Manipulation de Palettes de Couleurs sous X/MOTIF

Gauthier LEBRUN

10 juin 1997

Résumé

Lorsqu'une application a besoin de couleurs pour pouvoir fonctionner, elle crée ses propres couleurs et les alloue dans la palette de couleurs par défaut.
Or, il se peut que cette palette soit pleine. Par exemple, lorsqu'une autre application tourne en même temps et qui a déjà allouée la palette avec ses couleurs, il ne reste plus de place à notre application pour pouvoir allouer ses couleurs.
La solution est donc de pouvoir créer une palette de couleurs propre à l'application et invisible aux autres. Aprés un bref descriptif des capacités de Xlib à gérer les palettes de couleurs, un exemple sera présenté.

1 Les palettes de couleurs sous Xlib

Une palette de couleurs physique est un registre dans lequel l'écran lit les intensités RVB[1] qui vont générer les couleurs à lécran. La pluspart des stations n'ont qu'une palette de couleurs physique. Dans ce cas, toutes les fenêtres de l'écran sont interprétées en utilisant la même palette de couleurs. Certaines stations ont plusieurs palettes physiques, dans ce cas plusieurs fenêtres peuvent avoir leur palette indépendente.

Si la palette de couleurs physique ne peut être modifiée, elle est dite *immuable*. Les systèmes monochromes ont une palette de couleurs immuable. Dans ce cas, aucune application cliente ne peut allouer de couleurs privées et toutes les valeurs RVB sont prédéfinies. Dans ces systèmes, `XCreateColormap` fonctionnera, mais il donnera seulement une autre copie de la palette par défaut.

Dans la pluspart des stations de travail, on peut écrire de nouvelles valeurs dans la palette de couleurs physique. Ces palettes sont dites *modifiables*.

1. RVB = Rouge, Vert, Bleu

X gère de multiples palettes de couleurs en gardant des *palettes de couleurs virtuelles* en mémoire et en les installant sur instruction du gestionnaire de fenêtres (*window manager*). L'*installation* d'une palette de couleurs est l'opération effectuant le passage d'une palette virtuelle dans la palette physique. Seules les palettes installées sont utilisées pour déterminer les couleurs apparaissant à l'écran. Quand il n'y a qu'une seule palette physique et qu'une palette virtuelle est installée, la palette virtuelle qui avait été installée précédemment devient *désinstallée*.

Sur la pluspart des stations de travail couleurs avec quatre ou huit plans, il est assez fréquent que le client alloue toutes les couleurs de la palette. Les palettes virtuelles sont une solution à ce problême. Quand un client ne peut pas allouer les couleurs dont il a besoin dans la palette installée, il peut créer une nouvelle palette virtuelle. Le *window manager* installera alors cette palette virtuelle lorsque l'application est utilisée.

Quand une palette de couleurs virtuelle est installée et qu'il n'y a qu'une seule palette de couleurs physique, tous les clients qui utilisaient l'ancienne palette seront affichée en fausses couleurs puisque les valeurs des pixels de leurs fenêtres seront interprétées d'après la nouvelle palette.

Quand une application crée une palette virtuelle, elle doit modifier l'attribut `colormap window` de sa fenêtre principale (`topLevel`) pour que le *window manager* puisse trouver quelle palette il doit installer. Par défaut cet attribut indique la palette par défaut. Si ses sous-fenêtres utilisent différentes palettes de couleurs, une propriété peut être modifiée pour l'indiquer au *window manager*.

2 Fonctions pour la manipulation de palettes de couleurs

Les fonctions suivantes peuvent être utilisées par les applications ayant besoin de gérer leurs propres palettes de couleurs : `XCreateColormap()`, `XFreeColormap()`, `XListInstalledColormaps()`, `XCopyColormapAndFree()`, `XSetWindowColormap()`.

Les fonctions `XInstallColormap()` et `XUninstallColormap()` sont utilisées par le gestionnaire de fenêtres pour installer et désinstaller les palettes de couleurs.

2.1 XCreateColormap()

2.1.1 Description

Cette fonction crée une palette de couleurs virtuelle, soit avec aucune entrée alouée, soit avec toutes ses entrées allouées en lecture/écriture, qui correspondent au *visual* passée en paramètre. Si aucune entrée n'est allouée, elle peuvent être soit allouées en lecture/écriture, soit en lecture seule. Si toutes les entrées sont allouées en lecture/écriture, la palette est totalement privée et a juste besoin que ses couleurs soient stockées avec `XStorecolors()`.

2.1.2 Arguments

```
Colormap XCreateColormap(display, w, visual, alloc)
        Display *display;
        Window w;
        Visual *visual;
        int alloc;
```

display : Spécifie la connection au Serveur X.

w : Spécifie la fenêtre de l'écran sur laquelle on veut créer la palette.

visual : Spécifie un type de support visuel supporté par l'écran. Si le type de visuel n'est pas l'un de ceux supporté par l'écran, alors la fonction renvoie une erreur *BadMatch*.

alloc : Spécifie les entrées de la palette à allouer. Ce paramètre peut être *AllocNone* ou *AllocAll*.

2.2 XFreeColormap()

2.2.1 Description

Cette fonction désinstalle la palette virtuelle et libère les resources associées à la palette. Elle envoie un événement `ColormapNotify` à toutes les fenêtres qui utilisaient la palette de couleurs.

2.2.2 Arguments

```
XFreeColormap(display, colormap)
        Display *display;
        Colormap colormap
```

display : Spécifie la connection au Serveur X.

colormap : Spécifie la palette à libérer.

2.3 XListInstalledColormaps()

2.3.1 Description

Cette fonction liste les palettes actuellement installées.

2.3.2 Arguments

```
Colormap *XListInstalledColormaps(display, w, num_return)
        Display *display;
        Window w;
        int *num_return;
```

display : Spécifie la connection au Serveur X.

w : Spécifie la fenêtre qui détermine l'écran.

num_return : Spécifie le nombre de palettes actuellement installées.

2.4 XCopyColormapAndFree()

2.4.1 Description

Cette fonction copie toutes les entrées de la palette existante dans une nouvelle palette et libère ces entrées de l'ancienne palette. Elle est utilisée quand l'allocation de couleurs échoue et que certaines couleurs ont déjà été allouées. Pour les applications ayant un besoin important de couleurs qu'elles ne peuvent assouvir, elles peuvent appeler `XCopyColorMapAndFree`, modifier l'attribut *colormap window*, et continuer d'allouer les couleurs dans la nouvelle palette.

La fonction `XCopyColorMapAndFree` crée une palette du même type de *visual* et pour le même écran que celui de la palette passée en paramètre. Elle retourne l'identificateur de la nouvelle palette.

2.4.2 Arguments

```
Colormap XCopyColormapAndFree(display, colormap)
        Display *display;
        Colormap colormap
```

display : Spécifie la connection au Serveur X.

colormap : Spécifie la palette à copier.

2.5 XSetWindowColormap()

2.5.1 Description

Cette fonction modifie l'attribut *colormap window* d'une fenêtre. La palette doit avoir le même de type de *visual* que la fenêtre sinon une erreur *BadMatch* est renvoyée.

2.5.2 Arguments

```
XSetWindowColormap(display, w, colormap)
        Display *display;
        Window w;
        Colormap colormap;
```

display : Spécifie la connection au Serveur X.

w : Spécifie la fenêtre.

colormap : Spécifie la palette à copier.

2.6 XInstallColormap()

2.6.1 Description

Cette fonction installe une palette de couleurs. Chaque fenêtre utiliant cet identificateur de palette dans son attribut *colormap* reçoit un événement *ColormapNotify*.

2.6.2 Arguments

```
XInstallColormap(display, colormap)
        Display *display;
        Colormap colormap;
```

display : Spécifie la connection au Serveur X.

colormap : Spécifie la palette à installer.

2.7 XUninstallColormap()

2.7.1 Description

Cette fonction supprime la palette virutelle de l'ensemble des palettes installées. Sur les systêmes à une seule palette physique, la palette par défaut est réinstallée. Elle envoie un événement *ColormapNotify* aux fenêtres qui utilisaient la palette spécifiée.

2.7.2 Arguments

```
XUninstallColormap(display, colormap)
        Display *display;
        Colormap colormap;
```

display : Spécifie la connection au Serveur X.

colormap : Spécifie la palette à désinstaller.

3 Application dans Xspect

La gestion des couleurs dans Xspect se trouve essentiellement dans le module `color.c`. On y définit les couleurs à utiliser dans le programme, et si nécessaire la palette privée de Xspect.
Une des premières tâches du programme principal (fonction `main` de `xspect.c`) est d'appeler la fonction `initColors(...)`. Cette fonction essaie d'allouer les couleurs dans la palette de couleurs par défaut. Si cette allocation échoue alors on crée une nouvelle palette virtuelle propre à Xspect grâce à la fonction `XCopyColormapAndFree()`.

Si une palette virtuelle est créée, l'utilisateur a alors la possibilité de l'activé ou de la dśactivée dans le menu Context|Option, par le champ privateColormap.

L'allocation des couleurs s'effectue par l'appel à la fonction `getcolors()`. Celle-ci crée une table des couleurs et essaie de l'allouer par un appel à `XAllocColorCells()`.

3.1 Remarques sur XAllocColorCells

Outre l'allocation de couleurs dans une palette, cette fonction nous permet de superposer des couches graphiques sans les détruire. Cette méthode permet d'améliorer les performances du programme en réduisant le nombre de graphiques complexes à réafficher. Elle est donc particulièrement utile lors des sélections.

L'astuce permettant de dessiner sans détruire ce qui a déjà été dessiné repose sur le fait que l'on peut dessiner sur un plan de la zone graphique. Cela est possible en utilisant le composant **plane_mask** du contexte graphique. Ce sont ces autres plans qui contiennent les informations de ce qui a déjà été dessiné. L'inconvénient de cette méthode est qu'elle nous oblige à allouer plus de couleurs que nous aurions normalement besoin.

3.1.1 Arguments

```
XAllocColorCells(display, colormap, contig, plane_masks_return,
                 nplanes, pixels_return,npixels)
        Display *display;
        Colormap colormap;
        Bool contig;
        unsigned long plane_masks_return[];
        unsigned int nplanes;
        unsigned long pixels_return[];
        unsigned int npixels;
```

display : Spécifie la connection au Serveur X.

colormap : Spécifie la palette à désinstaller.

contig : Valeur booléenne indiquant que les plans soint contiguës ou non..

plane_masks_return : Renvoie un tableau des masques des plans.

nplanes : Indique le nombre de plans dans le tableau précédent.

pixels_return : Renvoie un tableau de pixels.

npixels : Indique le nombre de pixels.

4 Code source du programme d'essai

Ce programme reprend en grande partie la méthode utilisé dans Xspect. Cette-à-dire qu'il essaie d'allouer les couleurs dans la palette courante, et en cas d'échec il crée une palette virtuelle. Pour tester, les résultats, nous affichons une mire des couleurs.

```
/* INCLUDES */
/* -------- */

/* standard */
#include <stdio.h>
#include <stdlib.h>

/* XLib */
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>

/* MOTIF */
#include <Xm/FileSB.h>
#include <Xm/DrawingA.h>
#include <Xm/PushBG.h>
#include <Xm/RowColumn.h>

/* DEFINES */
```

```
/* ------- */
#define charset XmSTRING_DEFAULT_CHARSET
#define MAX_PLANE           2
#define MAX_CELL            ((MAX_COLOR) * (1 << (MAX_PLANE)))
#define MAX_COLOR           16
#define CANNOT_OVERLAY      0
#define CAN_OVERLAY         1
#define MAX_CHAR            256
#define UNDEFINED           -1
#define WIDTH     400
#define HEIGHT     300

/* TYPES */
/* ----- */
typedef char STRING[MAX_CHAR];

/* Constantes de couleurs */
/* ---------------------- */
static short valStd8[][3] =
{
  0xFF, 0xFF, 0xFF,
  0x00, 0xFF, 0xFF,
  0xFF, 0x00, 0xFF,
  0x00, 0x00, 0xFF,
  0xFF, 0xFF, 0x00,
  0x00, 0xFF, 0x00,
  0xFF, 0x00, 0x00,
  0x00, 0x00, 0x00
};


/* Variables globales */
/* ------------------ */
Display *display;
int screen_num;
Window window;
unsigned long planeMasks[MAX_PLANE];
unsigned long pixels[MAX_COLOR];
XColor exactDefs[MAX_CELL];

Colormap colormap, mycolormap;
int iwhite = UNDEFINED;
int iblack = UNDEFINED;
int ifg = UNDEFINED;
int ibg = UNDEFINED;
Pixmap pixmap;



void blkToZero(str)
  char *str;
{
  int i;
  char *ptr = str;

  for (i = 0; i < strlen(str); i++, ptr++)
    {
      if (*ptr == ' ')
*ptr = '0';
    }
}
```

```
void fin()
{
  /* Restore the old colormap */
  if (mycolormap!=colormap)
    {
      printf("Restore the old colormap\n");
      XFreeColormap(display,mycolormap);
    }

  exit();
}


/* Programme Principal */
/* ------------------- */
main(argc,argv)
int argc;
char *argv[];
{
  Widget topLevel,text_w, dialog,drawing_a, pb;
  XtAppContext app;
  XGCValues gcv;
  GC gc;

  extern void exit(), echo_file();
  void DessinePalette();
  void initColors();
  void drawing_area_callback();

  topLevel=XtVaAppInitialize(&app, "Demos", NULL, 0,
     &argc, argv, NULL,
     XmNwidth, WIDTH,
     XmNheight, HEIGHT,
     NULL);

  display = XtDisplay(topLevel);
  screen_num = DefaultScreen(display);
  window=XtWindow(topLevel);

  initColors(8);
  XtVaSetValues(topLevel,XmNcolormap,mycolormap,NULL);

  /* Create a drawing area widget */
  drawing_a=XtVaCreateWidget("drawing_a",
    xmDrawingAreaWidgetClass, topLevel,
    NULL);
  XtAddCallback(drawing_a,XmNexposeCallback,drawing_area_callback,NULL);

  gc = XCreateGC(XtDisplay(drawing_a),
  RootWindowOfScreen(XtScreen(drawing_a)), 0, NULL);
  XtVaSetValues(drawing_a,XmNuserData,gc,NULL);

  /* Create a pixmap the same size as the drawing area */
  pixmap=XCreatePixmap(XtDisplay(drawing_a),
       RootWindowOfScreen(XtScreen(drawing_a)),WIDTH,HEIGHT,
       DefaultDepthOfScreen(XtScreen(drawing_a)));


  /* Clear pixmap with white */
  XSetForeground(XtDisplay(drawing_a), gc, WhitePixelOfScreen(XtScreen(drawing_a)));
  XFillRectangle(XtDisplay(drawing_a), pixmap,gc,0,0,WIDTH,HEIGHT);
```

```
  DessinePalette(drawing_a);


  /* Add a pushbutton the user can use to close */
  pb = XtVaCreateManagedWidget("Close",
       xmPushButtonGadgetClass, drawing_a,
       NULL);

  /* if activated call the callback */
  XtAddCallback(pb,XmNactivateCallback, fin,NULL);

  /* create a file selection dialog */
  dialog=XmCreateFileSelectionDialog(topLevel,"filesb",NULL,0);
  XtAddCallback(dialog,XmNcancelCallback,exit,NULL);
  XtAddCallback(dialog,XmNokCallback,echo_file,NULL);

  XtManageChild(dialog);

  /* main loop */
  XtManageChild(drawing_a);
  XtRealizeWidget(topLevel);
  XtAppMainLoop(app);

}


/* Fonction Callback appelee lors de l'activation de la fenetre */
/* ou de sa re-exposition */
/* ------------------------------------------------------------ */
void drawing_area_callback(widget,data,cbs)
Widget widget;
XtPointer data;
XmDrawingAreaCallbackStruct *cbs;
{
  XEvent *event=cbs->event;
  Display *dpy=event->xany.display;

  if (cbs->reason == XmCR_EXPOSE || cbs->reason == XmCR_ACTIVATE)
    {
      GC gc;
      XtVaGetValues(widget,XmNuserData,&gc,NULL);
      XCopyArea(dpy,pixmap, event->xany.window,gc,0,0,WIDTH,HEIGHT,0,0);
    }
}


/* Write the name of the selected file on the screen */
/* ------------------------------------------------- */
void echo_file(fs,client_data,cbs)
Widget fs; /* file selection box */
XtPointer client_data;
XmFileSelectionBoxCallbackStruct *cbs;
{
  char *filename;

  if (!XmStringGetLtoR(cbs->value,charset,&filename))
    return;

  if (!*filename)
    {
      printf("No file selected.");
```

```
      XtFree(filename);
      return;
    }

  printf("Filename given : %s \n",filename);
  XtFree(filename);
}


/* Draw the colormap in a window */
/* ----------------------------- */
void DessinePalette(widget)
Widget widget;
{
  int i;
  int step = (4 * WIDTH / 5) / MAX_COLOR;
  GC gc;

  XtVaGetValues(widget,XmNuserData,&gc,NULL);

  for (i=0;i<MAX_COLOR;i++)
    {
      XSetForeground(XtDisplay(widget), gc, exactDefs[i].pixel);
      XFillRectangle(XtDisplay(widget),pixmap,gc,i*step,0,step,HEIGHT);
    }
}


/* colormap initialisation */
/* ----------------------- */
void initColors(ncolor)
int ncolor;
{
  XVisualInfo visualInfo;
  int defaultDepth=DefaultDepth(display,screen_num);

  int nplane, ncell;

  colormap = DefaultColormap(display,screen_num);
  mycolormap=colormap;

  nplane = 2;
  ncell = ncolor * (1 << nplane);

  if (defaultDepth == 1)
    {
      /* must be StaticGray, use black and white */
      printf("The screen can only have 2 colors.");

      ncolor = 0;
      nplane = 2;
      ncell = 2;
    }

  if (!XMatchVisualInfo(display, screen_num, defaultDepth, PseudoColor, &visualInfo))
    {
      if (!XMatchVisualInfo(display, screen_num, defaultDepth, DirectColor, &visualInfo))
{
  /* No PseudoColor or TrueColor visual available at defaultDepth.
   * Some applications might try for a GrayScale visual
   * here if they can use gray to advantage, before
   * giving up and using black and white.
```

```
   */
  printf("initColors: TODO HERE: try for a GrayScale... \n");
}
    }

  if ( getColors(ncolor, nplane, ncell, &planeMasks[0], &pixels[0], &exactDefs[0]) == CANNOT_OVERLAY )
      printf("We can't allocate enough colors for overlay.\n");
}

int getColors(maxColors, maxPlanes, maxCells, planeMasks, pixels, exactDefs)
     int maxColors;
     int maxPlanes;
     int maxCells;
     unsigned long planeMasks[MAX_PLANE];
     unsigned long pixels[MAX_COLOR];
     XColor exactDefs[MAX_CELL];
{
  STRING name[MAX_CELL];
  XColor *pExDef;
  Colormap defaultCmap = colormap;
  int nbCells = maxCells;
  int power2maxPlanes;
  int i, j, k;
  int class;

  void createPrivateMap();

  /* compute our private color map */
  createPrivateMap(&valStd8[0][0], name, maxColors, maxPlanes, maxCells, planeMasks, pixels, exactDefs);
  iwhite = 0;
  ibg = 1; /* cyan */
  iblack = 7;
  ifg = 3; /* blue */

  if (XAllocColorCells(display, defaultCmap, False, planeMasks, maxPlanes, pixels, maxColors) == 0)
    {
      fprintf(stderr,"Can't alloc enough colors in the current color map. Creation of a new one. \n");

      if ((defaultCmap=XCopyColormapAndFree(display,colormap)) == BadAlloc)
fprintf(stderr, "\nCan't Create new colormap\n");

      if (XAllocColorCells(display, defaultCmap, False, planeMasks, maxPlanes, pixels, maxColors) == 0)
{
  fprintf(stderr,"\nCan't allocate %d colors after created the new color map\n\
                        This is a big problem !\n",maxColors);
  exit(0);
}
    }

  /* allocated colorcells succesfully,
   * now set their colors
   */

  for (i = 0; i < maxCells; i++)
    {
      exactDefs[i].flags = DoRed | DoGreen | DoBlue;
      if (! XParseColor(display, defaultCmap, name[i],  &exactDefs[i]))
  fprintf(stderr,"basic: color name %s not in database", name[i]);

      printf("%s\n",name[i]);
    }
```

```
  /* set pixel value in struct to the allocated ones */
  for (i = 0; i < (1 << maxPlanes); i++)
    for (j = 0; j < maxColors; j++)
      {
unsigned long msk = 0;

for (k = 0; k < maxPlanes; k++)
  {
    int bool = ((1 << k) & i) >> k;
    msk |= bool * planeMasks[k];
  }

exactDefs[i * maxColors + j].pixel = pixels[j] | msk;
      }

  /* this sets the color of the read/write cells */
  XStoreColors(display, defaultCmap, exactDefs, nbCells);

  if (defaultCmap!=colormap)
    {
       mycolormap=defaultCmap;
       XInstallColormap(display,mycolormap);
    }

  if (nbCells >= 4)
       return (CAN_OVERLAY);
   else
       return (CANNOT_OVERLAY);
}


void createPrivateMap(val, name, maxColors, maxPlanes, maxCells, planeMasks, pixels, exactDefs)
     short val[][3];
     STRING name[MAX_CELL];
     int maxColors;
     int maxPlanes;
     int maxCells;
     unsigned long planeMasks[MAX_PLANE];
     unsigned long pixels[MAX_COLOR];
     XColor exactDefs[MAX_CELL];
{
  int i, j;
  int fr1, fg1, fb1;
  int fr2, fg2, fb2;
  int fr3, fg3, fb3;
  double coeff = 1.5;

  printf("createPrivateMap..\n");

  /* plan 1 */
  fr1 = 0;
  fg1 = 255;
  fb1 = 0;

  /* plan 2 */
  fr2 = 255;
  fg2 = 0;
  fb2 = 0;

  /* plan 1 & 2 */
  fr3 = 255;
  fg3 = 255;
```

```
  fb3 = 0;

  i = 0;
  for (j = 0; j < maxColors; j++)
      {
sprintf(name[i * maxColors + j], "#%2X%2X%2X\0", val[j][0],  val[j][1],  val[j][2]);
blkToZero(name[i * maxColors + j]);
      }

  i = 1;
  for (j = 0; j < maxColors; j++)
      {
double cr, cg, cb;

cr = (coeff * val[j][0] + fr1) / (coeff + 1.);
cg = (coeff * val[j][1] + fg1) / (coeff + 1.);
cb = (coeff * val[j][2] + fb1) / (coeff + 1.);

if (cr > 255.) cr = 255.;
if (cg > 255.) cg = 255.;
if (cb > 255.) cb = 255.;
sprintf(name[i * maxColors + j], "#%2X%2X%2X\0", (int) cr, (int) cg, (int) cb);
blkToZero(name[i * maxColors + j]);
      }

  i = 2;
  for (j = 0; j < maxColors; j++)
      {
double cr, cg, cb;

cr = (coeff * val[j][0] + fr2) / (coeff + 1.);
cg = (coeff * val[j][1] + fg2) / (coeff + 1.);
cb = (coeff * val[j][2] + fb2) / (coeff + 1.);

if (cr > 255.) cr = 255.;
if (cg > 255.) cg = 255.;
if (cb > 255.) cb = 255.;
sprintf(name[i * maxColors + j], "#%2X%2X%2X\0", (int) cr, (int) cg, (int) cb);
blkToZero(name[i * maxColors + j]);
      }

  i = 3;
  for (j = 0; j < maxColors; j++)
      {
double cr, cg, cb;

cr = (coeff * val[j][0] + fr3) / (coeff + 1.);
cg = (coeff * val[j][1] + fg3) / (coeff + 1.);
cb = (coeff * val[j][2] + fb3) / (coeff + 1.);

if (cr > 255.) cr = 255.;
if (cg > 255.) cg = 255.;
if (cb > 255.) cb = 255.;
sprintf(name[i * maxColors + j], "#%2X%2X%2X\0", (int) cr, (int) cg, (int) cb);
blkToZero(name[i * maxColors + j]);

      }
}
```