.

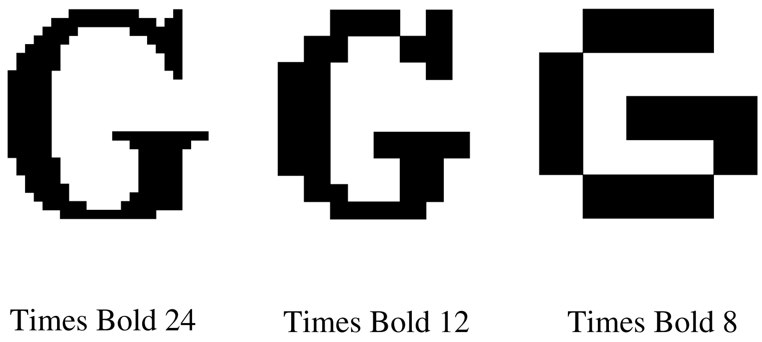


Figure 2: Big fonts are not obtained by scaling small fonts

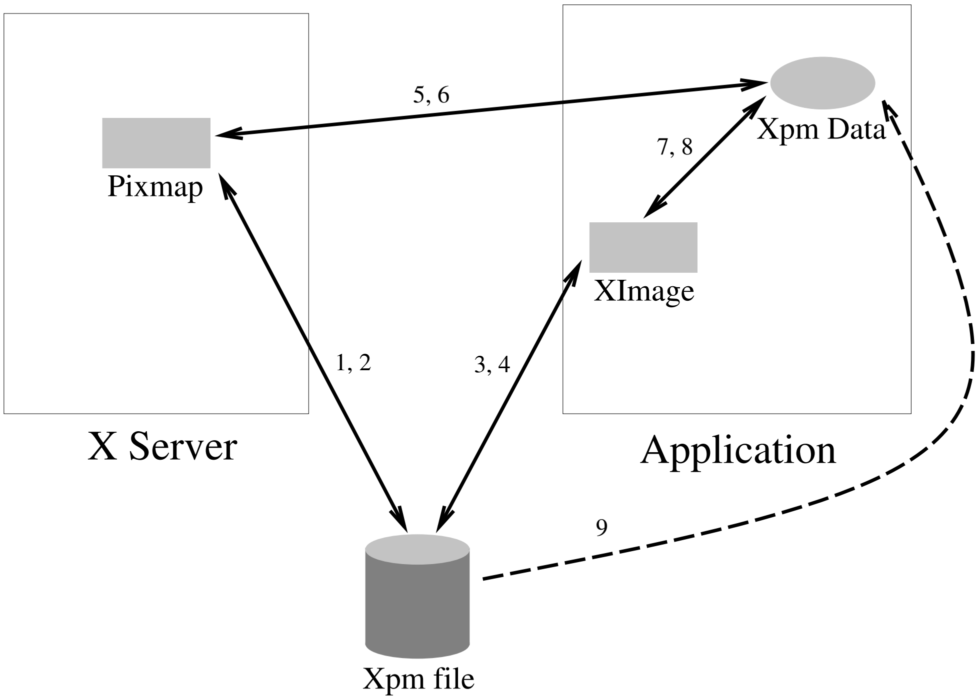
Also XPM does not provide anything to manage collections of icons, which one may need to deal with animations made from several icons. However we think that this issue can be addressed with some external mechanism such as the unix archive maintainer **ar** or with some structure such as the **XFont** structure which is already used to manage a collection of cursors bitmaps and that could be added as an extension to the basic format. Besides we also think compression must be handled outside the format. At the present time the XPM library supports compression on unix platforms by using the standard unix compress program as a filter. Furthermore we think that using external tools to provide such features clearly separates the different problems and allows us to change them if needed.

6 XPM and the Xlib

The XPM library provides a set of Xlib-level functions which allows us to deal with images, pixmaps, XPM file, and data (included XPM file) in many ways (figure 3).

All of them provide the same kind of interface, to give an example we describe below the XpmReadFileToPixmap function which creates a pixmap and possibly its shape mask from an XPM file as described below:

int XpmReadFileToPixmap(*display, d, filename,*
pixmap_return, shapemask_return, attributes)
Display **display;*
Drawable *d;*



1. XpmReadFileToPixmap
2. XpmWriteFileFromPixmap
3. XpmReadFileToImage
4. XpmWriteFileFromImage
5. XpmCreatePixmapFromData
6. XpmCreateDataFromPixmap
7. XpmCreateImageFromData
8. XpmCreateDataFromImage
9. include file

Figure 3: The Xpm library functions

```
    char *filename;
    Pixmap *pixmap_return;
    Pixmap *shapemask_return;
    XpmAttributes *attributes;
```

display	Specifies the connection to the X server.
d	Specifies which screen the pixmap is created on.
filename	Specifies the file name to use.
pixmap_return	Returns the pixmap which is created.
shapemask_return	Returns the shapemask which is created, if any.
attributes	Specifies the location of an XpmAttributes structure to get and store information.

To use this function in its simplest way the caller can ignore the mask pixmap and the attributes structure by giving NULL pointers, he will get a pixmap as described in the specified file and built with default values for attributes such as the colormap, the visual, etc. Otherwise these values can be explicitly specified by using the XpmAttributes, which contains a valuemask in order to let the caller specify which attributes are set. It is also through this structure that the colors can be overridden, the hotspot retrieved, as well as many other things.

The XpmAttributes structure is defined as follows:

```
typedef struct {
    unsigned long valuemask;          /* Specifies which attributes 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;               /* Returns the width of the created pixmap */
    unsigned int height;              /* Returns the height of the created pixmap */
    unsigned int x_hotspot;           /* Returns the x hotspot's coordinate */
    unsigned int y_hotspot;           /* Returns 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 */

    /* Infos */
    int ncolors;                      /* Number of colors */
    char ***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's color table index */
} XpmAttributes;
```

7 XPM and the Toolkits

In order to address the problem of customizing applications using icons, we are in the processing of defining and providing toolkits with a string to pixmap converter. This allows one to specify via the X resources the XPM file to use and the colors to override.

8 History and perspectives

In April 1989, Colas Nahaboo and Daniel Dardailler, from the Bull Koala Team, designed the first version of the XPM format based on the X BitMap format, Daniel also developed a first library providing functions similar to the ones provided by the X library to deal with bitmaps. It was first distributed in June 1989 via public ftp and as part of GWM (Generic Window Manager), developed by Colas, which has been part of the X11 contrib since R4. In October 1990, Arnaud Le Hors and Colas designed the second version of XPM. It handled multiple syntax in order to be includable in languages other than C, such as Lisp, and the major improvement of symbolic colors with multi-defaults for different visual types. A new library supporting the new features was developed by Arnaud and distributed via public ftp. Several releases of the library succeeded one another. A "Birds of a feather" was chaired at the X conference in January 1991 where XPM appeared to satisfy most of people who use it in several applications. After many discussions on a mailing list in April 1991 the third version is proposed where the multiple syntax feature, which adds complexity and appears not to be actually used, is removed. This new version also provides two new features: hotspot information (optional coordinates) and transparent color. A new library providing a full set of functions with a simple and homogeneous interface was then developed and given as an R5 contrib at the beginning of November 1991. In addition Lionel Mallet from Simulog has developed, and given as an R5 contrib too, an XPM editor which is based on the bitmap editor developed by Davor Matic from Mit and which supports most features of the format.

As XPM seems to become a de-facto standard within the X community, we are planning to give it to the MIT for inclusion in the MIT X tape (Xmu library), and to submit it as an X Consortium standard.

9 Conclusion

Among all the existing image formats none addresses the peculiar needs of icons, we think XPM fills in this gap in the X library software. The number of applications [1] using it proves this.

[1] such as ICS Xcessory, IXI X.desktop, most window managers, etc.

References

[1] MIT X Consortium Standard *Xlib - C Language Interface.* Part of the documentation provided with X Window System Version 11 Release 5, 1991.

[2] Bull Koala Team, Arnaud Le Hors *XPM Manual - The X PixMap Format* Documentation provided with XPM Version 3.0 November 1991.