

X-I

Xlib : les couleurs

- o **Concepts**
- o **Classes visuelles**
- o **Gestion des couleurs**
- o **Utilisation des couleurs**
- o **Gestion des palettes**
- o **Palettes standard**

X-I

La couleur

- **Presque toutes les couleurs visibles peuvent être obtenues par somme pondérée de trois couleurs fondamentales judicieusement choisies, le *rouge*, *vert*, *bleu* (RGB en anglais).**
- **Un écran couleur est recouvert de 3 couches de phosphore. Toute partie de l'écran définissant un pixel est frappé par 3 faisceaux d'électrons indépendants. L'impact produit sur chacune de ces 3 couches l'émission d'un signal lumineux dans la couleur rouge, vert ou bleu respectivement dont l'intensité croît avec l'énergie du choc.**
- **Pour un écran donné, les valeurs possibles de chaque couleur de base sont discrétisées.**
 - ▮ **Toute combinaison des intensités de chaque couleur de base donne une couleur affichable.**
 - ▮ **Les combinaisons d'intensités sont groupés dans une *palette* ou *table des couleurs*. La taille de palette donne le nombre de couleurs simultanément affichables.**
- **Une telle structure réalise donc un *codage* des couleurs.**
- **Penser à la métaphore de la bibliothèque: vous pouvez sortir n'importe quel livre, mais à tout moment votre choix est limité à 3 ouvrages).**

X-I

Valeurs de pixels

- A tout écran est associé une *mémoire d'écran*.
 - | Elle contient une entrée par point de l'écran.
 - | Le nombre de bits par point est le *nombre de plans*.
 - | Le nombre de couleurs *simultanément affichables* est 2^N , pour N plans.
- La valeur stockée pour chaque point est une *valeur de pixel* (“pixel value”).
 - | La traduction de la valeur de pixel en couleur se fait via une *palette de couleurs* (“colormap”).
 - | Une entrée d'une palette de couleurs est une *cellule de couleur* (“color cell”).
- Pour disposer d'une couleur, le client demande un accès à une cellule de couleur. C'est l'*allocation* de couleurs. Deux méthodes:
 - | Le client *spécifie une couleur* par son nom, ou par ses composantes RGB, ou dans un autre espace de couleurs (version 5), et le serveur retourne une *valeur de pixel* correspondante.
 - | Le client demande la *réservation de cellules*; le serveur alloue les cellules et le client y affecte des couleurs par nom ou par composantes.

X-I

Noms des couleurs

- o Le système X dispose d'une base de données de noms de couleurs, habituellement dans le fichier `/usr/lib/X11/rgb.txt`
- o
- o Cette base est connue du serveur (elle est chargée avec le serveur), et peut être consultée par le client.
- o
- o Le serveur peut adapter les valeurs RGB en fonction des qualités graphiques de l'écran, au moment de l'allocation :
 - | les valeurs RGB du fichier sont *théoriques* ou *exactes*;
 - | les valeurs RGB réalisables par l'écran sont *pratiques* ou *matérielles* ("hardwarecolor").

255	250	250	snow
245	245	245	white smoke
255	255	240	ivory
255	255	255	white
0	0	0	black
192	192	192	gray
25	25	112	midnight blue
0	0	255	blue
173	216	230	light blue
0	255	255	cyan
46	139	87	sea green
0	255	0	green
127	255	0	chartreuse
240	230	140	khaki
238	232	170	PaleGoldenrod
255	255	0	yellow
255	215	0	gold
238	221	130	light goldenrod
255	165	0	orange
255	0	0	red
255	0	255	magenta
166	166	166	gray65
255	255	255	grey100

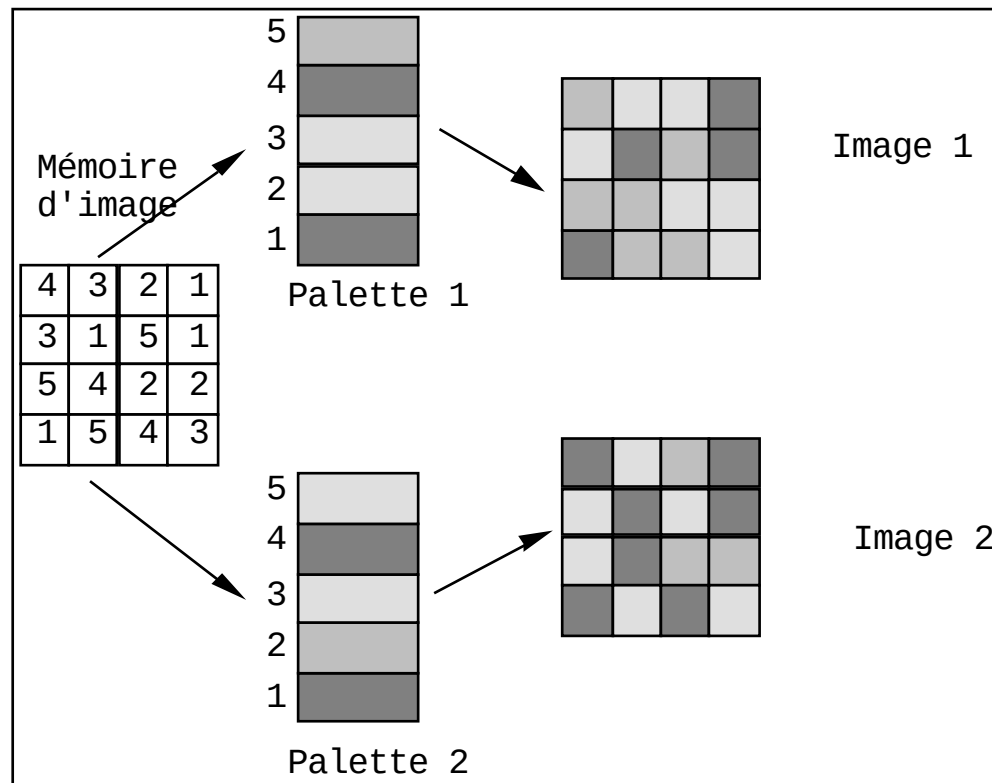
X-I

Qu'est-ce qu'une couleur pour le système X ?

```
typedef struct {  
    unsigned long pixel;  
    unsigned short red, green, blue;  
    char flags;  
    char pad;  
} XColor;
```

- o Le champ `pixel` contient la valeur de pixel. Elle est un indice dans une table de couleurs.
- o C'est cette valeur qui est utilisée dans toutes les fonctions de dessin (comme `XSetForeground`, `XDrawLine`)
- o Les composantes **RGB** sont comprises entre 0 et 65535. Ce sont soit les valeurs théoriques, soit les valeurs matérielles des intensités
- o L'indicateur `flags` est une combinaison de trois noms
 - | `DoRed`, `DoGreen`, `DoBlue` qui décrivent les composantes concernées.
 - | Prendre `DoRed | DoGreen | DoBlue`
- o Le dernier champ sert à aligner la structure sur un nombre pair de mots.

X-I

La notion essentielle: la palette

- o Si N^2 est le nombre de pixels de l'écran et M le nombre de couleurs dans la table, il suffit de modifier M entrées au lieu de N^2 (typiquement 256 au lieu de 1 0 0 0 0 0 0)
- o Un serveur possède une ou plusieurs palettes *standard* et parmi celles-ci une palette *par défaut*
- o On distingue
 - | palette *virtuelle* : structure de données X
 - | palette *physique*, qui est un dispositif de contrôle du signal video
- o Le gestionnaire de fenêtres a la responsabilité de charger les palettes virtuelles dans les palettes physiques lorsque cela est possible et nécessaire (cf. classes visuelles).

X-I

Couleurs publiques et privées

- o Les entrées dans une palette de couleurs sont limitées :
 - | noir et blanc : 2
 - | niveau de gris : 16 à 256
 - | couleur : 16 à 4096 ou plus.
- o Une cellule de couleur est *publique*, ou *consultable*, ou *partageable*, ou “read-only” si, après allocation, ses composantes RGB ne peuvent plus être modifiées.
 - | Un telle cellule peut être partagée entre plusieurs clients.
 - | Elle continue à être utilisable après la fin d’un client.
- o Une cellule est *privée*, ou *modifiable*, ou “read/write” si elle est réservée à un client.
 - | Ce client peut en modifier le contenu à tout moment.
 - | La cellule est libérée à la fin du client.

X-I

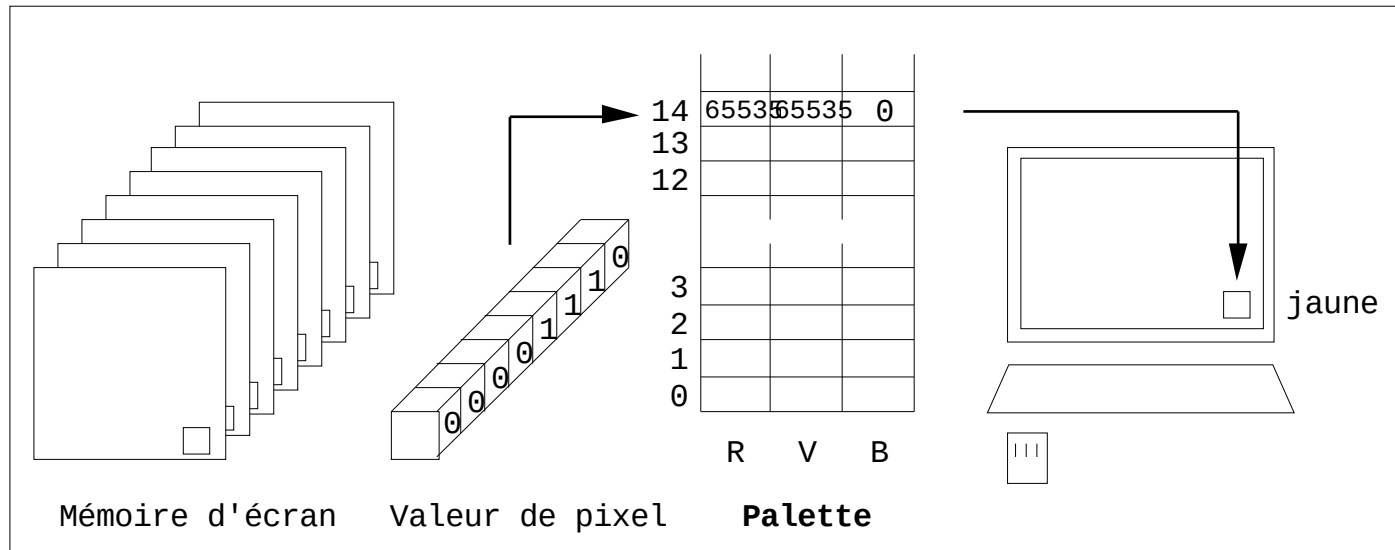
Classes visuelles

- o Pour capter les possibilités des écrans existants, X définit 6 catégories, appelées *classes visuelles*, qui codifient les possibilités graphiques d'une palette de couleur:

type de palette	read-only	read/write
monochrome/gris	StaticGray	GrayScale
indice direct pour RGB	StaticColor	PseudoColor
indices séparés RGB	TrueColor	DirectColor

- o Les palettes des classes visuelles “read-only” ont un contenu
 - | qui est prédéfini,
 - | qui ne peut être modifié,
 - | et qui remplit la table (pas d'ajout possible)
- o Les palettes des classes visuelles “read/write” ont un contenu
 - | qui est modifiable,
 - | qui peut contenir des cellules publique et des cellules privées
 - | et dont les cellules sont gérées par le serveur.
- o Les classes les plus courantes sont PseudoColor et GrayScale

X-I

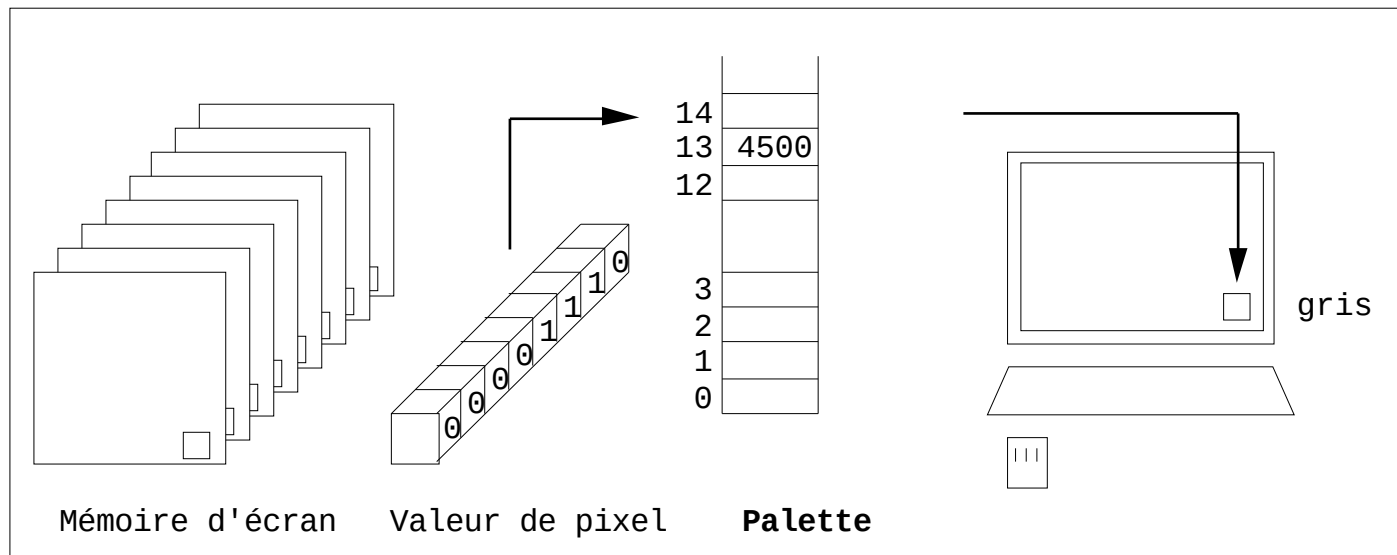
PseudoColor/StaticColor

- o La palette de couleurs contient, pour chaque valeur de pixel, trois composantes RGB.
- o Pour repeindre une partie de l'écran, il n'est pas nécessaire de changer les valeurs de pixels : il suffit de changer l'entrée de la palette.

Par exemple, pour peindre en bleu ce qui était en jaune, on attribue à l'entrée 14 de la palette les valeurs : 0, 0, 65535

X-I

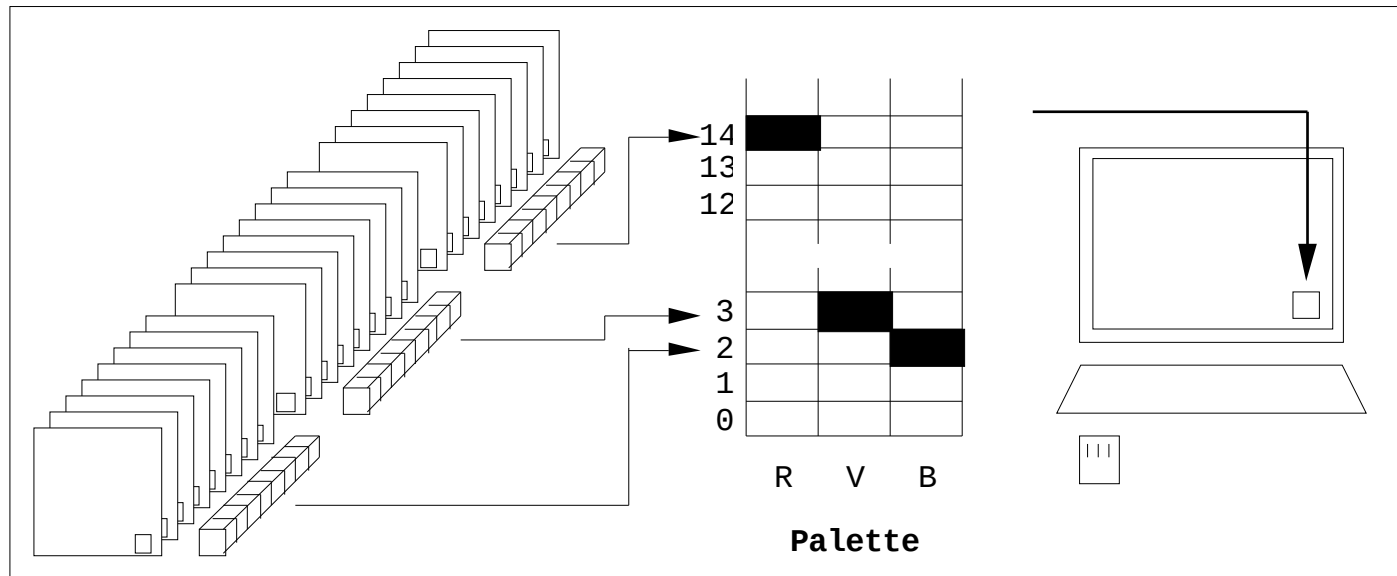
GrayScale/StaticGray



- o La palette contient, pour chaque indice, un niveau d'intensité de gris.

X-I

DirectColor/TrueColor



- o Chaque valeur de pixel est *décomposée* en trois indices.
- o Chaque indice repère indépendamment des deux autres l'intensité dans une des couleurs.
- o Si la palette a N entrées, il y a N^3 couleurs décrites. Ainsi, avec un octet par couleur, c'est-à-dire 3 octets par pixel, on a plus de 16 millions de couleurs avec une table de 256 entrées.
- o La décomposition des bits de chaque pixel est décrite par trois masques dans la structure `Visual` donnée plus loin.

X-I

Palette par défaut

- o La *palette par défaut* est accessible par une macro:
Colormap pal;
pal = DefaultColormap(dpy, ecran);
- o Le *nombre total de cellules* contenues dans la palette par défaut est un entier obtenu par
DisplayCells(dpy, ecran)
- o La *classe visuelle* par défaut s'obtient en deux temps:
 - | On passe par une structure appelée Visual ("structure visuelle") qui contient un ensemble d'informations;
 - | le champ de nom `class` contient la classe visuelle.

```
Visual *v;  
int classe;  
  
v = DefaultVisual(dpy, ecran);  
classe = v->class;
```

Les valeurs symboliques du champ class

StaticGray	0
GrayScale	1
StaticColor	2
PseudoColor	3
TrueColor	4
DirectColor	5

X-I

Le type “Visual”

- o La structure `Visual` est intéressante parce qu’elle montre comment X se réserve des possibilités d’extension.
- o Les masques ici servent à décomposer une valeur de pixel dans le cas de l’indexation séparée `DirectColor/TrueColor`.
- o Le champ `map_entries` est le nombre d’entrées dans la palette.

```
typedef struct {  
    XExtData *ext_data;           /* hook for extension to hang data */  
    VisualID visualid;            /* visual id of this visual */  
    int class;                    /* class of screen (monochrome, etc.) */  
    unsigned long red_mask, green_mask, blue_mask; /* mask values */  
    int bits_per_rgb;             /* log base 2 of distinct color values */  
    int map_entries;              /* color map entries */  
} Visual;
```

- o Chaque fenêtre a, parmi ses attributs, un pointeur vers une structure `Visual`, dont la valeur est héritée en général de la mère.
- o La commande `xdpyinfo` donne les informations sur les capacités de l’écran.

X-I

Connaître les caractéristiques d'une station

- o Une station peut posséder plusieurs classes visuelles, mais une d'entre elles est la classe par *défaut*. Il existe des fonctions et macros qui renseignent sur les caractéristiques de cette classe ou sur l'ensemble des classes d'une station
- o Nombre maximal de plans à l'écran
`nplans = DisplayPlanes(dpy, ecran);`
- o Nombre de plans par défaut de la fenêtre racine
`profondeur = DefaultDepth(dpy, ecran);`
- o Nombre de cellules de la palette par défaut
`ncellules = DisplayCells(dpy, ecran);`
- o Valeur du blanc et du noir
- o La palette par défaut
`XColormap pal;`
`pal = DefaultColormap(dpy, ecran);`
- o Classe visuelle par défaut
`Visual visuel;`
`visuel = DefaultVisual(dpy, ecran);`

X-IExemple

```
vendor string: Tektronix, Inc.  
depths (2):      1, 8  
depth of root window:      8 planes  
number of colormaps:      minimum 1, maximum 1  
default colormap:      0x26  
default number of colormap cells:      256  
preallocated pixels: black 0, white 1  
  
number of visuals:      6  
default visual id:      0x20  
visual:  
visual id:      0x20  
class:      PseudoColor  
depth:      8 planes  
size of colormap:      256 entries  
red, green, blue masks:      0x0, 0x0, 0x0  
significant bits in color specification:      8 bits  
  
visual id:      0x24  
class:      TrueColor  
depth:      8 planes  
size of colormap:      8 entries  
red, green, blue masks:      0x7, 0x38, 0xc0  
significant bits in color specification:      8 bits
```

X-I

Allouer des couleurs

- o **Allouer des couleurs partageables**
 - | XAllocColor valeur de pixel de RGB
 - | XAllocNamedColor valeur de pixel de nom
- o **Consulter une palette**
 - | XLookupColor composantes RGB exactes et effectives d'une couleur nommée
 - | XQueryColor composantes RGB d'une valeur de pixel
 - | XParseColor composantes RGB exactes d'une couleur nommée
- o **Allouer des couleurs privées**
 - | XAllocColorCells allocation de cellules personnelles
 - | XStoreColor remplissage avec une cellule de couleur.

X-I

Allouer des couleurs partageables

```
XAllocColor(dpy, pal, &couleur)  
avec  
XColor couleur;
```

- o En *entrée*, couleur contient les composantes RGB de la couleur;
- o En *sortie*, couleur contient la valeur de pixel et les composantes RGB *pratiques*, calculées au mieux.
- o Si la palette est non modifiable (StaticColor, TrueColor, StaticGray), la valeur retournée est la plus proche possible;
- o Si la palette est modifiable (PseudoColor, DirectColor, GrayScale), il peut y avoir échec de l'allocation (et XAllocColor retourne 0) lorsque
 - | Il n'y a plus de cellule libre et
 - | aucune des couleurs existantes ne contient les valeurs RGB demandées (à la correction locale près)

X-I

Exemple (gerercouleurs.c)

```
void NouvelleCouleur(void)
{
    int r,g,b;
    XColor C;

    printf("Composantes RGB : ");
    scanf("%d", &r); scanf("%d", &g); scanf("%d", &b);
    printf("J'ai lu : %d %d %d\n", r,g,b);
    C.red =r;C.green = g; C.blue =b;

    if (XAllocColor(dpy,pal, &C))
        printf("Pixel %d, rouge = %d, vert = %d, bleu = %d\n",
            C.pixel, C.red, C.green, C.blue)
    else
        printf("Pas d'allocation.\n");
}
```

X-IScript

```
Nombre de couleurs 256
Pixel 0,   rouge = 0,      vert = 0,      bleu = 0
Pixel 1,   rouge = 65535,  vert = 0,      bleu = 0
Pixel 2,   rouge = 0,      vert = 65535,   bleu = 0
Pixel 3,   rouge = 65535,  vert = 65535,   bleu = 0
Pixel 13,  rouge = 36494,  vert = 14392,   bleu = 36494
  Composantes RGB : 40000 25000 56000
  J'ai lu : 40000 25000 56000
Pixel 127, rouge = 40092,  vert = 24929,   bleu = 56026
  Composantes RGB : 41000 25000 56000
  J'ai lu : 41000 25000 56000
Pixel 128, rouge = 41120,  vert = 24929,   bleu = 56026
  Composantes RGB : 42000 25000 56000
  J'ai lu : 42000 25000 56000
Pixel 129, rouge = 42148,  vert = 24929,   bleu = 56026
  Composantes RGB : 42500 25000 56000
  J'ai lu : 42500 25000 56000
Pixel 130, rouge = 42662,  vert = 24929,   bleu = 56026
```

X-I

Allocation par le nom

```
XAllocNamedColor(dpy, pal, nom,&couleur_effective, &couleur_exacte);  
avec  
XColor couleur_exacte, &couleur_effective;  
char *nom;
```

- o En *entrée*, on fournit un *nom de couleur* ou une spécification démodée;
- o En *sortie*, on récupère
 - | l'équivalent RGB *effectif* et la valeur de pixel allouée dans la première couleur;
 - | l'équivalent RGB *exact* dans la première couleur

```
unsigned long PixelDeCouleur(char *nom)  
{  
    XColor couleur,couleur_exacte;  
    if ( !XAllocNamedColor(dpy, DefaultColormap(dpy,ecran),  
                           nom, &couleur, &couleur_exacte))  
    {  
        printf("Probleme d'allocation pour %s\n",nom);  
        return WhitePixel(dpy,ecran);  
    }  
    return couleur.pixel;  
}
```

X-I

Consulter une palette

```
XLookupColor(dpy, pal, nom, &coul_exacte, &coul_eff)
```

- o **Consulte l'écran pour obtenir les composantes RGB exactes et effectives sur l'écran. L'argument `pal` n'est présent que pour spécifier l'écran.**

```
XParseColor(dpy, pal, nom, &coul_exacte)
```

- o **Retourne les composantes RGB exactes. L'argument `nom` peut être une spécification de l'ancien style.**
- o **L'argument `pal` n'est pas utilisé.**
- o **Les spécifications anciennes sont**
 - | **#RGB, #RRGGBB, #RRRGGBBB ou #RRRRGGGGBBBB**
 - | **par exemple #0f0, #00ff00, #000fff000 ou #0000ffff0000**

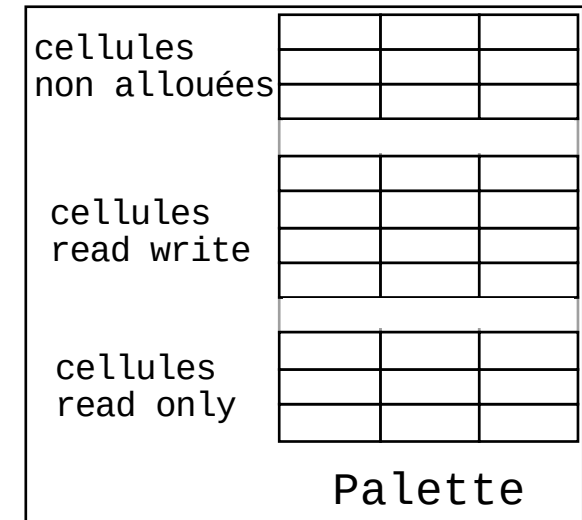
```
XQueryColor(dpy, pal, &couleur)
```

- o **En *entrée*, `couleur` contient une valeur de pixel;**
- o **en *sortie*, les composantes RGB de la couleur correspondante.**

X-I

Allouer des couleur privées

- o Le processus est en deux étapes:
 - | L'allocation de cellules (i. e. leur réservation);
 - | l'attribution de couleurs à ces cellules (i. e. leur affectation).
- o La fonction d'allocation permet des réservations de cellules et de plans.
- o Syntaxe:



```
Status XAllocColorCells(dpy, pal, contigu, m, k, p, n)
Display*      dpy,
Colormap      pal,
Bool          contigu,
unsigned long* m /* plane_masks_return */ ,
unsigned int  k /* nplanes */ ,
unsigned long* p /* pixels_return */ ,
unsigned int   n /* npixels */ ;
```

X-I

Allocation sans plans

- o La plupart du temps, on ne fait pas usage de l'allocation de plans. Dans ce cas, on appelle la fonction par

```
XAllocColorCells(dpy, pal, False, NULL, 0, p, n);
```

- o En cas d'allocation réussie (retour non null), le tableau `p` contient `n` valeurs de pixels correspondant à des cellules de couleurs allouées au client.

- o Pour ranger une couleur :

```
XStoreColor(dpy, pal, &couleur);
```

- l En *entrée*, `couleur` contient une valeur de pixel et une définition RGB.
- l En *sortie*, `couleur` contient les valeurs RGB effectives.
- l La fonction range cette définition dans la palette.

- o Il existe une version plurielle:

```
XStoreColors(dpy, pal, c, nc);
```

- l où `c` est un tableau de `nc` couleurs.

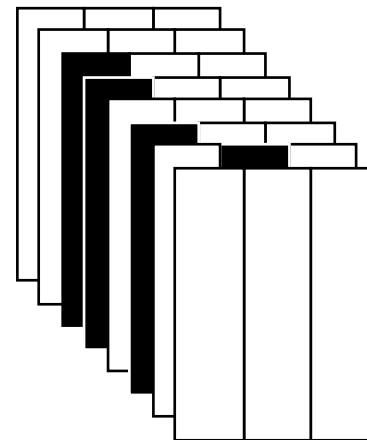
X-I

Superposition de plans (overlay)

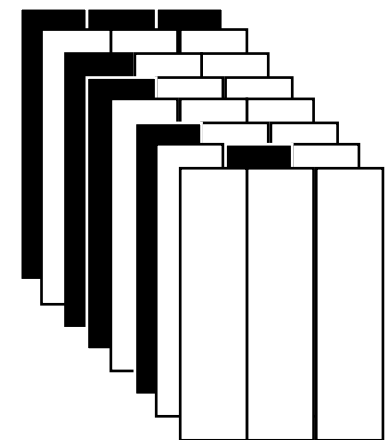
- o La superposition de plans est une technique ancienne et fréquemment utilisée pour afficher et effacer une partie d'un dessin, par arithmétique sur les valeurs de pixels.
- o Exemple.
- o On dispose de 6 valeurs de pixels (des entiers), dont 3 sont paires et 3 des nombres impairs qui les suivent (par exemple 0, 42, 64 et 1, 43, 65).
- o A chaque valeur de pixel est attachée une couleur.

- o Une fenêtre contient un dessin fait en couleurs paires.
 - | On veut *ajouter* un dessin dans les couleurs impaires associées.
 - | Et on veut *effacer* le dessin impair sans changer ni retracer le dessin pair.

00000000	0	noir
00101100	42	vert
01000000	64	jaune



00000001	1	rouge
00101101	43	bleu
01000001	65	cyan

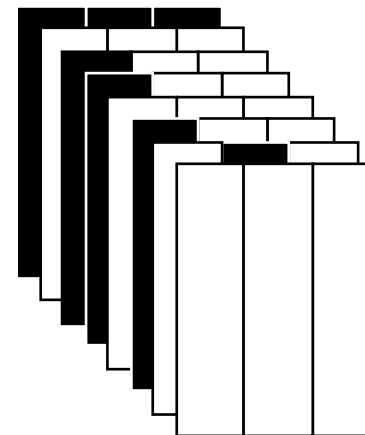
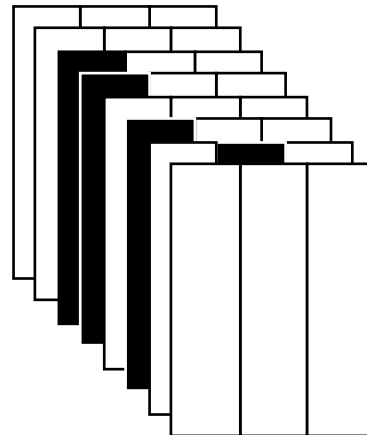


X-I

Superposition de plans (overlay)

00000000	0	noir
00101100	42	vert
01000000	64	jaune

00000001	1	rouge
00101101	43	bleu
01000001	65	cyan



- o Pour *ajouter* le dessin impair, on ajoute 1 aux valeurs de pixels concernées, par un contexte graphique a spécifié par:

```
SetPlaneMask(dpy, a, 1);
SetFunction(dpy, a, GXset);
```

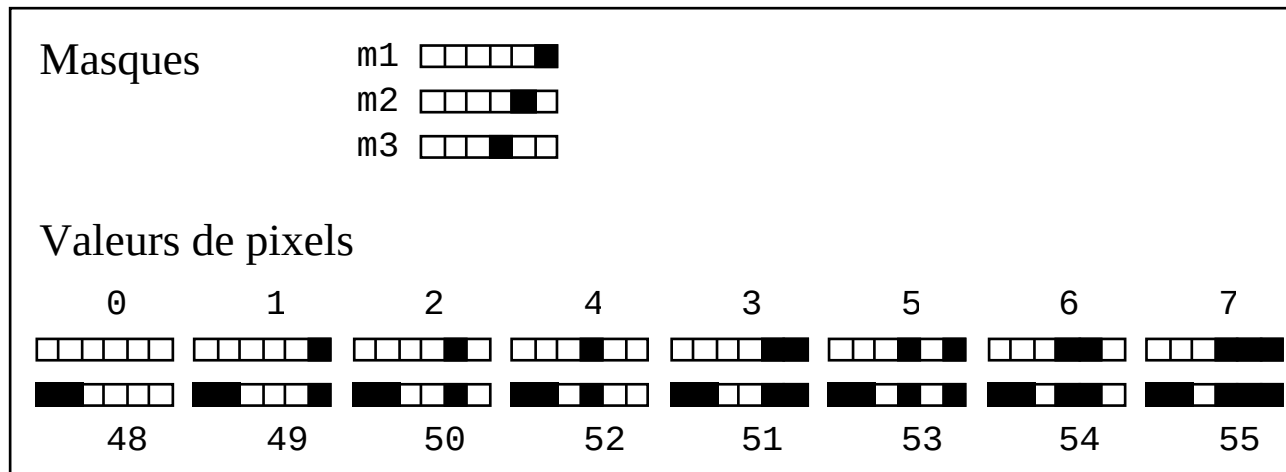
- o Pour *effacer* le dessin impair, on met 0 dans le dernier bit des valeurs de pixels par un contexte graphique e spécifié par:

```
SetPlaneMask(dpy, e, 1);
SetFunction(dpy, e, GXclear);
```

X-I

Extensions

- o Les couleurs sont groupées selon le k-ième bit qui n'est pas forcément le dernier:
 - | les valeurs de pixel ayant 0 dans le k-ième bit (couleurs “paires”)
 - | les valeurs de pixel ayant 1 dans le k-ième bit (couleurs “impaires”)
 - | le plan est défini par `SetPlaneMask(dpy, a, m)` et $m = 1 \ll k$.
- o On a plusieurs masques, disons 3, et les valeurs de pixels sont groupées en $8=2^3$ classes.



X-I

Allocation avec plans

- o **L'appel de fonction**

```
XAllocColorCells(dpy, pal, False ou True, m, k, p, n);
```

alloue $m \cdot 2^k$ couleurs, de la manière suivante:

- o $m[0], \dots, m[k-1]$ contiennent chacun un masque avec exactement un bit à 1, et les masques sont distincts;
- o $p[0], \dots, p[n-1]$ sont n valeurs de pixels distincts dont tous les bits couverts par un masque sont nuls.
- o Les valeurs de pixel des cellules allouées dans la palette s'obtiennent en partant d'un $p[i]$ et en faisant les 2^k combinaisons "ou" possibles avec les masques.
- o L'argument `contigu` vaut `False` ou `True`; si `True`, on demande que les bits égaux à 1 dans les masques soient contigus. Ceci permet de l'arithmétique sur les valeurs de pixels.

X-I

Palettes physiques et virtuelles

Le terme palette recouvre deux notions:

- **Palette *physique* (matérielle):** dispositif de pilotage de l'écran. Chaque écran en possède au moins une, certains plusieurs;
- **Palette *virtuelle* (notion X):** structure de données contenant les informations nécessaires aux palettes physiques.
- **Si la palette physique de l'écran ne peut être modifiée (palette *immuable*)**
 - | aucune allocation privée n'est possible;
 - | les valeurs RGB sont fixes;
 - | la classe visuelle est donc `StaticColor`, `StaticGray`, `TrueColor`.
- **Si la palette est modifiable,**
 - | on peut changer des valeurs RGB dans cette palette;
 - | on peut remplacer le contenu de la palette globalement.
- **X gère des palettes multiples, en conservant les palettes virtuelles en mémoire et en changeant le contenu de la palette physique.**

X-I

Créer et attribuer une palette

- o Les applications créent et gèrent les palettes virtuelles

```
Colormap XCreateColormap(dpy, fen, visual, allouer)  
Visual *visual;  
int allouer;
```

- l La fenêtre sert à connaître l'écran auquel est associé la palette;
- l allouer vaut AllocNone ou AllocAll.
 - À AllocNone : aucune cellule n'est réservée au client. On peut alors allouer des couleurs privées ou publiques.
 - À AllocAll : toutes les cellules sont réservées au client, donc privées.

- o Pour associer une palette à une fenêtre, on fixe l'attribut Colormap, soit à la création de la fenêtre, soit par XChangeWindowAttributes, soit par

```
XSetWindowColormap(dpy, fen, pal);
```

X-I

Installer une palette

- ***Installer*** une palette consiste à charger une palette virtuelle dans une palette physique.
- Si l'écran possède une seule palette physique, l'installation provoque des modification de couleurs sur *tout* l'écran.
- Les règles de l'ICCCM (*Inter Client Communication Convention Manual*) demandent que l'installation des palettes soit de la responsabilité du gestionnaire de fenêtres.
- L'installation d'une nouvelle palette provoque l'envoi d'un évènement ColormapNotify.

X-I

Palettes standard

- o En plus des palettes de couleurs dont les cellules sont gérées soit explicitement, soit implicitement, X fournit la possibilité d'utiliser des palettes prédéfinies, immuables, contenant des nuances de couleurs, à l'arithmétique exploitable: les *palettes standard*.
- o Une *palette standard* est une palette (virtuelle) où la correspondance entre valeurs de pixels et couleurs est prévisible et prédéfinie.
- o Exemple : palette de nuances de rouge, de bleu ou de vert.
- o X spécifie des palettes standard via des *propriétés*.
- o Le calcul des valeurs de pixels se fait par une formule particulière

```
typedef struct {  
    Colormap colormap;  
    unsigned long red_max;  
    unsigned long red_mult;  
    unsigned long green_max;  
    unsigned long green_mult;  
    unsigned long blue_max;  
    unsigned long blue_mult;  
    unsigned long base_pixel;  
    VisualID visualid;  
    XID killid;  
} XStandardColormap;
```

X-I

Calculer la valeur de pixel

```
XStandardColormap s;  
unsigned long pixel;
```

- o **Si les composantes fondamentales sont données par des valeurs entières r, g, b vérifiant $0 \leq r \leq \text{red_max}$, $0 \leq g \leq \text{green_max}$, $0 \leq b \leq \text{blue_max}$, alors :**

```
pixel = s.base_pixel +  
        r* s.red_mult + g* s.green_mult + b* s.blue_mult;
```

- o **Si les composantes fondamentales sont données par des valeurs décimales R, G, B vérifiant $0.0 \leq R \leq 1.0$, $0.0 \leq G \leq 1.0$, $0.0 \leq B \leq 1.0$, alors on les convertit en entiers, et on applique la même formule :**

```
unsigned long r,g,b;  
r = (unsigned long)(0.5 + R*s.red_max);  
g = (unsigned long)(0.5 + G*s.green_max);  
b = (unsigned long)(0.5 + B*s.blue_max);  
pixel = s.base_pixel +  
        r* s.red_mult + g* s.green_mult + b* s.blue_mult;
```


X-I

Exemple

o Avec

red_max = 7
green_max = 7
blue_max = 3

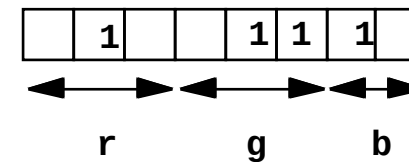
il y a 8 nuances de rouge et de vert, et 4 nuances de bleu, donc au total 256 couleurs.

o Si

red_mult = 32
green_mult = 4
blue_mult = 1
base_pixel = 0

les nuances sont sur les 3 bits de plus fort poids pour les rouges, sur les 3 bits suivants pour les verts, et les deux bits restant pour les bleus.

```
typedef struct {  
    Colormap colormap;  
    unsigned long red_max;  
    unsigned long red_mult;  
    unsigned long green_max;  
    unsigned long green_mult;  
    unsigned long blue_max;  
    unsigned long blue_mult;  
    unsigned long base_pixel;  
    VisualID visualid;  
    XID killid;  
} XStandardColormap;
```



X-I

Utiliser une palette standard

```
XStandardColormap s;  
if (!XGetStandardColormap(dpy,  
    DefaultRootWindow(dpy), &s, XA_RGB_GRAY_MAP))  
{  
    printf("Palette inexistante");  
    exit(1);  
}  
XSetWindowColormap(dpy, fen, s.colormap);
```

- **On récupère les informations nécessaires dans une palette standard.**
- **La fonction teste si la propriété est bien définie et, dans l'affirmative, retourne l'identifiant dans le champ colormap.**
- **Si la palette standard n'existe pas, on peut la fabriquer à la main ou utiliser le client X xstdcmap qui fait ce travail.**

X-I

Une station possède en général plusieurs palettes par défaut qui sont connues par leur nom symbolique (des *atomes*). Certaines des entrées de ces palettes sont libres.

XA_RGB_COLOR_MAP sous-ensemble de la palette par défaut attachée à la racine. Configuration typique pour une palette à 256 entrées: 6 valeurs pour chacun des 3 champs (soit $6 \times 6 \times 6 = 216$ entrées uniformément réparties).

XA_RGB_BEST_MAP définit un ensemble “optimal” de couleurs visiblement différentes. Configuration typique avec 256 entrées: 7 valeurs pour le rouge et le vert, 3 valeurs pour le bleu ($256 - 7 \times 7 \times 3 = 109$ entrées disponibles)

o **Atomes des palettes standard**

```
XA_RGB_COLOR_MAP
XA_RGB_BEST_MAP
XA_RGB_BLUE_MAP
XA_RGB_DEFAULT_MAP
XA_RGB_GRAY_MAP
XA_RGB_GREEN_MAP
XA_RGB_RED_MAP
```

X-I

Extensions X11R5

- o **La version 5 introduit le “Color Managing System” qui permet de spécifier des couleurs dans divers formats (“espaces de couleurs”):**
 - | **RGB : composantes r,g, b entre 0 et 65535;**
 - | **RGBi: composantes r,g, b entre 0.0 et 1.0;**
 - | **CIEXYZ : modèle CIE, composantes entre 0 et 1**
 - | **CIEuvY, CIExyY: variantes du modèle CIE**
 - | **TekHVC: (hue,value,chroma)**
 - | **...**
- o **Les fonctions de gestion de couleurs prennent le préfixe cms et un argument supplémentaire:**
 - | **`XcmsAllocColor(dpy, pal, &coul, format);`**
 - | **format prend une des valeurs:**
 - | **`XcmsRGBFormat, XcmsRGBiFormat, XcmsCIEFormat...`**
 - | **coul est de type `XcmsColor` qui est essentiellement l’union des types pour chaque spécification d’espace.**