

X-I

Xlib : les dessins

- o **Le processus graphique**
- o **Primitives de tracé**
- o **Les contextes graphiques**
- o **Pixmap et remplissage**
- o **Evènements associés aux dessins**

X-I

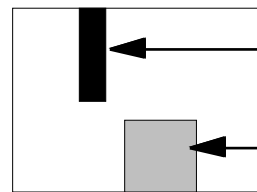
- **Que dessiner ?** Les *primitives graphiques* permettent une assez grande variété de dessins d'objets graphiques:
 - | des points, lignes, arcs, cercles, ellipses, rectangles, polygones.
- **Comment dessiner ?** Une structure unique, appelée *contexte graphique*, regroupe l'ensemble des valeurs des paramètres applicables à un tracé. Plusieurs contextes graphiques peuvent coexister simultanément. Les paramètres concernent :
 - | tracé : épaisseur des traits, pointillés, arrondis en fin de trait, liaison entre traits consécutifs.
 - | remplissage : motifs et pavages, couleurs.
 - | découpage: par zone de “clipping”; respect des sous-fenêtres.
 - | fonction “logique” de combinaison du dessin fait avec le dessin existant.
- **Où dessiner ?** Les dessins se font indistinctement dans des fenêtres et ou des pixmaps, réunis sous le terme de “drawables” ou *planches*.

X-I

Processus graphique

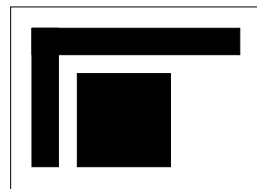
- o La réalisation des dessins se fait en 5 étapes qui constituent le *processus graphique*.
- o Durant les 4 premières étapes, les dessins se font (conceptuellement) dans une *planche brouillon*.
- o Lors de la dernière étape, le brouillon est *combiné*, au moyen d'une fonction "logique", avec le dessin existant, et substitué au dessin de la fenêtre *cible*.
- o Etapes
 1. *Former* le dessin par les primitive graphique (traits, polygones,...) par *sélection* de pixels dans le brouillon.
 2. *Peindre* le dessin en appliquant un *motif* ou une *couleur* aux pixels sélectionnés.
 3. *Restreindre* le dessin à la zone de découpage ("clipping") éventuelle.
 4. *Exclure* (ou au contraire conserver) du dessin les régions occupées par des fenêtres descendantes.
 5. *Combiner* le brouillon avec le dessin existant au moyen d'une fonction "logique".

X-I

Exemple

○ Dessin existant
○ Fenêtre destination
○ Fenêtre fille

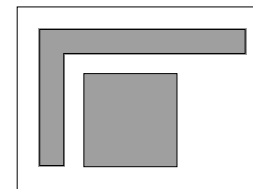
Dessin



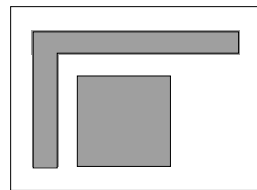
+



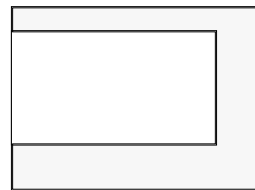
=



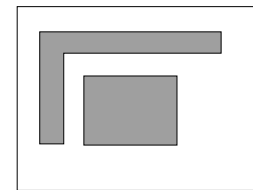
Découpage



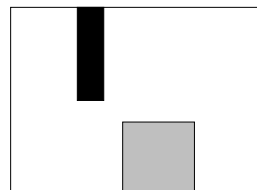
+



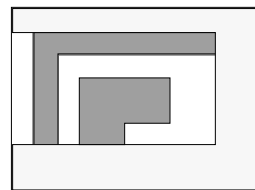
=



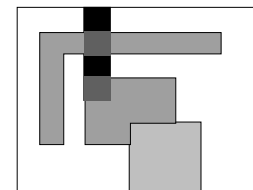
fonction xor



+



=



○ Résultat

X-I

Les contextes graphiques

- Un *contexte graphique* (type GC) est un pointeur vers une structure opaque qui contient l'ensemble des paramètres de mise en forme de primitives de dessin. La structure est conservée par le client, mais une structure correspondante existe bien entendue sur le serveur.
- Ces *paramètres* sont réunis dans une structure `XGCValues` dont les champs sont accessibles et modifiables soit par la méthodes des masques, soit par des fonctions utilitaires.
- La *création* d'un contexte graphique se fait par `XCreateGC`.
- La *modification* d'un contexte graphique se fait par `XChangeGC` ou mieux encore par des fonctions utilitaires.
- A la connexion, le serveur retourne au client un contexte graphique par défaut que l'on peut appeler par `DefaultGC`.

X-I

Création

- o **Version générale:** spécification éventuelle de valeurs des paramètres à la création

```
GC ctx;  
XGCValues x;  
Mask masque;  
...  
ctx =XCreateGC(dpy, fen, masque, &x);
```

- o **Version simplifiée:** spécification de nouvelles valeurs après création, soit globalement, soit par des fonctions utilitaires

```
GC ctx;  
...  
ctx =XCreateGC(dpy, fen, 0, NULL);
```

```
ctx =DefaultGC(dpy, DefaultScreen(dpy));
```

- o **NB:** Malgré la présence d'une fenêtre (fen) dans les paramètres, les contextes graphiques sont *globaux* à l'application, et sont applicables au dessins *dans toutes les planches* qui ont la *même profondeur* que la fenêtre fen .

```
GC ctx;  
...  
ctx =XCreateGC(dpy, DefaultRootWindow(dpy), 0, NULL);
```

X-I

Fonctions de tracé

XDrawPoint	affiche un point
XDrawPoints	affiche une suite de points
XDrawLine	trace un segment
XDrawLines	trace une suite de segments consécutifs
XDrawSegments	trace une suite de segments
XDrawRectangle	dessine le contour d'un rectangle
XDrawRectangles	dessine plusieurs rectangles
XDrawArc	trace un arc d'ellipse
XDrawArcs	trace des arcs d'ellipse

- o **Dans la suite**
 - d** désigne le display
 - f** une fenêtre
 - C** un contexte graphique

X-I

Points

Type point :

```
typedef struct {  
    short x, y;  
} XPoint;
```

- o Tracer un point : `XDrawPoint(d, f, c, x, y)`
`int x, y;`

- o Tracer une suite de points: `XDrawPoints(d, f, c, p, np, mode)`
`XPoint *p;`
`int np;`
`int mode;`

mode prend une des deux valeurs suivantes:

CoordModOrigin

les coordonnées des points sont par rapport à l'origine de la fenêtre
(coordonnées absolues)

CoordModPrevious

les coordonnées des points sont par rapport au point précédent
(coordonnées relatives), sauf pour le premier.

X-I

Lignes

- o Tracer une ligne:

```
XDrawLine(d, f, c, x1, y1, x2, y2)  
    int x1, y1, x2, y2
```

- o Tracer des lignes consécutives :

```
XDrawLines(d, f, c, p, np, mode)  
    XPoint *p;  
    int np;  
    int mode;
```

- o Tracer des segments de droites :

```
XDrawSegments(d, f, c, s, ns)  
    XSegment *s;  
    int ns;
```

Type segment :

```
typedef struct {  
    short x1, y1, x2, y2;  
} XSegment;
```


X-I

Rectangles

- o **Tracer le contour d'un rectangle:**

```
XDrawRectangle(d, f, c, x, y, l, h)
    int x,y;           (coin supérieur gauche)
    unsigned int l,h;   (largeur et hauteur)
```

- Type d'un rectangle:**

```
typedef struct {
    short x, y;
    unsigned short
        width, height;
} XRectangle;
```

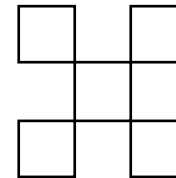
- o **Tracer des rectangles :**

```
XDrawRectangles(d, f, c, r, nr)
    XRectangle *r;
    int nr;
```

- o **Exemple :**

```
XDrawRectangles(d, f, c, r, 5);
```

```
XRectangle r[5] =
    {{10,10,5,5},{20,10,5,5},{15,15,5,5},{10,20,5,5},{20,20,5,5}};
```

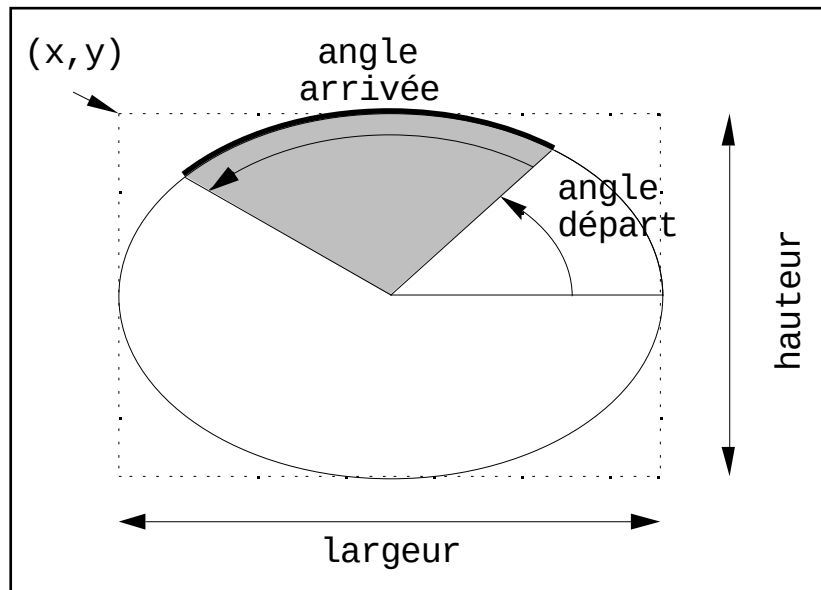


X-I

Arcs

o Tracer un arc d'ellipse:

XDrawArc(d, f, c, x, y, l, h, angledepart, anglearrivee)
 int x,y; (coin supérieur gauche)
 unsigned int l,h; (largeur et hauteur)
 int angledepart, anglearrivee; (en 64ème de degrés)



o un cercle complet a donc une différence d'angles égale à

$$360 \times 64 = 23040$$

o Un cercle de rayon r, et de centre (cx, cy), s'obtient par :

```
XDrawArc(d, f, c, cx-r, cy-y, r, r, 0, 23040)
```

X-I

- o **Tracer des arcs d'ellipse:**

```
XDrawArcs(d, f, c, a, na)
    XArc *a;
    int na;
```

Le type est:

```
typedef struct {
    short x, y;
    unsigned short width, height;
    short angle1, angle2;
} XArc;
```

- o **Il est important de vérifier que la largeur et la hauteur sont bien positifs avant de lancer la commande.**

X-I

Fonction de remplissage

XFillRectangle	remplit un rectangle
XFillRectangles	remplit plusieurs rectangles
XFillArc	remplit un arc d'ellipse
XFillArcs	remplit des arcs d'ellipse
XFillPolygon	remplit un polygone

- o Pour tracer le contour *et* remplir un objet, appliquer `Fill` puis `Draw`

X-I

XFillRectangle (d, f, c, x, y, largeur, hauteur)

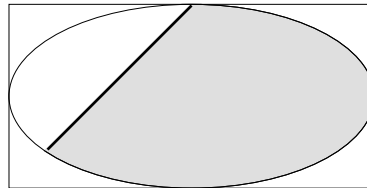
XFillRectangles (d, f, c, r, nr)

- | remplit le ou les rectangles selon le style de remplissage du contexte C.
- | Si le style est FillSolid, c'est la couleur d'encre qui est utilisée.

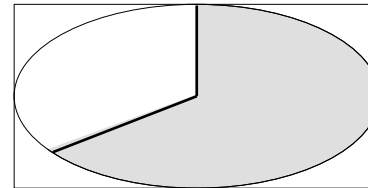
XFillArc(d, f, c, x, y, largeur, hauteur, a_dep, a_arr)

XFillArcs(d, f, c, a, na)

- | remplit le ou les arcs.
- | deux façons de fermer l'arc, selon que arc_mode est ArcChord ou ArcPieSlice.



ArcChord



ArcPieSlice

X-I

Remplissage de polygones

```
XFillPolygon(d, f, c, p, np, forme, mode)
```

```
    XPoint *p;
```

```
    int np;
```

```
    int forme;
```

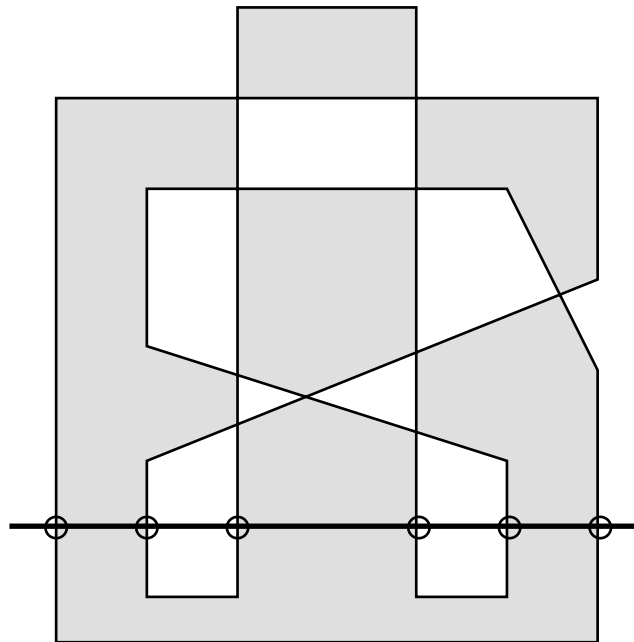
```
    int mode;      /* CoordModeOrigin ou CoordModePrevious */
```

- o **forme est une aide (*) au serveur pour choisir l'algorithme d'affichage:**
 - | Convex si le polygone est convexe,
 - | Nonconvex si le polygone ne se coupe pas,
 - | Complex cas général.
- o **Remplissage selon que, dans le contexte c, le champ fill_rule vaut**
EvenOddRule ou
WindingRule.

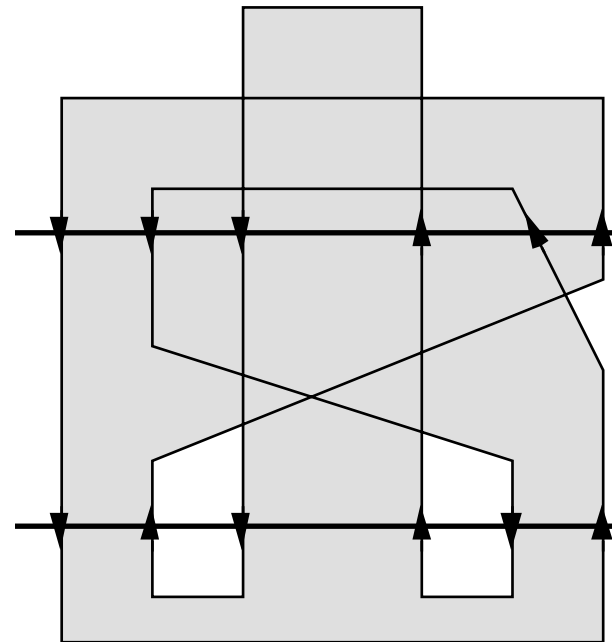
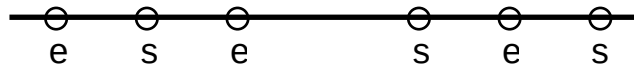
(*) Les fonctions de tracé et de remplissage sont optimisées sur le serveur. Selon la nature des objets, on prend l'algorithme le plus rapide.

X-I

EvenOdd et Winding



EvenOddRule



WindingRule



X-I

Contextes graphiques

- Un *contexte graphique* (type GC) est un pointeur vers une structure opaque qui contient l'ensemble des paramètres de mise en forme de primitives de dessin. La structure est conservée par le client, mais une structure correspondante existe bien entendue sur le serveur.
- Ces *paramètres* sont réunis dans une structure `XGCValues` dont les champs sont accessibles et modifiables soit par la méthodes des masques, soit par des fonctions utilitaires.
- Les paramètres se groupent en:
 - | **Forme et couleurs**
 - | **Attributs de lignes**
 - | **Attributs de remplissage**
 - | **Découpage (clipping)**
 - | **et quelques attributs plus difficiles à cataloguer**

X-I

Structure C

```

typedef struct {
    int function;           /* logical operation */
    unsigned long plane_mask; /* plane mask */
    unsigned long foreground; /* foreground pixel */
    unsigned long background; /* background pixel */
    int line_width;        /* line width */
    int line_style;        /* LineSolid, LineOnOffDash, LineDoubleDash */
    int cap_style;         /* CapNotLast, CapButt, CapRound, CapProjecting */
    int join_style;        /* JoinMiter, JoinRound, JoinBevel */
    int fill_style;        /* FillSolid, FillTiled, FillStippled, FillOpaqueStippled */
    int fill_rule;         /* EvenOddRule, WindingRule */
    int arc_mode;          /* ArcChord, ArcPieSlice */
    Pixmap tile;           /* tile pixmap for tiling operations */
    Pixmap stipple;        /* stipple 1 plane pixmap for stippling */
    int ts_x_origin;       /* offset for tile or stipple operations */
    int ts_y_origin;
    Font font;             /* default text font for text operations */
    int subwindow_mode;    /* ClipByChildren, IncludeInferiors */
    Bool graphics_exposures; /* boolean, should exposures be generated */
    int clip_x_origin;     /* origin for clipping */
    int clip_y_origin;
    Pixmap clip_mask;      /* bitmap clipping; other calls for rects */
    int dash_offset;       /* patterned/dashed line information */
    char dashes;
} XGCValues;

```

X-I

Valeurs par défaut

int	function;	GCCopy	#define	GCFunction	(1L<<0)
unsigned_long	planemask;	tous à 1	#define	GCPlaneMask	(1L<<1)
unsigned long	foreground;	0	#define	GCForeground	(1L<<2)
unsigned long	background;	1	#define	GCBgground	(1L<<3)
int	line_width;	0	#define	GCLineWidth	(1L<<4)
int	line_style;	LineSolid	#define	GCLineStyle	(1L<<5)
int	cap_style;	CapButt	#define	GCCapStyle	(1L<<6)
int	join_style;	JoinMiter	#define	GCJoinStyle	(1L<<7)
int	fill_style;	FillSolid, FillTiled,	#define	GCFillStyle	(1L<<8)
int	fill_rule;	EvenOddRule	#define	GCFillRule	(1L<<9)
int	arc_mode;	ArcPieSlice	#define	GCTile	(1L<<10)
Pixmap	tile;	couleur de foreground	#define	GCStipple	(1L<<11)
Pixmap	stipple;	rempli de 1	#define	GCTileStipXOrigin	(1L<<12)
int	ts_x_origin;	0	#define	GCTileStipYOrigin	(1L<<13)
int	ts_y_origin;	0	#define	GCFont	(1L<<14)
Font	font;	dépend de l'implém.	#define	GCSubwindowMode	(1L<<15)
int	subwindow_mode;	ClipByChildren	#define	GCGraphicsExposures	(1L<<16)
Bool	graphics_exposures;	True	#define	GCClipXOrigin	(1L<<17)
int	clip_x_origin;	0	#define	GCClipYOrigin	(1L<<18)
int	clip_y_origin;	0	#define	GCClipMask	(1L<<19)
Pixmap	clip_mask;	None	#define	GCDashOffset	(1L<<20)
int	dash_offset;	0	#define	GCDashList	(1L<<21)
char	dashes;	4	#define	GCArcMode	(1L<<22)

X-I

Composantes d'un contexte graphique

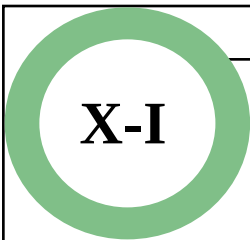
o **Forme et couleurs**

Membre	masque	Valeur par défaut
function	GCFunction	CXCopy
plane_mask	GCPlaneMask	que des 1
foreground	GCForeground	0
background	GCBbackground	1

l'attribut background définit une deuxième couleur, de remplissage. Elle sert dans : XDrawImageString, dans FillOpaqueStippled et LineDoubleDash. Ne pas confondre avec le background dans la définition d'une fenêtre.

o **Attributs de lignes**

Membre	masque	Valeur par défaut
line_width	GCLineWidth	0
line_style	GCLineStyle	LineSolid
cap_style	GCCapStyle	CapButt
join_style	GCJoinStyle	JoinMiter
dash_offset	GCDashOffset	0
dashes	GCDashList	4

**X-I**o **Remplissage**

Membre	masque	Valeur par défaut
fill_style	GCFillStyle	FillSolid
fill_rule	GCFillRule	EvenOddRule
tile	GCTile	pixmap de foreground
stipple	GCStipple	bitmap plein de 1
ts_x_origin	GCTileStipXOrigin	0
ts_y_origin	GCTileStipYOrigin	0

o **Découpage**

Membre	masque	Valeur par défaut
clip_x_origin	GCClipXOrigin	0
clip_y_origin	GCClipYOrigin	0
clip_mask	GCClipMask	None

o **Divers**

Membre	masque	Valeur par défaut
font	GCFont	dépend de l'implém.
subwindow_mode	GCSubwindowMode	ClipByChildren
graphics_exposures	GCGraphicsExposures	True

X-I

Exemples

Fixer la *couleur d'encre* (foreground) et la *couleur de remplissage* (background).

- o A la création

```
GC ctx;  
XGCValues x;  
  
x.foreground = Noir;  
x.background = Blanc;  
ctx =XCreateGC(dpy, fen, GCForeground | GCBackground, &x);
```

- o Par utilitaires :

```
GC ctx;  
ctx =XCreateGC(dpy, fen, 0, NULL);  
XSetForeground(dpy, ctx, Noir);  
XSetBackground(dpy, ctx, Blanc);
```

- o Par changement :

```
GC ctx;  
XGCValues x;  
ctx =XCreateGC(dpy, fen, 0, NULL);  
x.foreground = Noir;  
x.background = Blanc;  
XChangeGC(dpy, ctx, GCForeground | GCBackground, &x);
```

X-I

Gérer les contextes graphiques

- o On *récupère* les valeurs d'un contexte graphique par `XGetGCValues`.
- o Exemple :

```
GC ctx;  
XGCValues x;  
Mask m;  
XGetGCValues(dpy, ctx, m, &x);
```
- o Deux attributs ne peuvent pas être récupérés. Ce sont `dashes` et `clip_mask`.
- o On *copie* des valeurs d'un contexte graphique dans un autre par `XCopyGC`.

- o Exemple:

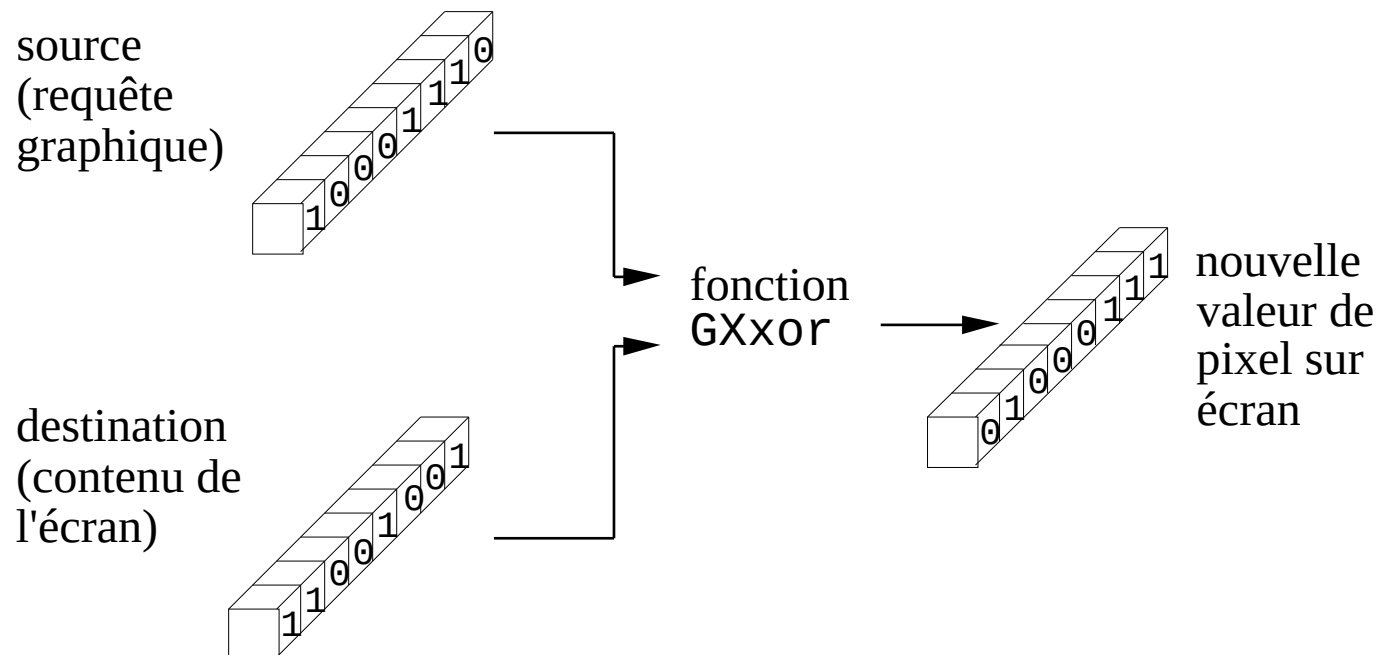
```
GC ctx, cty;  
Mask m;  
  
XCopyGC(dpy, ctx, m, cty);
```

Les attributs copiés sont ceux spécifiés dans le masque m.

X-I

Membre “function”

- o Le membre **function**, de masque **GCFunction**, contrôle la valeur de pixel en fonction de la valeur du pixel *source* et celle du pixel *destination*.
- o Par défaut, le pixel destination n'est pas pris en compte: **GXcopy**.
- o On fixe ce membre à **GXxor** par exemple par
XSetFunction(dpy, ctx, GXxor);
- o Exemple:



X-I

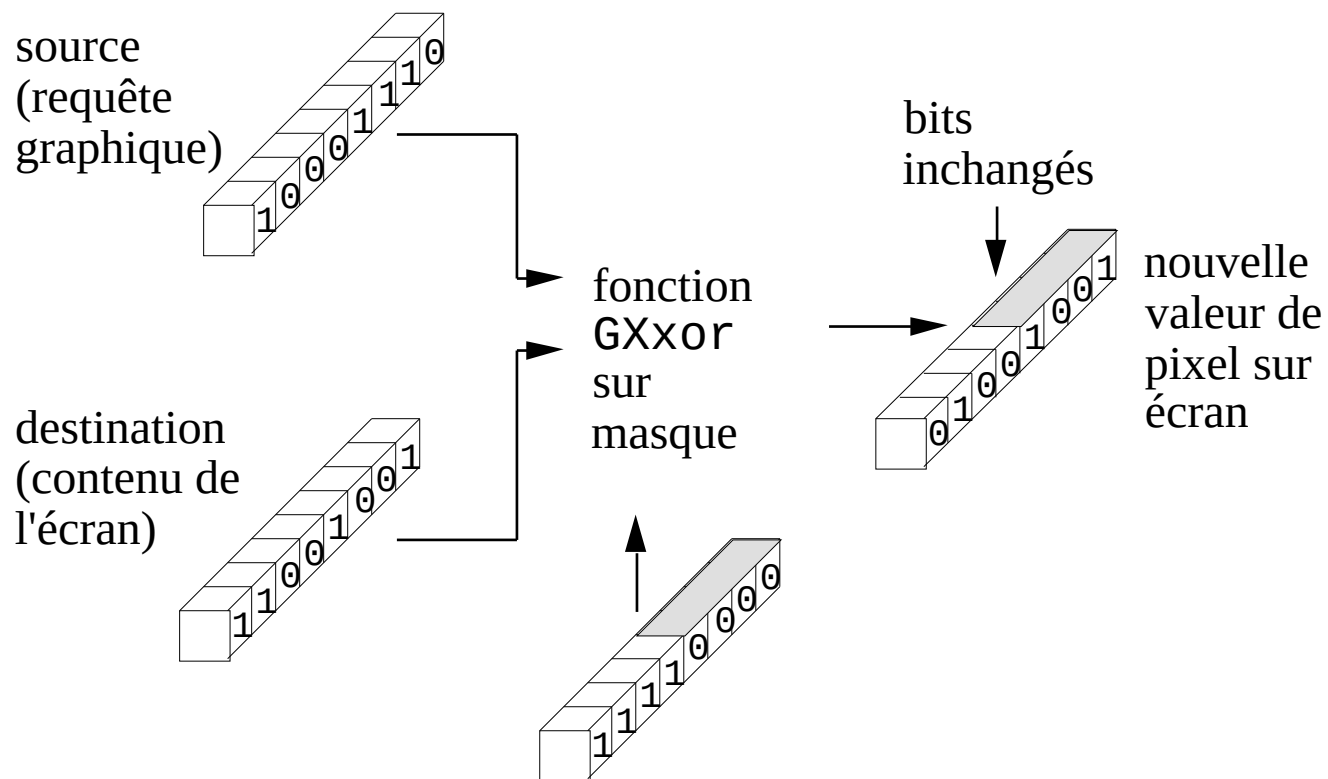
Les 16 fonctions “logiques”

Nom	source dest hexa	0 0 1 1 0 1 0 1	effet
GXclear	0x0	0 0 0 0	/* 0 */
GXand	0x1	0 0 0 1	/* src AND dst */
GXandReverse	0x2	0 0 1 0	/* src AND NOT dst */
GXcopy	0x3	0 0 1 1	/* src */
GXandInverted	0x4	0 1 0 0	/* NOT src AND dst */
GXnoop	0x5	0 1 0 1	/* dst */
GXxor	0x6	0 1 1 0	/* src XOR dst */
GXor	0x7	0 1 1 1	/* src OR dst */
GXnor	0x8	1 0 0 0	/* NOT src AND NOT dst */
GXequiv	0x9	1 0 0 1	/* NOT src XOR dst */
GXinvert	0xa	1 0 1 0	/* NOT dst */
GXorReverse	0xb	1 0 1 1	/* src OR NOT dst */
GXcopyInverted	0xc	1 1 0 0	/* NOT src */
GXorInverted	0xd	1 1 0 1	/* NOT src OR dst */
GXnand	0xe	1 1 1 0	/* NOT src OR NOT dst */
GXset	0xf	1 1 1 1	/* 1 */

X-I

Masque de plan

- o `plane_mask` sert à restreindre l'effet des fonctions “logiques” à une partie des plans de la planche. Très important pour les écrans couleur ou niveaux de gris .
- o Exemple:



X-I

- o **Fonction utilitaire :** `SetPlaneMask(dpy, ctx, masque);`

- o **En particulier :** `SetPlaneMask(dpy, ctx, Noir ^ Blanc);`
restreint l'effet du xor aux bits correspondant aux deux couleurs.

- o **Règle générale pour le calcul du bit *résultat* :**

$$((\text{src FONC dst}) \text{ AND plane_mask}) \text{ OR } (\text{dst AND (NOT plane_mask)})$$

- o **ce qui signifie :**

- o les bits *hors* du masque de plans sont *inchangés* : `dst AND (NOT plane_mask)`

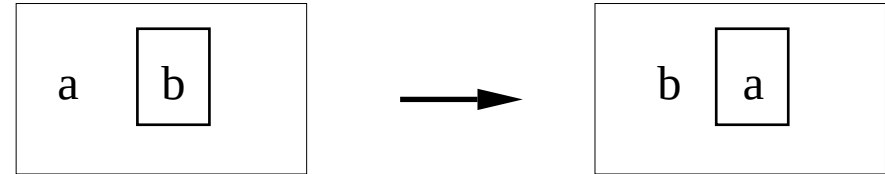
- o les bits *dans* le masque du plan sont calculés avec la fonction logique :

$$(\text{src FONC dst}) \text{ AND plane_mask}$$

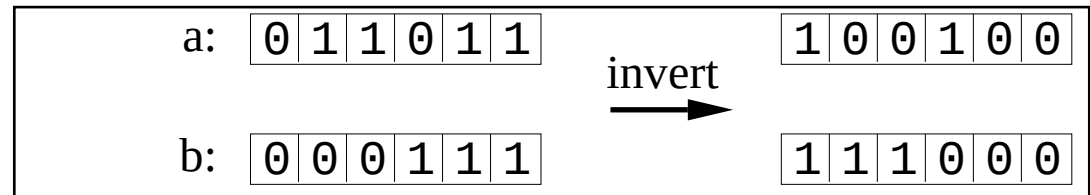
X-I

Un exemple

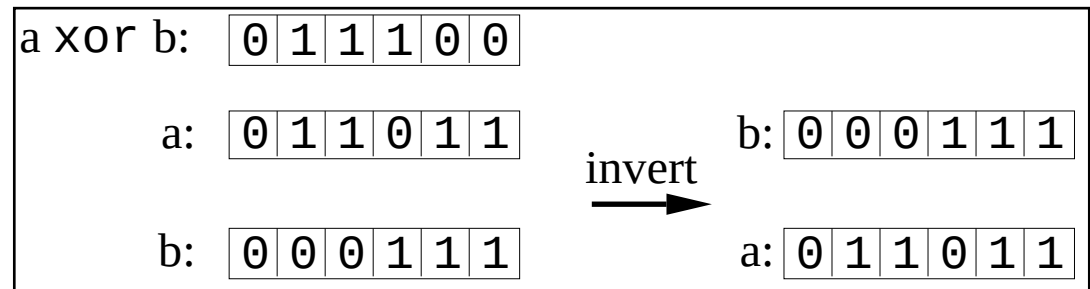
- o Exemple d'utilisation : Echanger deux couleurs a et b :



- o GXinvert ne fait pas l'affaire :



- o GXinvert avec un masque égal à a xor b :



- o Exemple :

```
c = XCreateGC(d, racine, 0, NULL);
XSetPlaneMask(d, c, Noir^Blanc);
XSetFunction(d, c, GXInvert);
XSetForeground(d, c, Noir);
XSetWindowBackground(d, f, Blanc);
XFillRectangle(d, f, c, 0,0, largeur, hauteur);
```

X-I

Attributs de tracés de lignes

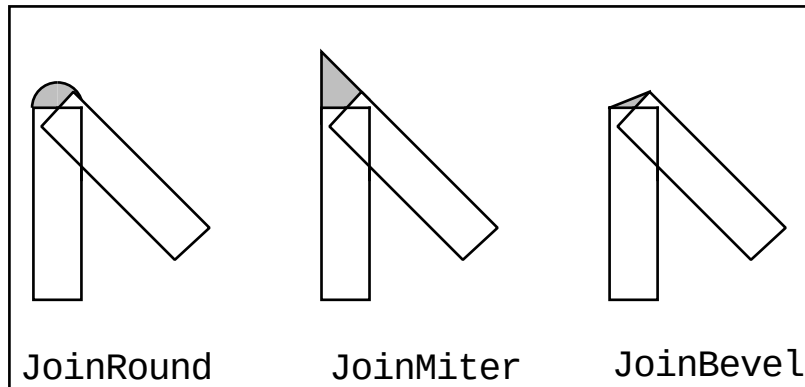
Membre	masque	Valeur par défaut
line_width	GCLineWidth	0
line_style	GCLineStyle	LineSolid
cap_style	GCCapStyle	CapButt
join_style	GCJoinStyle	JoinMiter
dash_offset	GCDashOffset	0
dashes	GCDashList	4

- o **line_width** pour l'*épaisseur* du trait.
 - | **En pixels écran.**
 - | **La valeur 0 autorise la mise en œuvre, par le serveur, de son meilleur algorithme.**
 - | **Il n'existe pas d'utilitaire pour fixer uniquement cette largeur.**
- o **line_style** pour un *pointillé* éventuel.
 - | **valeurs possibles** LineSolid, LineOnOffDash, LineDoubleDash.
- o **cap_style** pour la *terminaison* des lignes.
 - | **valeurs possibles** CapNotLast, CapButt, CapRound, CapProjecting.
- o **join_style** pour la *jonction* de lignes consécutives
 - | **valeurs possibles** JoinRound, JoinMiter et JoinBevel.

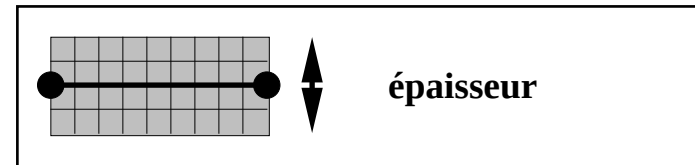
X-I

Illustration des attributs de lignes

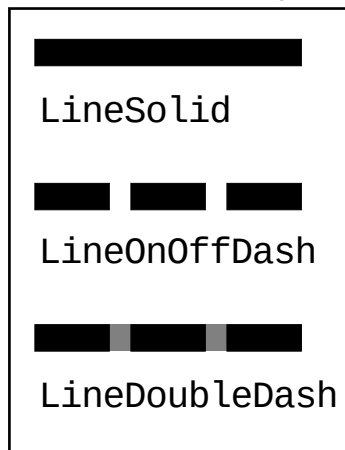
o join_style



o line_width

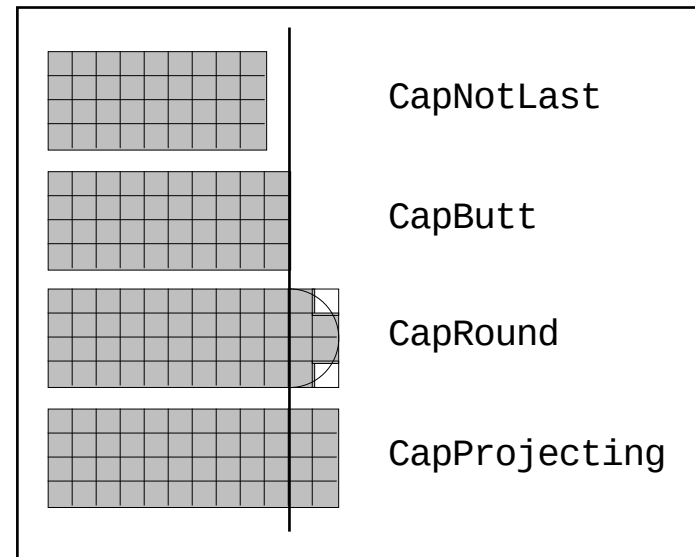


o line_style



o couleur de
remplissage
(background)

o cap_style



X-I

- Se fixent par

```
XSetLineAttributes(dpy, ctx, epaisseur, pointille,
    terminaison, jonction);
unsigned int epaisseur;
int pointille, terminaison, jonction;
```

- Exemple :

```
XSetLineAttributes(dpy, ctx, 2, LineSolid, CapButt,
    JoinBevel);
```

- Pointillés:

```
XSetDashes(dpy, ctx, offset, rythme, longueur);
char rythme[];
int offset, longueur;
```

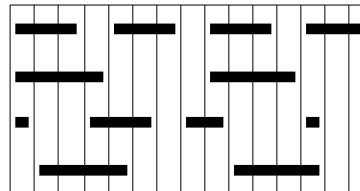
- Exemple :

0 (3,1)

0 (4)

0 (1,2,3)

7 (4)



X-I

Remplissage

Membre	masque	Valeur par défaut
fill_style	GCFillStyle	FillSolid
fill_rule	GCFillRule	EvenOddRule
tile	GCTile	pixmap de foreground
stipple	GCStipple	bitmap plein de 1
ts_x_origin	GCTileStipXOrigin	0
ts_y_origin	GCTileStipYOrigin	0

- o **fill_style pour le remplissage, éventuellement par les motifs ou pavés:**
 - | **FillSolid** : pour remplissage avec couleur d'encre
 - | **FillTiled** : pour remplissage par pavés (un *pixmap*)
 - | **FillStippled et FillOpaqueStippled** : pour remplissage par filigrane (“stipple”, un *bitmap*).
- o **ts_x_origin, ts_y_origin**
 - | **sont les coordonnées du décalage du motif.**

(Un bitmap est un pixmap de profondeur 1)

X-I

Utilitaires

XSetTile (dpy, ctx, p)

- o Le pixmap p doit avoir la profondeur de la planche dans laquelle il sera utilisé.

XSetStipple(dpy, ctx, p)

- o Le pixmap p doit avoir profondeur 1.

XSetTSOrigin(dpy, ctx, x_o, y_o)

XSetFillStyle(dpy, ctx, style)

- o style peut être
 - | FillSolid,
 - | FillTiled,
 - | FillStippled,
 - | FillOpaqueStippled

XSetFillRule(dpy, ctx, regle)

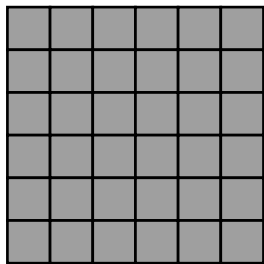
- o regle peut être
 - | WindingRule ou
 - | EvenOddRule.
- o n'intervient que pour le remplissage de polygones.

X-I

-  pixel non modifié
-  pixel de foreground
-  pixel de background

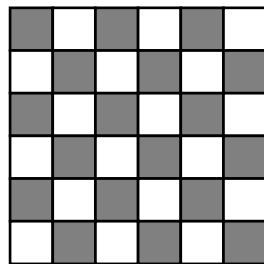
Exemple

- o fill_style: 4 valeurs possibles



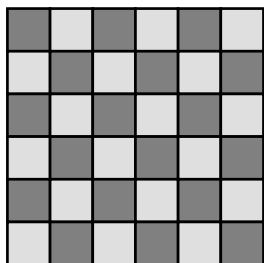
FillSolid: couleur unie, sans mystère

FillTiled: répétition d'un motif de type Pixmap
(profondeur de la planche)



1	0	1
0	1	0
1	0	1

FillStippled: pochoir ou filigrane spécifiant les pixels
laissés invariants et ceux qui sont peints avec la couleur
foreground (c'est un Pixmap à un seul plan)

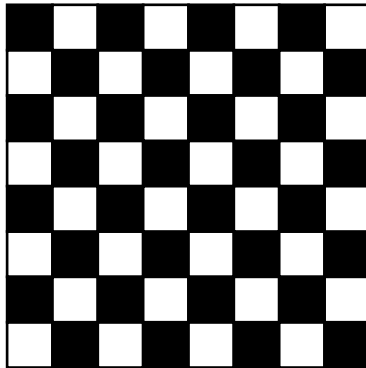


FillOpaqueStipple: définit les pixels peints avec la
couleur foreground et ceux peints avec background

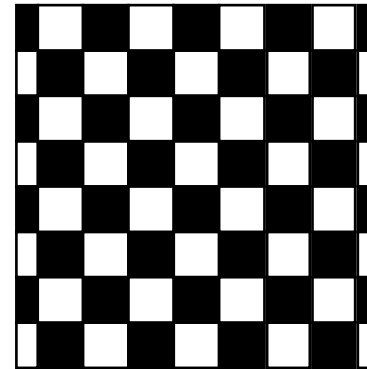
X-I

Décalage du motif ou du filigrane

Décalage du filigrane ou du motif par rapport aux abscisses et aux ordonnées
`XSetTSOrigin(dpy, ctx, ts_x_origin, ts_y_origin);`



`ts_x_origin = 0`
`ts_y_origin = 0`



`ts_x_origin = 2`
`ts_y_origin = 0`

X-I

Découpage (clipping)

- o La zone de découpage par défaut est la fenêtre tout entière.
 - o On peut spécifier une autre zone de découpage, en définissant le pixmap `clip_mask` à l'aide d'une ou plusieurs des fonctions:
 `XSetClipMask, XSetClipRectangles`
 `XSetRegion, XSetClipOrigin`
 - o On utilise une zone de découpage pour ne redessiner que les parties exposées dans un dessin complexe. Il y a des optimisations internes pour accélérer le dessin.
-

`XSetClipMask(dpy, ctx, p)` où `p` est un pixmap de profondeur 1

`XSetClipRectangles(dpy, ctx, xorig, yorig, r, nr, ordre)`
 `XRectangle *r;`
 `int ordre;`
 | où `ordre` indique si les rectangles dans le tableau `r` sont triés, et si oui comment.

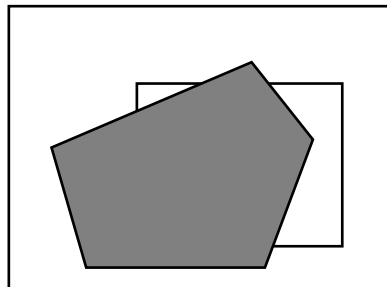
`XSetRegion(dpy, ctx, region)`
 `XRegion region;`
 | où `XRegion` est un type opaque qui permet de gérer de façon efficace des zones de bits dans un rectangle.

X-I

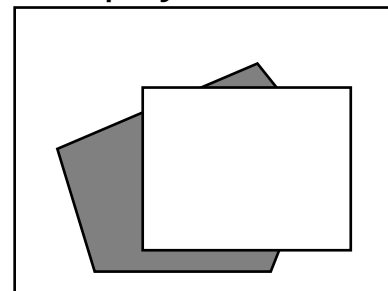
Que faire des fenêtres descendantes?

- Par défaut (ClipByChildren) les pixels d'une fenêtre cible recouverts par les fenêtres filles ne sont pas affectés par la requête de dessin
- On peut dessiner en ignorant les frontières des fenêtres filles (IncludeInferiors)

IncludeInferiors



ClipByChildren



X-I

Pixmaps et bitmaps

- o Un *pixmap* est un tableau rectangulaire de valeurs de pixels.
- o Un pixmap a une *largeur*, une *hauteur*, et une *profondeur*.
- o Un pixmap est stocké du côté du *serveur* (dans l'écran concerné), et est repéré par un entier (type `Pixmap`)
- o Un *bitmap* est un pixmap de profondeur 1. Il n'y a pas de type X pour les bitmaps.
- o Il existe une convention X pour *stocker un bitmap* sous forme d'un fichier ASCII. Une telle convention n'existe pas pour les pixmaps.
- o On peut dessiner dans un pixmap comme dans une fenêtre. La réunion des deux forme les “drawables” (planche).
- o Les pixmaps et bitmaps sont utiles :
 - | pour les icônes
 - | pour les boutons
 - | les pavages (“tiling”, pixmap) et les filigranes (“stipple”, bitmap)

X-I

Bitmaps

- o Le programme `bitmap` permet de dessiner facilement des bitmaps, stockés sous forme de fichier ASCII.
- o Ce fichier ASCII comporte :
 - | la hauteur
 - | la largeur
 - | le point de référence (deux entiers, facultatif sert pour les curseurs)
 - | un tableau de caractères représentant le dessin.
- o De plus, des conventions de nom en facilitent la lecture.

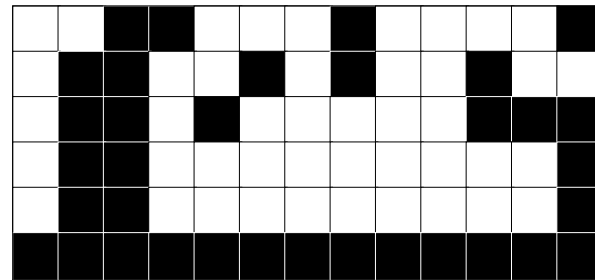
- o **Fichier** `13x6.btm`

```
#define 13x6_width 13
#define 13x6_height 6
static char 13x6_bits[] = {
    0x8c, 0x10, 0xa6, 0x04,
    0x16, 0x1c, 0x06, 0x10,
    0x06, 0x10, 0xff, 0x1f};
```

X-I

codage des bitmaps

codage LSBFirst



c		8		0		1		13x6_bits[] = { 0x8c, 0x10, 0xa6, 0x04, 0x16, 0x1c, 0x06, 0x10, 0x06, 0x10, 0xff, 0x1f};
6		a		4		0		
6		1		c		1		
6		0		0		1		
6		0		0		1		
f		f		f		1		

X-I

Création de pixmaps

- o La création d'un pixmap ne le remplit pas. Deux façons :
 - | Création ex nihilo, pour y copier ensuite de fenêtres, ou préparer des dessins;
 - | Création et remplissage à partir d'un tableau ou d'un fichier.

- o Ex nihilo : Création d'un pixmap de profondeur donnée :

```
Pixmap XCreatePixmap(dpy, d, largeur, hauteur, profondeur);
```

 - | avec Drawable d.
 - | Toutes les profondeurs ne sont pas autorisées. Deux profondeurs au moins : 1 et `DefaultDepth(dpy, ecran)`.
 - | La planche d ne sert qu'à déterminer l'écran qui stocke le pixmap.
 - | Exemple :

```
p = XCreatePixmap(dpy, DefaultRootWindow(dpy), largeur, hauteur, DefaultDepth(dpy, ecran));
```
- o Libération de pixmaps (les pixmaps sont une ressource précieuse):

```
XFreePixmap(dpy, p)
```

X-I

Création à partir d'un tableau d'octets

- o **Création d'un pixmap à partir d'un tableau d'octets :**

`Pixmap XCreatePixmapFromBitmapData(dpy, d,`
 du pixmap
 couleur_1, couleur_0, **couleurs pour les 1 et les 0**
 profondeur) **du pixmap**

- o **Exemple :**

```
#include "xi.bitmap"
p = XCreatePixmapFromBitmapData(dpy, DefaultRootWindow(dpy),
    xi_bits, xi_width, xi_height, rouge, vert, DefaultDepth(dpy,
    ecran));
```

- o **Création d'un bitmap (i.e. pixmap de profondeur 1):**

`Pixmap XCreateBitmapFromBitmapData(dpy, d,`
 tableau, largeur, hauteur)

X-I

Lecture d'un fichier bitmap

- o Un bitmap (i. e. un pixmap de profondeur 1) peut être créé à partir d'un fichier ASCII conforme aux conventions, par :

```
XReadBitmapFile(dpy, d, fichier, largeur, hauteur, bm, x_hot, y_hot)  
    Drawable d;  
    char *fichier;  
    unsigned int *largeur, *hauteur;  
    Pixmap *bm;  
    int *x_hot, *y_hot;
```

- o Exemple (ah.c):

```
XReadBitmapFile(dpy, DefaultRootWindow(dpy), "xi.btm", &k,&k, &fond, &k, &k)
```

X-I

Effacement et copie de zones

- o Effacer le contenu d'une fenêtre signifie redessiner le fond de la fenêtre avec la couleur de fond *de la fenêtre*. Ceci se fait bien entendu *sans* contexte graphique.
- o Deux fonctions :
 - `XCclearWindow(dpy, fen)`
 - | remplit la fenêtre avec sa couleur de fond.
 - `XCclearArea(dpy, fen, x, y, largeur, hauteur, exposition)`
`Bool exposition;`
 - | remplit le rectangle spécifié avec le fond;
 - | `exposition` est `True` ou `False`, et indique si un évènement de type `Expose` doit être engendré.
- o NB.: Un pixmap ne peut pas être effacé de cette manière, car il n'a pas de fond !
- o NB.: Ne pas confondre avec un `XFillRectangle` qui ferait appel à la couleur de fond (de *remplissage*) d'un contexte graphique.

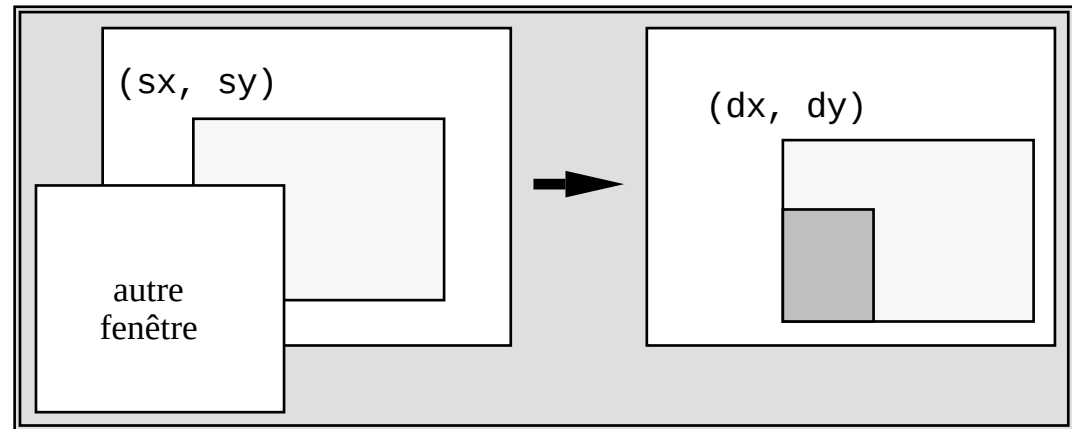
X-I

Copie de zones

XCopArea(dpy, source, dest, ctx, sx, sy, largeur, hauteur, dx, dy)

Drawable source, dest;
int sx, sy, dx, dy;

- o **Copie le rectangle de source spécifié par sx, sy, largeur, hauteur dans dest, au coin supérieur gauche dx, dy.**

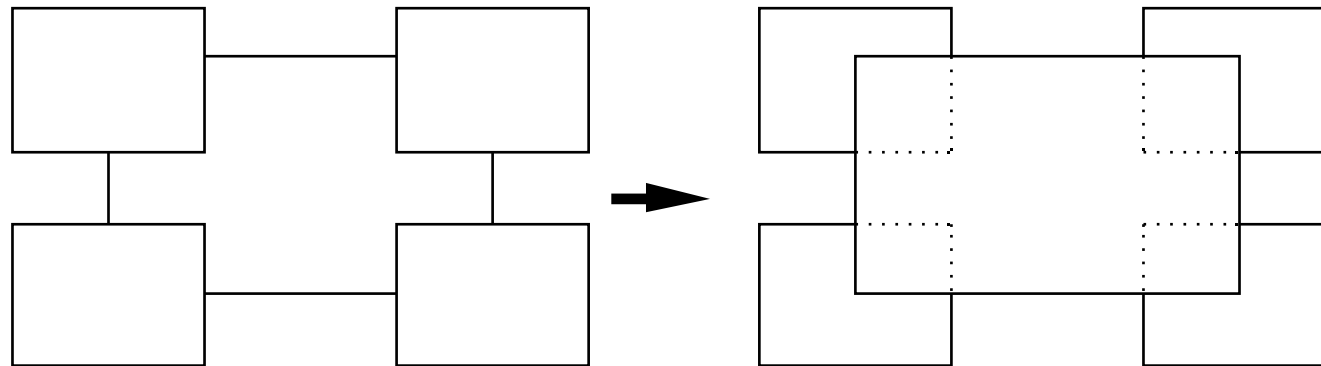


- o **Si une partie du rectangle dans source est cachée, elle n'est pas copiée, et les parties correspondantes de dest sont remplies avec la couleur de fond de la fenêtre de destination.**
- o **Des événements non masquables sont envoyés pour signaler le résultat. Cela dépend d'un champ du contexte graphique :**
 - l Si graphics_exposures est True, des événements
 - GraphicsExpose sont engendrés s'il y a des parties couvertes;
 - NoExpose dans le cas contraire.
 - l Si graphics_exposures est False, rien n'est engendré.

X-I

Evènement Expose

- Un évènement Expose est engendré lorsqu'une fenêtre devient visible ou lorsqu'une partie cachée devient visible.
- Un bon style de programmation groupe les fonctions de dessin comme réponse à la réception d'un évènement Expose.
- Une seule action peut engendrer plusieurs évènements Expose sur une même fenêtre.



Quatre évènements Expose

X-I

- o Une fonction de dessin dans une fenêtre invisible est perdue.
- o En particulier, il est formellement déconseillé d'enchaîner:

```
XMapWindow(dpy, fen);  
XDraw...
```

sur les fenêtres principales, car à cause de l'interaction avec le gestionnaire de fenêtre, le serveur ne garantit pas que la fenêtre est “mappée” (sur requête du gestionnaire de fenêtres !) avant que la requête de dessin (du client !) soit honorée.

- o On peut utiliser aussi la *lecture bloquante* des évènements pour forcer l'attente:

```
XSelectInput(dpy, fen, ExposureMask);  
XMapWindow(dpy, fen);  
XNextEvent(dpy, &evmt);  
XDraw...
```

- o Les requêtes sont envoyées par nécessité: on force l'envoi des requêtes par

```
XFlush(dpy);
```

qui vide la file des requêtes.