Nuevas Tecnologías de la Programación

Módulo 1: Programación gráfica en entorno UNIX

Andrés Cano Utrera
Dpto. Ciencias de la Computación e I.A
Universidad de Granada

Septiembre de 2011

Indice

1. Introducción al Sistema X Window

Fundamentalmente se ha utilizado la siguiente bibliografía:

- N.Adrian. Xlib Programming Manual for Version 11. X Window System. Vol 1. O'Reilly and ASS, 1992.
  Disponible en biblioteca ETSII y en
  `http://www.sbin.org/doc/Xlib/`.
- N.Adrian. Xlib Reference Manual for Version 11. X Window System. Vol 2. O'Reilly and ASS, 1992.

1.1. Introducción

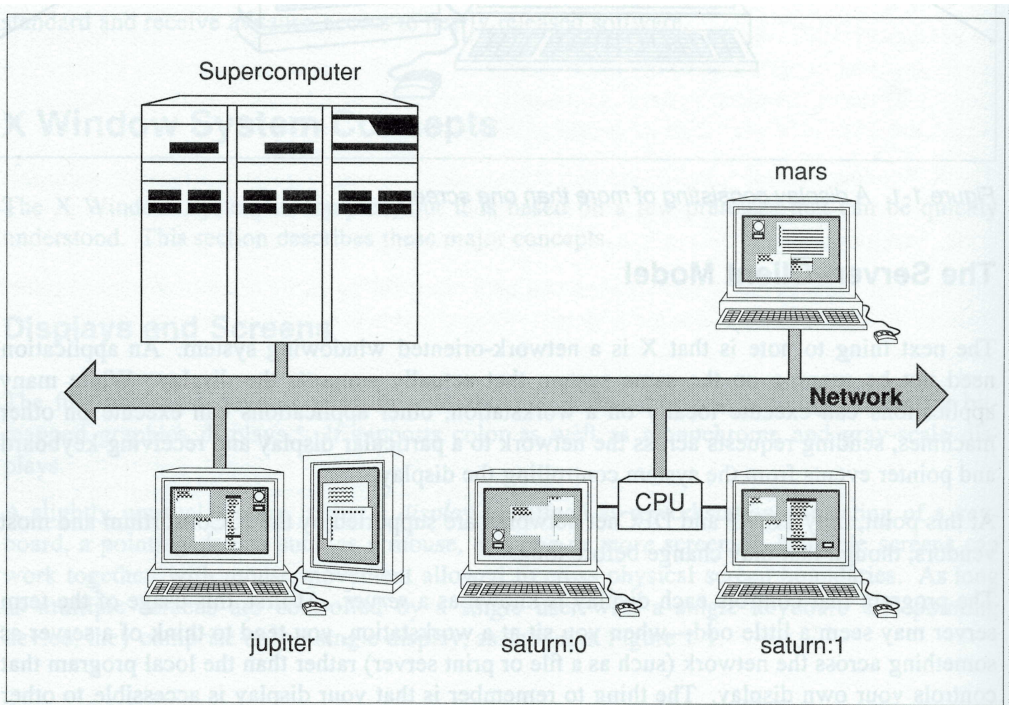
- X (X Window System) es un sistema de ventanas basado en **red** que puede funcionar en sistemas de **diferentes** fabricantes.
- Fue desarrollado por el MIT y DEC, tomando como base el sistema W de Stanford.
- Es un sistema independiente de:
  - Dispositivo gráfico.
  - Arquitectura de la máquina.
  - Sistema operativo.
  - Red y protocolo de conexión.
- Nosotros estudiaremos la biblioteca X (Xlib), que es el interfaz de programación para lenguaje C de la versión 11 del X Window System.
- Xlib es el interfaz de programación de más bajo nivel para X.

1.2. Display y screen

- Un display es una estación de trabajo consistente de un teclado, un dispositivo apuntador (ratón) y una o más pantallas (screens).
- X es un sistema de ventanas para displays basados en gráficos bitmap (cada punto de pantalla es uno o más bits en memoria).
- X soporta tanto displays en color como en monocromo.

1.3. Modelo cliente servidor

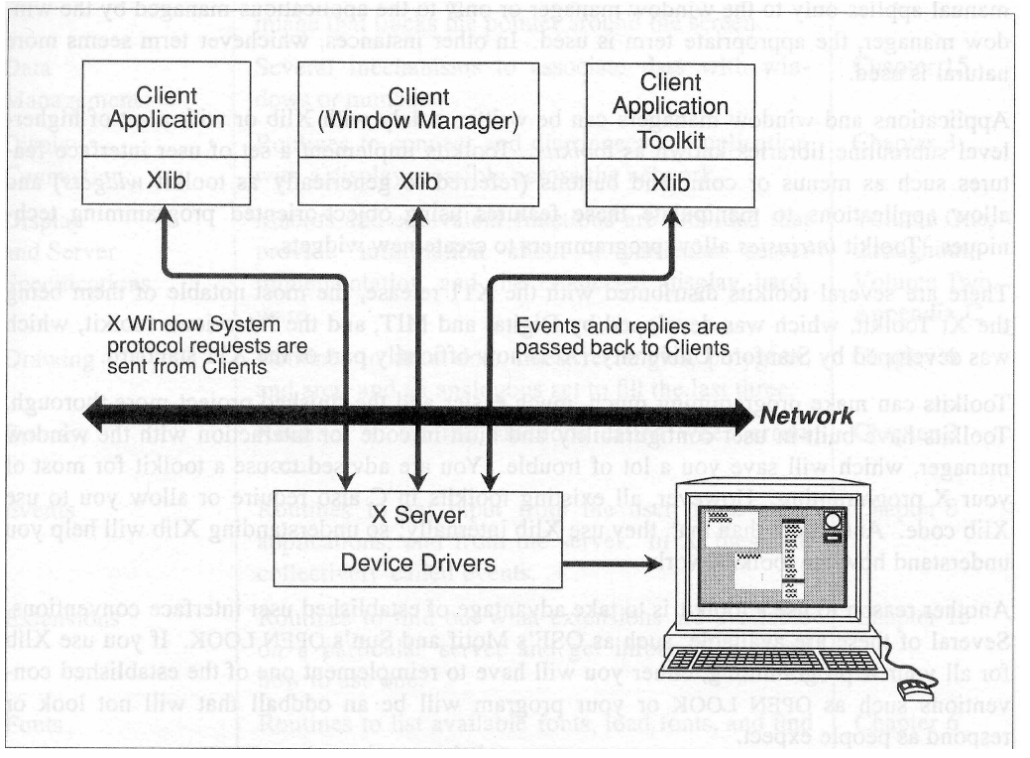
- Un programa en X puede ejecutarse en una máquina remota y visualizarse en la máquina local, aunque disponga de hardware y S.O. diferente.



- El **servidor** es el programa que controla cada display.
- Los programas de usuario locales o remotos son los **clientes**.
- Los programas cliente se construyen con la biblioteca `Xlib` que está escrita en `C`.
- Las funciones del servidor son:
  - Permite acceder al display a los clientes.
  - Interpretar los mensajes que los clientes le envían por la red.
  - Enviar información a los clientes (previa solicitud de éstos).
  - Hacer dibujos bidimensionales.
  - Mantener estructuras de datos complejas (ventanas, cursores, fonts, contexto gráfico, mapas de color, pixmap) como recursos que pueden ser compartidos entre clientes.
- Un X-terminal es un ordenador que ejecuta únicamente un servidor X.

1.4. Toolkits

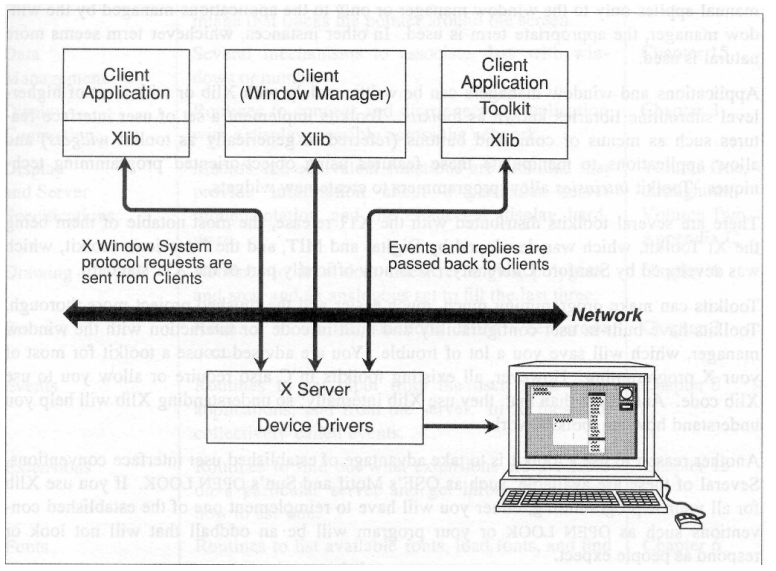
- Una aplicación puede construirse únicamente con la biblioteca `Xlib` o bien con bibliotecas de más alto nivel (*toolkits*) que permiten: (ver figura)
  - Programación más sencilla.
  - Fácil construcción de programas que sigan las convenciones de una determinada interfaz de usuario.



- Ejemplos de *toolkits*: Xview, Motif, Xforms, Qt.

1.5. Manejador de ventanas

- Las aplicaciones cliente no controlan directamente el aspecto y tamaño de las ventanas principales. Sólo dan *hints* (instrucciones) para indicar tamaño y posición.
- Estas cosas las realiza el *manejador de ventanas*.



- Es un cliente que tiene privilegios especiales:
  - Controla la apariencia de las ventanas y el estilo de interacción con el usuario.
  - Pone a la ventana principal de la aplicación una barra de títulos junto con ciertos botones (iconizar, maximizar, salir ...).
  - Permite mover y redimensionar las ventanas de más alto nivel de una aplicación.
  - Comenzar nuevas aplicaciones a través de unos menús predefinidos.
  - Controla el apilamiento de las ventanas según una determinada *política de capas*.

1.6. Eventos

- Un evento es un paquete de información generado por el servidor en respuesta a ciertas acciones:
  - Entradas de usuario: Pulsación de una tecla, click de ratón, movimiento de ratón, etc.
  - Interacción con otros programas: Expose.
- Los eventos se guardan en una cola y los clientes normalmente los procesan uno a uno en el orden de llegada.

1.7. Protocolo X

- Es la forma de comunicar el cliente y servidor, aunque ambos se ejecuten en la misma máquina.
- Existen 4 tipos de paquetes que pueden transferirse con este protocolo:
  - Peticiones (request): Las hace el cliente al servidor para solicitar algo (dibujar una línea, preguntar por el tamaño de una ventana, etc).
    La mayoría de las rutinas Xlib son peticiones.
  - Réplicas (reply): Enviadas del servidor al cliente en respuesta a ciertas peticiones.
  - Eventos (event): Enviados del servidor al cliente. Los clientes deben especificar para cada ventana que creen, qué eventos quieren recibir.
  - Errores (error): Enviados por el servidor cuando una petición previa fue inválida.

1.8. Buffering

- Xlib agrupa (encola) las peticiones: el cliente puede continuar ejecutándose ya que la mayoría de las llamadas a Xlib no requieren respuesta inmediata.
- Xlib envía el buffer completo bajo 3 condiciones:
  - Cuando una aplicación llama a una rutina Xlib que espera un evento y la cola de eventos no contiene ningún evento del tipo solicitado.
  - Cuando la llamada requiere una información del servidor de forma inmediata (rutinas `Query`, `Fetch` o `Get`).
  - El cliente envía el buffer manualmente con `XFlush(Display *d)` o `XSync(Display *d,Bool discardEvents)`.

1.9. Recursos e identificadores de recursos

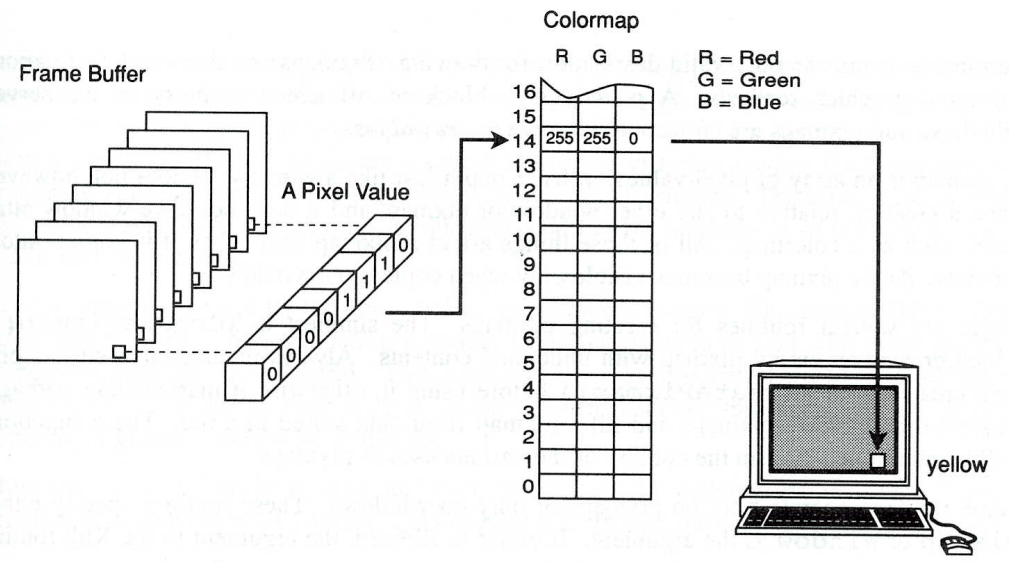
- Los **recursos** son las áreas en las que se almacena información de los servidores.
- Cada recurso tiene un número que lo identifica, mediante el cual los clientes acceden a él.
- Los recursos están almacenados en el servidor: los recursos no son enviados entre cliente y servidor (esto incrementa el rendimiento de la red)
- Recursos básicos:
  - **Ventanas**: Area rectangular mostrada en la pantalla.
  - **Pixmaps**: Area rectangular en la que se pueden dibujar gráficos.
  - **GCs**: Contexto gráfico en el que se almacenan los atributos de los gráficos.
  - **Cursores**: Símbolo que muestra la posición actual del puntero.
  - **Fuentes**: Almacena la forma de cada carácter.
  - **Colormaps**: Mapas o paletas de colores.

1.10. Propiedades y átomos

- Una **propiedad** es un paquete de información asociado con una ventana, que está disponible para todos los clientes que se ejecutan en el mismo servidor.
- Las propiedades son usadas por los clientes para almacenar información que otros clientes pueden necesitar y para leer información puesta por otros clientes.
- Cada propiedad tiene asociado un identificador alfabético (string) y un identificador numérico llamado **átomo**.
  Ejemplo: `WM_COLORMAP_WINDOWS`
- El programa `xprop` nos da las propiedades de una ventana.
- El átomo se obtiene con la función
  ```
  Atom XInternAtom(Display *d, char *propiedad,
  Bool only_if_exists)
  ```
- Hay átomos *predefinidos*, usados cuando se inicializa el servidor.
  Estos átomos están disponibles con constantes simbólicas que comienzan por `XA_`, con lo que no es necesario usar `XInternAtom()`.
  Estos átomos corresponden a propiedades cuyo contenido tiene cierto significado conocido por todos los clientes.
- Uno de los usos más importantes de las propiedades es para comunicar información desde las aplicaciones al window manager y viceversa.
  Ejemplo: Los clientes deben especificar en su ventana principal las propiedades estándar (por ej. rango de tamaños que prefiere). El window manager también especifica que tamaños de iconos prefiere.

1.11. Especificación de color

- **Valores de píxel y mapas de colores**
  - Pantalla del display: colección de píxels.
  - Cada posición de píxel almacena un valor que representa un color específico (índice en una tabla llamada **colormap**) (ver figura)



  - En displays a color la tabla tiene 3 entradas o colorcells (rojo, verde y azul).
  - Los clientes pueden compartir el mismo colormap, aunque cada uno puede definir el suyo propio si el colormap es escribible (writable).
- **Mapa de colores por defecto**:
  `Colormap DefaultColormap(Display *d,int n_screen)`
- **Píxeles y planos**
  - Planos: número de bits por pixel.
  - X11 soporta hasta 32 planos.
  - `int BlackPixel(Display *d,int n_screen)` y `int WhitePixel(Display *display,int n_screen)` devuelven el valor para el negro y blanco en el colormap por defecto.
  - Con n planos de color podemos ver simultaneamente 2^n colores distintos.

1.12. Pixmaps y Drawables

- **Pixmap**: Array de valores de pixel
  Tiene profundidad, pero no posición ni atributos de ventana.
  Se hace visible al copiarse a una ventana.
- Windows y Pixmaps son conocidos como **Drawables**.
- `XCopyArea` copia un pixmap a una ventana que debe ser de la misma profundidad.
  `XCopyArea(Display *d, Drawable src, Drawable dst, GC gc,`
  `int src_x, int src_y, unsigned int width,unsigned int height,`
  `int dest_x, int dest_y)`
- `XCopyPlane` copia un plano de un pixmap de cualquier profundidad a una ventana.
  `XCopyPlane(Display *d, Drawable src, Drawable dst, GC gc,`
  `int src_x, int src_y, unsigned int width,unsigned int height,`
  `int dest_x, int dest_y, unsigned long plane)`
- Los pixmaps se suelen usar para salvar los contenidos de las ventanas.
- Un pixmap de profundidad 1 es un `bitmap`.
  Son usados para definir fonts, cursores.

1.13. Tiles y Stipples

- **Tile**: Pixmap usado como patrón de relleno, con la misma profundidad del drawable donde será usado.
  Normalmente es de tamaño 16×16 y tiene 2 valores diferentes de pixel.
- **Stipple**: Pixmap de profundidad 1
  Es usado junto con un valor de pixel para el foreground y algunas veces un valor de background, para rellenar un área.

1.14. Contexto gráfico

- X dispone de rutinas para dibujar en ventanas y pixmaps puntos, líneas, rectángulos, polígonos, arcos, texto, etc.
- Todas estas rutinas tienen como parámetro un **contexto gráfico**.
- El **contexto gráfico** es un recurso del servidor que especifica las características con las que se dibujará en el drawable: grosor de línea, color, patrón de rellenado de polígonos, etc.
- Una aplicación puede usar varios GCs.

2. Estructura de un programa básico

2.1. Compilación del programa

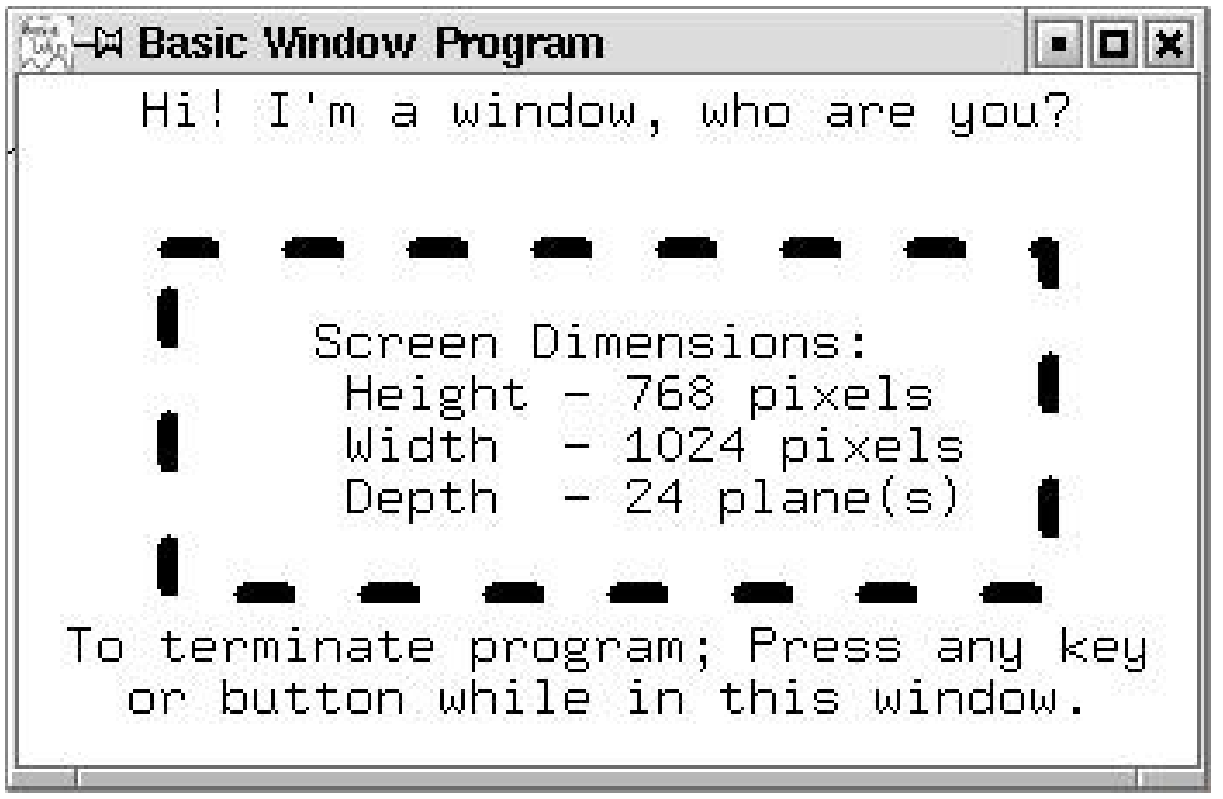
- Compilar y linkar al mismo tiempo:

```
gcc -o outputfile inputfile.c -lX11 -L/usr/X11R6/lib
                                    -I/usr/X11R6/include
```

- Compilar y linkar por separado:

```
gcc -c -o outputfile.o inputfile.c -I/usr/X11R6/include
gcc -o outputfile inputfile.o -lX11 -L/usr/X11R6/lib
```

2.2. Pasos básicos de un programa escrito en Xlib



- Inclusión de ficheros de cabecera

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>
```

- Conexión al servidor X con

```
Display *XOpenDisplay(char *display_name)
```

Suele usarse `XOpenDisplay(host:server.screen)` o `XOpenDisplay(NULL)`

- Obtención de información de la pantalla para calcular el tamaño de la ventana principal:

```
int DisplayWidth(Display *d,int screen_number)
int DisplayHeight(Display *d,int screen_number)
int DisplayWidthMM(Display *d,int screen_number)
int DisplayHeightMM(Display *d,int screen_number)
XGetGeometry(Display *d,Drawable w,...)
XGetWindowAttributes(Display *d,Drawable w,...)
Window RootWindow(Display *d, int screen_number)
```

- Crear las ventanas de la aplicación:

```
Window XCreateWindow(display, parent, x, y, width, height,
       border_width, depth, class, visual, valuemask, attributes)

Window XCreateSimpleWindow(display, parent, x, y, width, height,
       border_width, border, background)
```

Los tipos de los parámetros son:

```
Display *display;
Window parent;
int x, y;
unsigned int width, height;
unsigned int border_width;
int depth;
unsigned int class;
Visual *visual
unsigned long valuemask;
XSetWindowAttributes *attributes;
unsigned long border;
unsigned long background;
```

- Crear el icono de la aplicación:

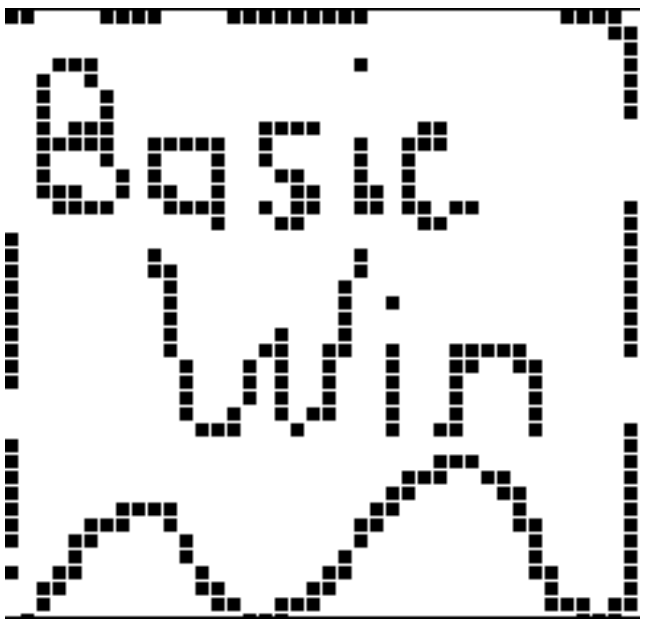
```
Pixmap XCreateBitmapFromData(Display *d,Drawable w,char *data,
                        unsigned int width, int heidht)
int XCreateBitmapFile(...)
int XReadBitmapFile(...)
XGetIconSizes(...)
```

Los iconos pueden dibujarse por ejemplo con el programa `bitmap`:

```
bitmap icon_bitmap 40x40
```

Ejemplo de un fichero de icono:

```
#define icon_bitmap_width 40
#define icon_bitmap_height 40
static char icon_bitmap_bits[] = {
   0xc3, 0xc3, 0x7f, 0x00, 0x78, 0x00, 0x00, 0x00, 0x00, 0xc0, 0x00, 0x00,
   0x00, 0x00, 0x80, 0x38, 0x00, 0x40, 0x00, 0x80, 0x24, 0x00, 0x00, 0x00,
   0x80, 0x44, 0x00, 0x00, 0x00, 0x80, 0x44, 0x00, 0x00, 0x00, 0x80, 0x74,
0x00, 0x0f, 0x0c, 0x00, 0x7c, 0x3e, 0x41, 0x0e, 0x00, 0x44, 0x22, 0x41,
0x02, 0x00, 0x84, 0x22, 0x46, 0x02, 0x00, 0x9c, 0x26, 0xcc, 0x02, 0x00,
0x78, 0x3c, 0xcd, 0x36, 0x80, 0x00, 0x20, 0x06, 0x0c, 0x80, 0x01, 0x00,
0x00, 0x00, 0x80, 0x01, 0x02, 0x40, 0x00, 0x80, 0x01, 0x06, 0x40, 0x00,
0x80, 0x01, 0x04, 0x20, 0x00, 0x80, 0x01, 0x04, 0x20, 0x01, 0x80, 0x01,
0x04, 0x20, 0x00, 0x80, 0x01, 0x04, 0x22, 0x00, 0x80, 0x01, 0x04, 0x33,
0xf1, 0x81, 0x01, 0x88, 0x12, 0x31, 0x03, 0x01, 0x88, 0x12, 0x11, 0x02,
0x00, 0x88, 0x12, 0x11, 0x02, 0x00, 0x48, 0x1a, 0x11, 0x02, 0x00, 0x70,
0x04, 0x19, 0x82, 0x01, 0x00, 0x00, 0x00, 0x80, 0x01, 0x00, 0x00, 0x38,
0x80, 0x01, 0x00, 0x00, 0xce, 0x80, 0x01, 0x00, 0x00, 0x83, 0x81, 0x81,
0x07, 0x80, 0x01, 0x81, 0xe1, 0x04, 0xc0, 0x00, 0x83, 0x31, 0x08, 0x40,
0x00, 0x82, 0x10, 0x08, 0x20, 0x00, 0x82, 0x19, 0x10, 0x30, 0x00, 0x86,
0x0c, 0x30, 0x18, 0x00, 0x84, 0x04, 0x60, 0x0e, 0x00, 0xdc, 0x02, 0x80,
0x03, 0x00, 0x70, 0x00, 0x00, 0x00, 0x00, 0x00};
```


Basic
win

- Activar las propiedades estándar para el WM
  - Una propiedad es un conjunto de información de lectura y escritura para el cliente y usada normalmente para comunicación entre aplicaciones.
  - Son asociadas a la ventana principal para comunicarse con el WM.
  - Un **hint** (instrucción) es un tipo de propiedad que puede que el WM no tenga en cuenta.
  - Propiedades mínimas que deberían especificarse:
    - Nombre de ventana: `WM_NAME` con `XSetWMName`
    - Nombre de icono: `WM_ICON_NAME` con `XSetWMIconName`
    - Nombre de comando y argumentos: `WM_COMMAND` con `XSetCommand`
    - Número de argumentos
    - Datos del XWMHints:
      - ◇ Estado inicial: Normal o iconizado.
      - ◇ Si la aplicación usa o no entrada de teclado.
      - ◇ Pixmap del icono.
    - Tamaños preferidos (configuración) de ventana: `WM_NORMAL_HINTS`. Para ello usaremos un struct `XSizeHints`
  - Datos del `XClassHints`: `WM_CLASS` con `XSetClassHint`
  - Funciones usadas en basicwin:

    `XSetWMProperties(...)`
    `XStringListToTextProperty(...)`

- Seleccionar los eventos para cada ventana:

  `XSelectInput(...)`
  `XCreateWindow(...)`
  `XChangeWindowAtributes(...)`

- Creación de los recursos necesarios:

  `XLoadQueryFont(...)`
  `XSetFont(...)`
  `XCreateGC(...)`

- Mapear cada ventana:

  `XMapWindow()`

- Construir el bucle de eventos, leyendo eventos con:

  `XNextEvent(Display *display, XEvent *evento)`

  - `Expose`: Contenidos de ventana deben repintarse.
  - `ConfigureNotify`: Cambio en la configuración de la ventana.
  - `KeyPress`: Pulsación de tecla
  - `ButtonPress`: Pulsación de botón de ratón.

- Repintar contenidos de ventanas con el evento `Expose`.
  - Redibujar todos los contenidos. (pero sólo cuando count==0).
  - Redibujar sólo las partes dañadas (expuestas): puede ser complicado.
  - Redibujar sólo las partes dañadas usando el `clip_mask` del GC.
  - Dibujar en un pixmap y luego copiarlo cada vez que sea necesario.
  - Mantener el **backing store** a on: sólo en estaciones de altas prestaciones.

- Acabar el programa: liberar recursos y cerrar el display

  `XUnloadFont()`
  `XFreeGC()`
  `XCloseDisplay()`

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>
#include <stdio.h>
#include "../bitmaps/icon_bitmap"

#define BITMAPDEPTH 1
#define TOO_SMALL 0
#define BIG_ENOUGH 1

Display *display;
int screen_num;
static char *progname; /* name this program was invoked by */

void main(int argc, char **argv)
{
  Window win;
  unsigned int width, height; /* window size */
  int x, y;  /* window position */
  unsigned int border_width = 4; /* four pixels */
  unsigned int display_width, display_height;
  unsigned int icon_width, icon_height;
  char *window_name = "Basic Window Program";
  char *icon_name = "basicwin";
  Pixmap icon_pixmap;
  XSizeHints size_hints;
  XIconSize *size_list;
  int count;
  XEvent report;
  GC gc;
  XFontStruct *font_info;
  char *display_name = NULL;
  int window_size = BIG_ENOUGH; /* or TOO_SMALL to display contents */
```

```
progname = argv[0];

if ( (display=XOpenDisplay(display_name)) == NULL ){
 (void) fprintf( stderr, "%s: cannot connect to X server %s\n",
                 progname, XDisplayName(display_name));
 exit( -1 );
}

screen_num = DefaultScreen(display);
display_width = DisplayWidth(display, screen_num);
display_height = DisplayHeight(display, screen_num);

x = y = 0;
width = display_width/3, height = display_height/4;
win = XCreateSimpleWindow(display, RootWindow(display,screen_num),
  x,y,width,height,border_width,BlackPixel(display,screen_num),
  WhitePixel(display,screen_num));

/* Get available icon sizes from Window manager */
if (XGetIconSizes(display, RootWindow(display,screen_num),
      &size_list, &count) == 0)
  (void) fprintf( stderr,
     "%s: WM didn't set icon sizes - using default.\n", progname);
else {
  ;
  /* A real application would search through size_list
   * here to find an acceptable icon size, and then
   * create a pixmap of that size.  This requires
   * that the application have data for several sizes
   * of icons. */
}
/* Create pixmap of depth 1 (bitmap) for icon */
icon_pixmap = XCreateBitmapFromData(display,win,icon_bitmap_bits,
  icon_bitmap_width, icon_bitmap_height);
```

```
size_hints.flags = PPosition | PSize | PMinSize;
size_hints.min_width = 300;
size_hints.min_height = 200;
{
XWMHints wm_hints;
XClassHint class_hints;

/* format of the window name and icon name
 * arguments has changed in R4 */
XTextProperty windowName, iconName;

/* These calls store window_name and icon_name into
 * XTextProperty structures and set their other
 * fields properly. */
if(XStringListToTextProperty(&window_name,1,&windowName)==0){
  fprintf( stderr, "%s: allocation for windowName failed.\n",
                progname);
  exit(-1);
}
if (XStringListToTextProperty(&icon_name, 1, &iconName) == 0) {
  fprintf( stderr, "%s: allocation for iconName failed.\n",
                 progname);
  exit(-1);
}
wm_hints.initial_state = NormalState;
wm_hints.input = True;
wm_hints.icon_pixmap = icon_pixmap;
wm_hints.flags = StateHint | IconPixmapHint | InputHint;
class_hints.res_name = progname;
class_hints.res_class = "Basicwin";
XSetWMProperties(display, win, &windowName, &iconName,
    argv, argc, &size_hints, &wm_hints, &class_hints);
}
```

```
XSelectInput(display, win, ExposureMask | KeyPressMask |
                ButtonPressMask | StructureNotifyMask);
load_font(&font_info);
getGC(win, &gc, font_info);
XMapWindow(display, win);

while (1)  {
  XNextEvent(display, &report);
  switch  (report.type) {
    case Expose:
      if (report.xexpose.count != 0)
        break;
      if (window_size == TOO_SMALL)
        TooSmall(win, gc, font_info);
      else {
        draw_text(win, gc, font_info, width, height);
        draw_graphics(win, gc, width, height);
      }
      break;
    case ConfigureNotify:
      width = report.xconfigure.width;
      height = report.xconfigure.height;
      if ((width < size_hints.min_width) ||
          (height < size_hints.min_height))
        window_size = TOO_SMALL;
      else
        window_size = BIG_ENOUGH;
      break;
    case ButtonPress:
    case KeyPress:
      XUnloadFont(display, font_info->fid);
      XFreeGC(display, gc);
      XCloseDisplay(display);
      exit(1);
```

```
        default:
          /* all events selected by StructureNotifyMask
           * except ConfigureNotify are thrown away here,
           * since nothing is done with them */
          break;
      } /* end switch */
    } /* end while */
}
void getGC(Window win, GC *gc, XFontStruct *font_info)
{
  unsigned long valuemask = 0; /* ignore XGCvalues and use defaults */
  XGCValues values;
  unsigned int line_width = 6;
  int line_style = LineOnOffDash;
  int cap_style = CapRound;
  int join_style = JoinRound;
  int dash_offset = 0;
  static char dash_list[] = {12, 24};
  int list_length = 2;
  *gc = XCreateGC(display, win, valuemask, &values);
  XSetFont(display, *gc, font_info->fid);
  XSetForeground(display, *gc, BlackPixel(display,screen_num));
  XSetLineAttributes(display, *gc, line_width, line_style,
                     cap_style, join_style);
  XSetDashes(display, *gc, dash_offset, dash_list, list_length);
}
void load_font(XFontStruct ** font_info)
{
  char *fontname = "9x15";
  if ((*font_info = XLoadQueryFont(display,fontname)) == NULL)
  {
    fprintf( stderr, "%s: Cannot open 9x15 font\n", progname);
    exit( -1 );
  }
}
```

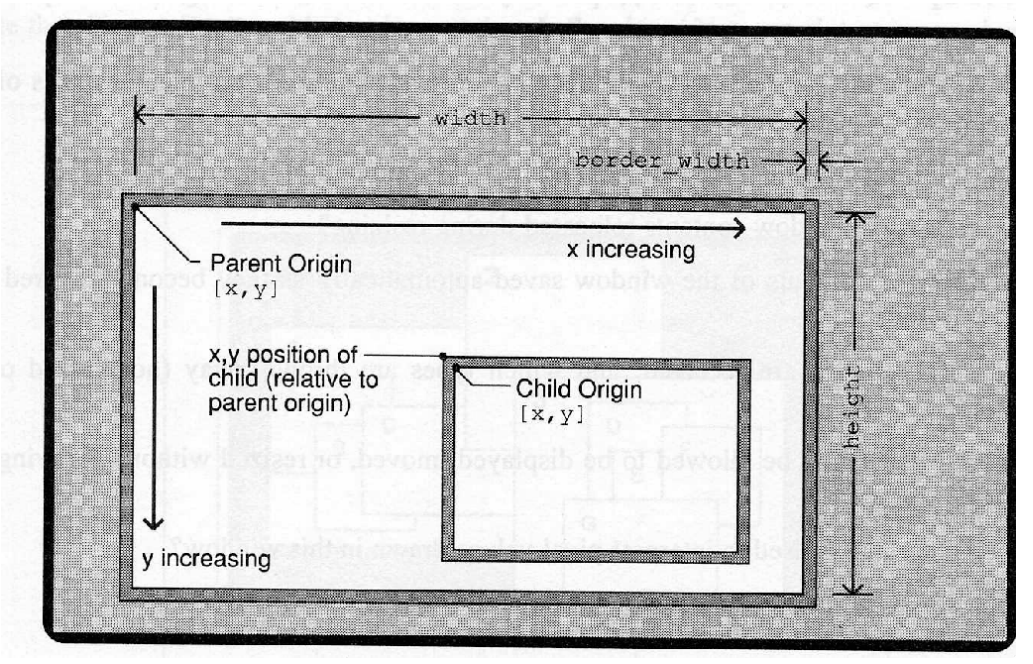
```
draw_text(Window win, GC  gc, XFontStruct *font_info,
  unsigned int win_width, unsigned int win_height)
{
  char *string1 = "Hi! I'm a window, who are you?";
  char *string2 = "To terminate program; Press any key";
  char *string3 = "or button while in this window.";
  char *string4 = "Screen Dimensions:";
  int len1, len2, len3, len4;
  int width1, width2, width3;
  char cd_height[50], cd_width[50], cd_depth[50];
  int font_height;
  int initial_y_offset, x_offset;
  len1 = strlen(string1);
  len2 = strlen(string2);
  len3 = strlen(string3);
width1 = XTextWidth(font_info, string1, len1);
width2 = XTextWidth(font_info, string2, len2);
width3 = XTextWidth(font_info, string3, len3);
font_height = font_info->ascent + font_info->descent;
XDrawString(display, win, gc, (win_width - width1)/2,
  font_height,string1, len1);
XDrawString(display, win, gc, (win_width - width2)/2,
  (int)(win_height - (2 * font_height)),string2, len2);
XDrawString(display, win, gc, (win_width - width3)/2,
  (int)(win_height - font_height), string3, len3);
(void) sprintf(cd_height, " Height - %d pixels",
   DisplayHeight(display,screen_num));
(void) sprintf(cd_width, " Width  - %d pixels",
   DisplayWidth(display,screen_num));
(void) sprintf(cd_depth, " Depth  - %d plane(s)",
   DefaultDepth(display, screen_num));
len4 = strlen(string4);
len1 = strlen(cd_height);
len2 = strlen(cd_width);
len3 = strlen(cd_depth);
```

```
  initial_y_offset = win_height/2 - font_height - font_info->descent;
  x_offset = (int) win_width/4;
  XDrawString(display, win, gc, x_offset, (int) initial_y_offset,
    string4,len4);
  XDrawString(display, win, gc, x_offset, (int) initial_y_offset +
    font_height,cd_height,len1);
  XDrawString(display, win, gc, x_offset, (int) initial_y_offset +
    2 * font_height,cd_width,len2);
  XDrawString(display, win, gc, x_offset, (int) initial_y_offset +
    3 * font_height,cd_depth,len3);
}

void draw_graphics(Window win, GC gc, unsigned int window_width,
  unsigned int window_height)
{
  int x, y;
  int width, height;
  height = window_height/2;
  width = 3 * window_width/4;
  x = window_width/2 - width/2;  /* center */
  y = window_height/2 - height/2;
  XDrawRectangle(display, win, gc, x, y, width, height);
}

void TooSmall(Window win, GC gc,XFontStruct * font_info)
{
  char *string1 = "Too Small";
  int y_offset, x_offset;
  y_offset = font_info->ascent + 2;
  x_offset = 2;

  XDrawString(display, win, gc, x_offset, y_offset, string1,
    strlen(string1));
}
```

3. Creación y manipulación de ventanas

3.1. Características de una ventana

1. Cada ventana tiene una **ventana padre** dónde está contenida.
   - Entrada y salida estará contenida en ventana padre.
   - La ventana raíz ocupa toda la pantalla, y es creada por servidor X.
2. Cada ventana tiene su propio **sistema de coordenadas**.

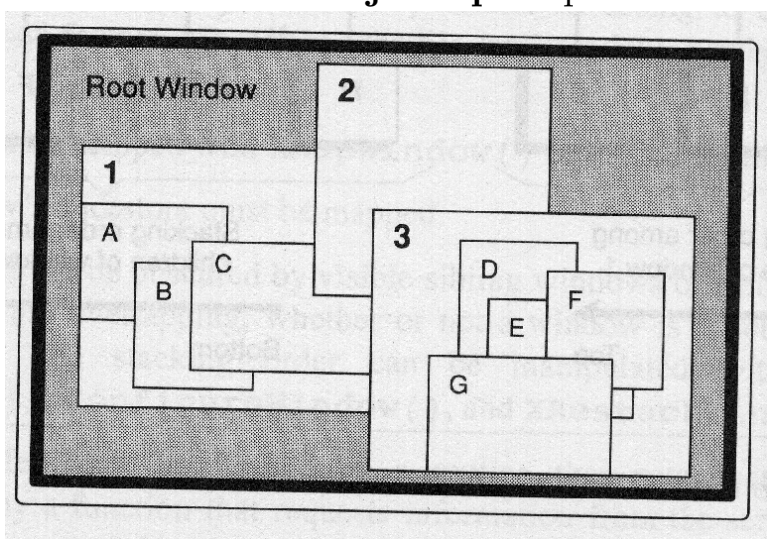


3. La **configuración de una ventana** consiste de su posición, anchura, altura, y anchura de borde, orden de apilamiento.
4. La **geometría** es la anchura, altura y posición.
5. Los colores que pueden usarse en una ventana se definen a través de la **profundidad** y **visual**:
   - Profundidad: Número de bits para definir cada pixel.
   - Visual: Forma en que los valores de pixel son transformados en un color concreto.
6. Cada ventana tiene una **clase**:
   - InputOutput
   - InputOnly: Ventana usada poco en programas de usuario. No tiene borde y el background es trasparente.

7. Cada ventana tiene un conjunto de **atributos** para controlar aspecto y respuesta.
   - Color o patrón para borde y background de ventana.
   - Si se salvan o no automáticamente los contenidos de las ventanas al ser tapadas por otras.
   - Tipos de eventos que reciben.
   - Si se pueden mostrar, mover o redimensionar sin notificarlo al WM.
   - Colormap que usa.
   - Cursor que usa.

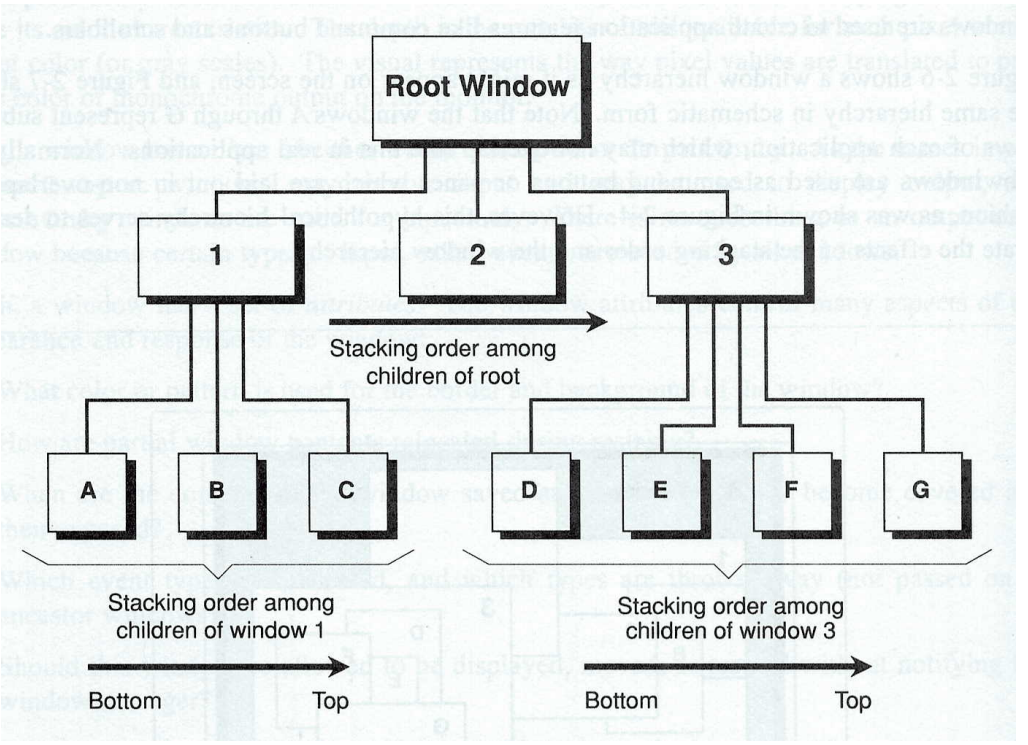
3.2. Jerarquía de ventanas

- Las ventanas se ordenan en una **jerarquía** que forman un árbol.



- Un **icono** es una ventana pequeña que indica que existe una ventana más grande que no está mapeada.
- La **ventana raíz**:
  - Se crea al iniciar el servidor
  - Es de clase InputOutput
  - Siempre está mapeada
  - Su tamaño no puede cambiarse
  - Borde de tamaño 0
  - Nunca se iconiza

- Existe un orden de apilamiento entre las ventanas de la jerarquía que determina qué ventanas están encima de otras:



3.3. Mapeado y visibilidad

- Al crear una ventana, no aparecerá inmediatamente en pantalla hasta que no sea mapeada.
- Mapear una ventana hace que sea marcada como elegible para visualizarse en el display.
- `XMapWindow` mapea la ventana en su posición en el orden de apilamiento (una ventana nueva es mapeada en el tope).
- `XMapRaised` coloca la ventana en el tope del orden de apilamiento.
- Desmapear se hace con `XUnMapWindow()` y `XUnmapSubWindows`
- Factores que afectan que una ventana sea visible en pantalla:
  - Que la ventana haya sido mapeada.
  - Que las ventanas antecesoras estén mapeadas.
  - Que no esté tapada por ventanas hermanas o hermanas de antecesoras.

  - Que el buffer de peticiones sea enviado al servidor (con `XFlush` o bien una petición de información al servidor).
  - El mapeo inicial de la ventana principal es un caso especial. Por razones complicadas, esta ventana debe esperar al primer evento `Expose` antes de suponer que la ventana es visible y que se puede dibujar en ella.

- La configuración y atributos de una ventana se mantienen cuando una ventana es desmapeada.
- Pero no se preservan automáticamente los contenidos de las ventanas:
  - Operaciones gráficas en ventanas no visibles o desmapeadas, no tienen ningún efecto.
  - Los contenidos gráficos de la ventana se borran cuando la ventana es tapada por otra y luego destapada.

3.4. Atributos de una ventana

- Los atributos pueden definirse al crear la ventana (con `XCreateWindow()`) o posteriormente (con `XChangeWindowAttributes`).
- Si creamos la ventana con `XCreateSimpleWindow()`, la mayoría de los atributos son heredados de ventana padre.
- Existen además funciones para modificar atributos individuales
- El proceso para definir los atributos con `XCreateWindow()` y `XChangeWindowAttributes` es el mismo. Se definen los miembros de una estructura `XSetWindowAttributes` con los valores deseados, y se crea una *máscara* que indica qué campos han sido definidos.

```
typedef struct {
    Pixmap background_pixmap; /* background or None or ParentRelative */
    unsigned long background_pixel; /* background pixel */
    Pixmap border_pixmap; /* border of the window */
    unsigned long border_pixel; /* border pixel value */
    int bit_gravity; /* one of bit gravity values */
    int win_gravity; /* one of the window gravity values */
    int backing_store; /* NotUseful, WhenMapped, Always */
    unsigned long backing_planes;/* planes to be preseved if possible */
    unsigned long backing_pixel;/* value to use in restoring planes */
    Bool save_under; /* should bits under be saved? (popups) */
    long event_mask; /* set of events that should be saved */
    long do_not_propagate_mask; /* set of events that should not propagate */
    Bool override_redirect; /* boolean value for override-redirect */
    Colormap colormap; /* color map to be associated with window */
    Cursor cursor; /* cursor to be displayed (or None) */
} XSetWindowAttributes;
```

- Para construir la máscara usaremos las constantes predefinidas de la siguiente tabla:

MIEMBRO	FLAG	BIT
`background_pixmap`	CWBackPixmap	0
`background_pixel`	CWBackPixel	1
`border_pixmap`	CWBorderPixmap	2
`border_pixel`	CWBorderPixel	3
`bit_gravity`	CWBitGravity	4
`win_gravity`	CWWinGravity	5
`backing_store`	CWBackingStore	6
`backing_planes`	CWBackingPlanes	7
`backing_pixel`	CWBackingPixel	8
`override_redirect`	CWOverrideRedirect	9
`save_under`	CWSaveUnder	10
`event_mask`	CWEventMask	11
`do_not_propagate_mask`	CWDontPropagate	12
`colormap`	CWColormap	13
`cursor`	CWCursor	14

- El valor por defecto de cada miembro es:

MIEMBRO	**Valor por defecto**
`background_pixmap`	None
`background_pixel`	Undefined
`border_pixmap`	CopyFromParent
`border_pixel`	Undefined
`bit_gravity`	FogetGravity
`win_gravity`	NorthWestGravity
`backing_store`	NotUseful
`backing_planes`	All 1's
`backing_pixel`	0 (zero)
`override_redirect`	False
`save_under`	False
`event_mask`	0
`do_not_propagate_mask`	0
`colormap`	CopyFromParent
`cursor`	None

- Selección de atributos al crear la ventana:

```
Display *display;
int x,y,depth;
unsigned int width, height, border_width;
Visual *visual;
unsigned int class;
XSetWindowAttributes setwinattr;
unsigned long valuemask;

valuemask = CWBackPixel | CWBorderPixel ;
setwinattr.background_pixel = WhitePixel(display,screen_num);
setwinattr.border_pixel = BlackPixel(display,screen_num);
window = XCreateWindow(display, parent, x , y ,width, height,
            border_width, depth, class , visual, valuemask,
            &setwinattr);
```

- Cambio de atributos de una ventana ya creada:

```
Display *display;
Window window;
XSetWindowAttributes setwinattr;
unsigned long valuemask;

valuemask = CWBackPixel | CWBorderPixel ;
setwinattr.background_pixel = WhitePixel(display,screen_num);
setwinattr.border_pixel = BlackPixel(display,screen_num);
XChangeWindowAttributes(display, window, valuemask, &setwinattr);
```

Background de la ventana

- Es la superficie de la ventana en la que se dibujan los gráficos.
- Puede ser definido con un color sólido, o con un patrón de relleno (pixmap).
- Atributos relacionados:
  1. `background_pixel:` Background de la ventana. (undefined ó pixel)
  2. `background_pixmap:` Pixmap usado como background. (None, pixmap ID, ParentRelative)
- Puede establecerse con `XSetWindowBackground()` `XSetWindowBackgroundPixmap()`
- El background es redibujado automáticamente por el servidor ante eventos `Expose`
- Una modificación en el background no se hace efectiva hasta que no se produzca un evento `Expose` (usar `XClearWindow` y `XFlush`).

Borde de la ventana

- Puede ser también un color sólido o un patrón de rellenado.
- Atributos relacionados:
  1. `border_pixmap:` (CopyFromParent, None, Pixmap ID)
  2. `border_pixel:` (undefined, pixel value)
- Puede establecerse con `XSetWindowBorderPixmap()` y `XSetWindowBorder()`

Bit de gravedad: `bit_gravity`

- Controla la nueva colocación de los contenidos de la ventana cuando ésta cambia de tamaño.
- Raramente se usa. Además algunos servidores X no la permiten.
- Tiene 11 posibles valores (página 101 libro): `ForgetGravity`, `StaticGravity`, `NorthWestGravity`, etc.

Gravedad de la ventana: `win_gravity`

- Controla la nueva colocación de subventanas cuando la ventana padre cambia de tamaño.
- Este atributo se define en la ventana hija.
- Tiene 11 posibles valores (página 105 libro):`UnmapGravity`, `StaticGravity`, `NorthWestGravity`, `NorthGravity`, etc.

Backing store:

- Permite que el server mantenga automáticamente los contenidos de la ventana.
- Disponible sólo en server de altas prestaciones (`DoesBackingStore()`).
- Atributos relacionados:
  1. `backing_store`**:** Indica cuándo se salvan los contenidos de la ventana: `NotUseful` (no se solicita), `WhenMapped` (cuando la ventana está mapeada), `Always` (siempre).
  2. `backing_planes`**:** Indica los planos que son preservados.
  3. `backing_pixel`**:** Valor de pixel para planos no preservados.

Save under: `save_under`

- Controla si los contenidos de la pantalla bajo la ventana deben salvarse automáticamente justo antes de que ésta sea mapeada, y repintados justo después de que se desmapee.
- Posibles valores: False, True.

Manejo de eventos

- Los atributos `event_mask` y `do_not_propagate_mask` controlan la propagación de eventos a través de la jerarquía de ventanas.
- `event_mask` es definida normalmente con `XSelectInput()`

Substructure Redirect Override: `override_redirect`:

- Con el valor True permite a la aplicación mapear, mover, redimensionar y cambiar anchura de borde de la ventana sin la intervención del WM.

Paleta de color: `colormap`:

- Especifica la paleta de color (colormap) que se usará para interpretar los valores de pixel dibujados en la ventana.
- Posibles valores: CopyFromParent, ID de colormap.
- Puede establecerse con `XSetWindowColormap()`

Cursor: `cursor`

- Posibles valores: None, ID de cursor.
- Puede establecerse con `XDefineCursor()` y `XUndefinedCursor()`

4. Programación con Eventos

4.1. Eventos

- Paquete de información generado por el servidor cuando ocurren ciertas acciones.
  Los eventos se ponen en una cola del cliente.
  Ejemplos: ButtonPress, ButtonRelease, MapNotify, UnMapNotify, EnterNotify, LeaveNotify

- Programación con eventos diferente a programación tradicional.

  En los programas tradicionales (sin interfaz gráfico), el programa se detiene en las operaciones de entrada, hasta que el usuario teclea el correspondiente dato.
  Ahora el programa no se puede detener nunca, ya que debe por ejemplo controlar los eventos Expose que pueden provocar otras aplicaciones en nuestras ventanas.

- No hay forma de predecir el orden en que llegan los eventos.

4.2. Esquema básico de un programa con eventos

1. Selección de eventos para cada ventana (XSelecInput(), XChangeWindowAtributes(), XCreateWindow())

   ```
   XSelectInput(display, window, ButtonPressMask|KeyPressMask
                |ExposureMask);
   ```

2. Mapear ventanas: `XMapWindow(display,window)`

3. Bucle de eventos: `XNextEvent(display,&evento)`

4.3. Procesamiento de eventos

- Un evento se implementa en un paquete de información almacenado en una estructura.
  La estructura de evento más simple es `XAnyEvent` que contiene los campos comunes a todos los eventos.

```
typedef struct {
    int type; // Tipo de evento
    unsigned long serial;  //Ultima peticion procesada por server
                           //(uso en depuracion)
    Bool send_event;  // Si evento fue enviado con XSendEvent
                      // (True o False)
    Display *display; // Display donde se ha leido el evento
    Window window;    // Ventana que recibe el evento
} XAnyEvent;
```

- El parámetro `evento` de `XNextEvent(Display *display, XEvent *evento)` es de tipo `XEvent *`. Este tipo está definido como una `union`, que contiene todas las estructuras de eventos.

```
typedef union _XEvent {
        int type; /* must not be changed; first element */
        XAnyEvent xany;
        XKeyEvent xkey;
        XButtonEvent xbutton;
        XMotionEvent xmotion;
        XCrossingEvent xcrossing;
        XFocusChangeEvent xfocus;
        XExposeEvent xexpose;
        XGraphicsExposeEvent xgraphicsexpose;
        XNoExposeEvent xnoexpose;
        XVisibilityEvent xvisibility;
        XCreateWindowEvent xcreatewindow;
        XDestroyWindowEvent xdestroywindow;
        XUnmapEvent xunmap;
        XMapEvent xmap;
        XMapRequestEvent xmaprequest;
        XReparentEvent xreparent;
```

```
        XConfigureEvent xconfigure;
        XGravityEvent xgravity;
        XResizeRequestEvent xresizerequest;
        XConfigureRequestEvent xconfigurerequest;
        XCirculateEvent xcirculate;
        XCirculateRequestEvent xcirculaterequest;
        XPropertyEvent xproperty;
        XSelectionClearEvent xselectionclear;
        XSelectionRequestEvent xselectionrequest;
        XSelectionEvent xselection;
        XColormapEvent xcolormap;
        XClientMessageEvent xclient;
        XMappingEvent xmapping;
        XErrorEvent xerror;
        XKeymapEvent xkeymap;
} XEvent;
```

- La unión `XEvent` contiene en primer lugar el campo `type` para conocer el tipo del evento:

```
XNextEvent(display,&report);
switch (report.type) {
  case Expose:
     ...
     break;
}
```

- Una vez conocido el tipo del evento usar la *estructura de evento* correspondiente. Por ejemplo para un evento `Expose` usaremos la estructura `XExposeEvent`.

```
XNextEvent(display, &evento);
switch  (report.type) {
  case Expose:
    if (evento.xexpose.count != 0)
      break;
    else
      ...
}
```

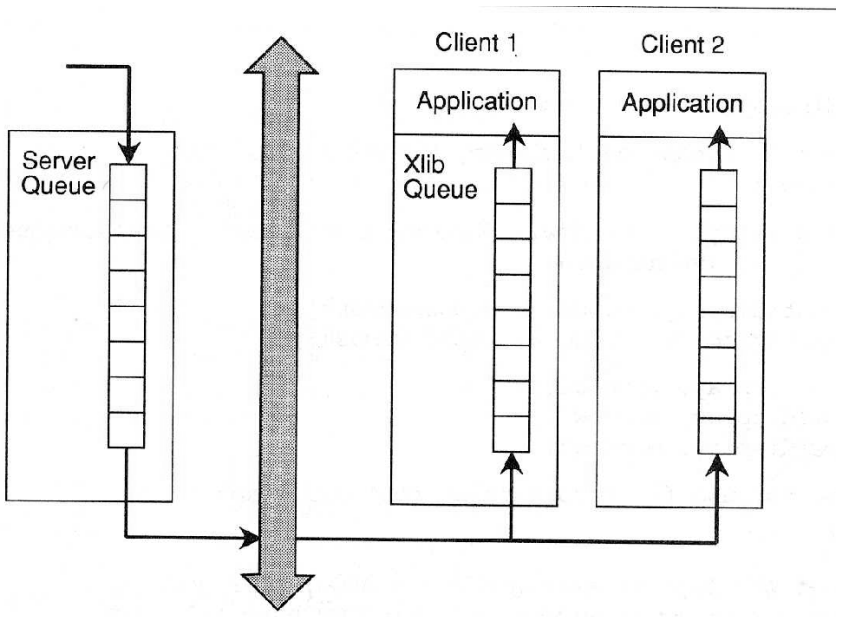
```
typedef struct {
        int type;
        unsigned long serial; /* # of last request processed by server */
        Bool send_event; /* true if this came from a SendEvent request */
        Display *display; /* Display the event was read from */
        Window window;
        int x, y;
        int width, height;
        int count; /* if non-zero, at least this many more */
} XExposeEvent;
```

- **Esquema básico del bucle de eventos**:
  - Mirar el tipo de evento.
  - Luego mirar la ventana dónde ocurrió el evento.

```
XEvent evento;
while(1) {
  XNextEvent(display,&evento);
  switch(evento.type) {
    case Expose:
      if(evento.xexpose.window==window1)
         ...
      else if(evento.xexpose.window==window2)
         ...
      else if(evento.xexpose.window==window3)
         ...
      break;
    case ButtonPress:
  }
}
```

4.4. Cola de eventos

- Cola de eventos del servidor: Enviada periódicamente a los clientes.
- Cola de eventos de cada cliente.



4.5. Funciones de lectura de eventos

Se diferencian en:

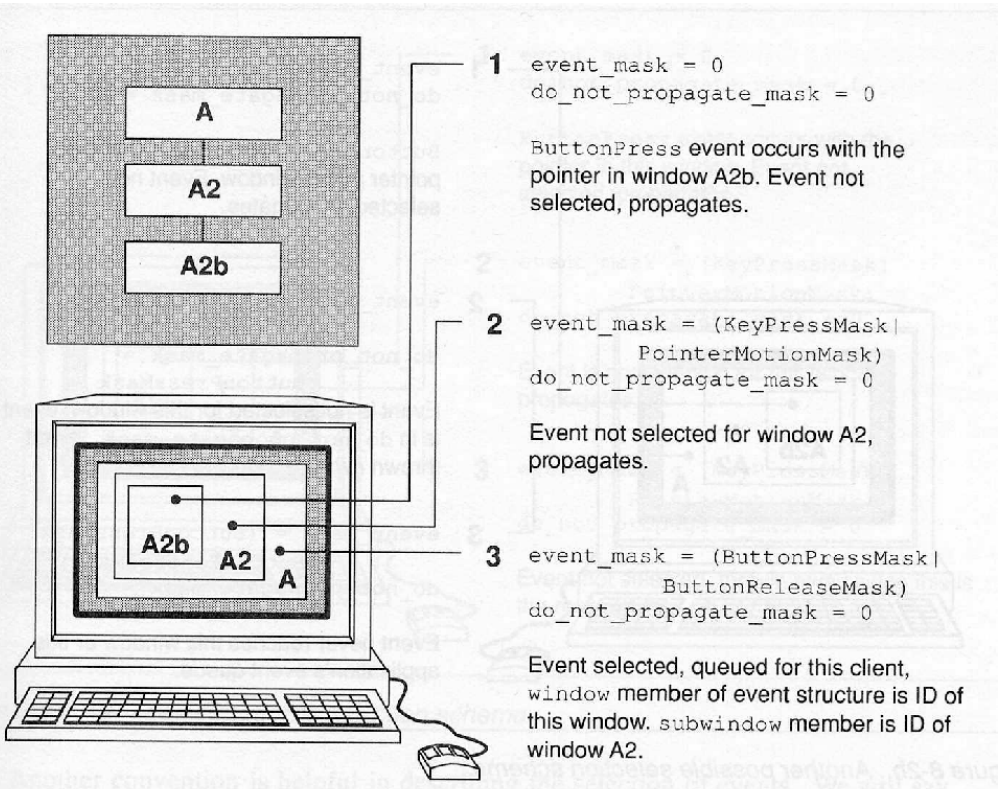
1. Si se fijan en la ventana o el tipo de evento.
2. Si borran el evento de la cola
3. Si detiene el programa
4. Si se envía el buffer de peticiones
5. Si usan una rutina para decidir si el evento debe o no devolverse.

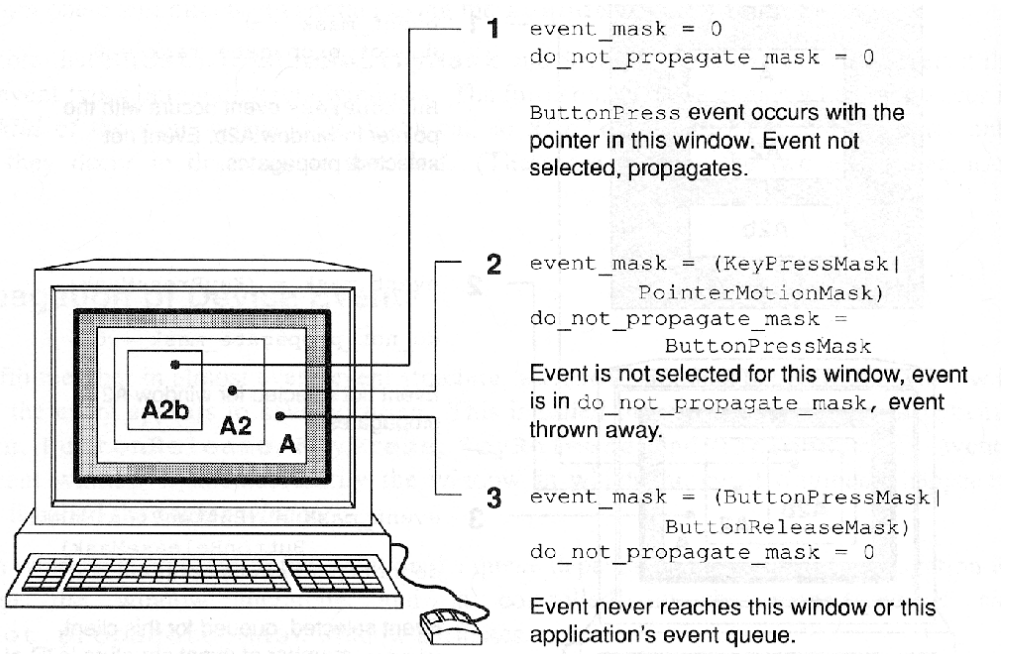
- `XNextEvent(Display *display,XEvent *event)`: Obtiene el siguiente evento de cualquier tipo y de cualquier ventana. Si no hay eventos en la cola se detiene el programa y además envía el buffer de peticiones al servidor.

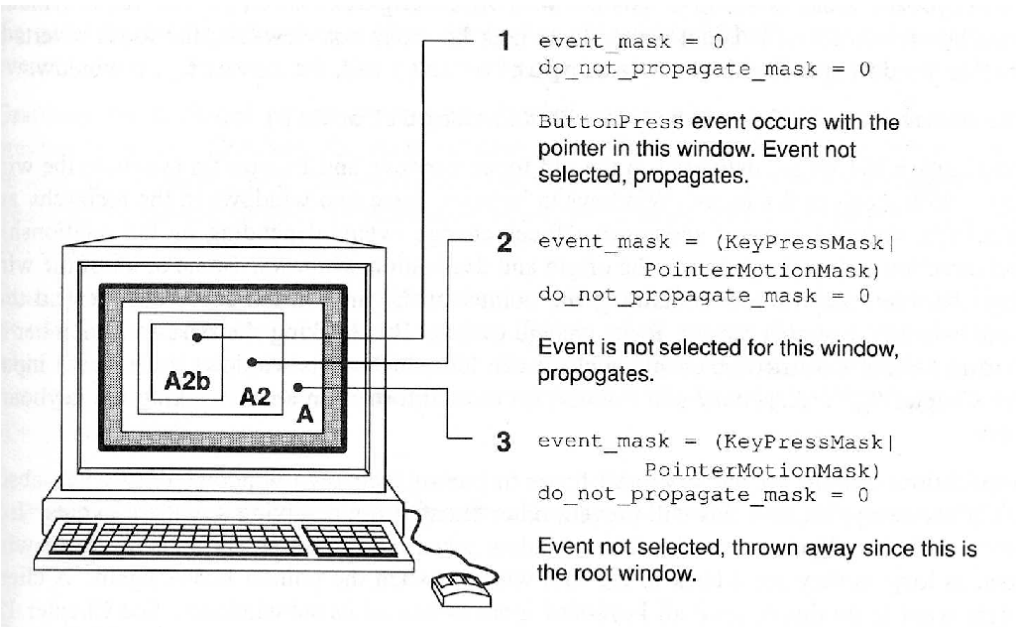
- `XMaskEvent(Display *display,long eventMask, XEvent *event)`: Obtiene siguiente evento del tipo solicitado de cualquier ventana. Espera si no encuentra tal evento enviando el buffer de peticiones al servidor.
- `XCheckMaskEvent(Display *display,long eventMask, XEvent *event)`: Igual a `XMaskEvent` pero devuelve `False` sin esperar cuando no encuentra el evento en la cola tras enviar el buffer de peticiones.
- `XWindowEvent(Display *display, Window w, long eventMask, XEvent *event)`: Obtiene el siguiente evento del tipo especificado y ventana especificada. Espera si no hay eventos de tal tipo enviando el buffer de peticiones.
- `XPeekEvent(Display *display,XEvent *event)`: Igual a `XNextEvent` pero no elimina el evento de la cola.
- `Bool XCheckWindowEvent(Display *display, Window w, long eventMask,XEvent *event)`: Igual a `XWindowEvent` devolviendo `False` sin esperar cuando no encuentra el evento en la cola tras enviar el buffer de peticiones.

4.6. Propagación de eventos

- El campo `window` aparece en todos los tipos de eventos. Ventana que recibe el evento (*event window*)
- En `ButtonPress`, `ButtonRelease`, `KeyPress`, `KeyRelease` y `MotionNotify` `window` no es necesariamente la ventana dónde ocurrió el evento (*source window*)
- La *source window* es la ventana visible más baja en la jerarquía que contiene el puntero cuando se produjo el evento (más pequeña).
- *event window* y *source window* coinciden si en la `event_mask` de *source window* estaba ese tipo de evento.
- Si el evento no fue seleccionado se propaga a ventana padre y así sucesivamente hasta que llega a una ventana que lo tiene seleccionado. Si ninguna antecesora lo tenía seleccionado el servidor no envía el evento.
- `do_not_propagated_mask` determina si cuando un evento llega a una ventana y encuentra que no está seleccionado, el evento debe enviarse o no al padre. Por defecto sí lo hará. Se utiliza muy poco
  Ejemplo de uso: En una aplicación con dos ventanas, una padre y una hija, supongamos que queremos que el usuario pueda dibujar en la ventana hija al mover el ratón mientras se mantiene pulsado uno de los botones. El programa acabará con un evento `ButtonPress` en la ventana padre. La ventana hija no necesita seleccionar el evento `ButtonPress`. Esto hace que este evento se propague hacia la padre al pulsar en la hija con la intención de dibujar mientras movemos el ratón, y por tanto que el programa termine. Si definimos `do_not_propagate_mask` de la ventana hija con `ButtonPressMask`, el problema se resuelve.


A
A2
A2b
A2b
A2
A
1
event_mask = 0
do_not_propagate_mask = 0
ButtonPress event occurs with the pointer in window A2b. Event not selected, propagates.
2
event_mask = (KeyPressMask|
PointerMotionMask)
do_not_propagate_mask = 0
Event not selected for window A2, propagates.
3
event_mask = (ButtonPressMask|
ButtonReleaseMask)
do_not_propagate_mask = 0
Event selected, queued for this client, window member of event structure is ID of this window. subwindow member is ID of window A2.


A2b
A2
A
1
event_mask = 0
do_not_propagate_mask = 0
ButtonPress event occurs with the pointer in this window. Event not selected, propagates.
2
event_mask = (KeyPressMask|
PointerMotionMask)
do_not_propagate_mask =
ButtonPressMask
Event is not selected for this window, event is in do_not_propagate_mask, event thrown away.
3
event_mask = (ButtonPressMask|
ButtonReleaseMask)
do_not_propagate_mask = 0
Event never reaches this window or this application's event queue.



4.7. Ventana con el foco de teclado (focus window)

- Sólo *focus window* y descendientes pueden recibir eventos `KeyPress` y `KeyRelease`. Por defecto, la ventana del foco es la raíz.
- `XSetInputFocus()` Cambia la ventana foco y se generan eventos `FocusIn` y `FocusOut`

4.8. Máscaras de eventos

- `KeyPressMask` y `KeyReleaseMask`: Generan un `keycode` que identifica la tecla (dependiente del servidor)
  Usar `XLookupString(XKeyEvent *event, char *buffer, int nbytes,KeySym *keysym, XComposeStatus *statusInOut)` para traducir el `keycode` a un `keysym` (código portable) y a un string ASCII.
- `ButtonPressMask`, `ButtonReleaseMask`:
  `ButtonPress` y `ButtonRelease` permiten obtener botones que causaron el evento y estado del resto de botones y modificadores de teclas (Control, Shift ...)
- `OwnerGrabButtonMask`: Si se activa, cuando se produce un evento `ButtonRelease`, éste se envía a la ventana del cliente dónde está el puntero de ratón (si esta ventana no es del cliente, se envía dónde ocurrió el `ButtonPress`). Si no lo está, los eventos se envían sólo a la ventana dónde ocurrió el evento `ButtonPress`.
- `PointerMotionMask`: Selecciona eventos de movimiento (`MotionNotify`) que ocurren cuando uno o ninguno de los botones están pulsados.
- `PointerMotionHintMask`: Usado junto a otras máscaras de movimiento de ratón para reducir el número de eventos generados. (El servidor envía sólo un evento `MotionNotify` cuando se pulsa tecla, botón cambia de estado, o el cliente llama a `XQueryPointer` o `XGetMotionEvents`)
- `ButtonMotionMask`: Selecciona eventos de movimiento (`MotionNotify`) cuando al menos un botón está pulsado.
- `Button1MotionMask`, `Button2MotionMask` ...:
- `FocusChangeMask`: Seleccionan eventos `FocusIn` y `FocusOut`
- `KeymapStateMask`: Selecciona evento `KeymapNotify`, el cual sigue siempre a un evento `EnterNotify` o `FocusIn` y que permite conocer las teclas que estaban pulsadas al producirse estos eventos.
- `ExposureMask`: Para los eventos `Expose`.

- `VisibilityChangeMask`:
- `ColormapChangeMask`: Selecciona evento `ColorNotify` que se produce cuando cambia el `colormap` de una ventana.
- `PropertyChangeMask`: Selecciona evento `PropertyNotify` que indica que se ha cambiado una propiedad de una ventana.
- `StructureNotifyMask` y `SubstructureNotifyMask`: Seleccionan un grupo de eventos que informan del cambio de estado de una ventana.
  - `ConfigureNotify`: Modificación en tamaño, posición, anchura de borde, orden de apilamiento.
  - `DestroyNotify`:
  - `GravityNotify`: Si la ventana se ha movido debido a su `win_gravity`
  - `MapNotify` y `UnMapNotify`
  - `ReparentNotify`
  - `CreateNotify` (sólo con `SubstructureNotifyMask`)
- Eventos seleccionados automáticamente:
  - `MappingNotify`: Informa de cambios en los *mappings* de botones y teclas.
  - `ClientMessage`:
  - `SelectionClear`, `SelectionNotify`, `SelectionRequest`: Para comunicar información entre aplicaciones.
  - `GraphicsExpose`, `NoExpose`: Generados por `XCopyArea` y `XCopyPlane`. El primero indica que un área fuente no pudo copiarse en destino porque el fuente estaba tapado. El segundo indica que la copia está completamente disponible.

5. Contexto gráfico

5.1. Introducción

- Especifica variables que se aplican a las primitivas gráficas (texto, puntos, líneas, imágenes, rellenado de áreas ...)
- Es un método que reduce el tráfico de información entre cliente y servidor.
- Simplifican el uso de las primitivas gráficas, al necesitar menos argumentos

5.2. Creación del contexto gráfico

- Se usa `XCreateGC(display,drawable,valuemask,values)`
  - `Display *display`
  - `Drawable drawable`: Sólo usado para conocer el screen usado, y su profundidad. Este GC podrá usarse en cualquier ventana o Pixmap con la misma screen y profundidad.
  - `unsigned long valuemask`: Especifica los campos de `values` que vamos a modificar.
  - `XGCValues *values`: Estructura con los campos del contexto gráfico.
- Ejemplo de creación:

```
Gc gc;
XGCValues values;
unsigned long valuemask;
...
values.foreground = BlackPixel(display,screen_num);
values.background = WhitePixel(display,screen_num);
gc=XCreateGC(display,RootWindow(display,screen_num),
             (GCForeground | GCBackground), &values);
```

```
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, FillOpaeueStippled */
    int fill_rule;      /* EvenOddRule, WindingRule */
    int arc_mode;    /* ArcChord, ArcPieSlice */
    Pixmap tile;    /* tile pixmap for tiling operations */
    Pixmap stipple;     /* stipple 1 plane pixmap for stipping */
    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;
```

Miembro	**Máscara**	**Bit**	**V. defecto**
`function`	GCFunction	0	GCcopy
`plane_mask`	GCPlaneMask	1	all 1's
`foreground`	GCForeground	2	0
`background`	GCBackground	3	1
`line_width`	GCLineWidth	4	0
`line_style`	GCLineStyle	5	LineSolid
`cap_style`	GCCapStyle	6	CapButt
`join_style`	GCJoinStyle	7	JoinMiter
`fill_style`	GCFillStyle	8	FillSolid
`fill_rule`	GCFillRule	9	EvenOddRule
`arc_mode`	GCArcMode	22	ArcPieSlice
`tile`	GCTile	10	Pixmap filled with foreground pixel
`stipple`	GCStipple	11	pixmap filled with 1's
`ts_x_origin`	GCTileStipXOrigin	12	0
`ts_y_origin`	GCTileStipYOrigin	13	0
`font`	GCFont	14	implementation dependent
`subwindow_mode`	GCSubwindowMode	15	ClipByChildren
`graphics_exposures`	GCGraphicsExposures	16	True
`clip_x_origin`	GCClipXOrigin	17	0
`clip_y_origin`	GCClipYOrigin	18	0
`clip_mask`	GCClipMask	19	None
`dash_offset`	GCDashOffset	20	0
`dashes`	GCDashList	21	4 (`[4,4]`)

5.3. Modificación y copia del contexto gráfico

```
XChangeGC(Display *d,GC gc,unsigned long valuemask,
                        XGCValues *values)

XCopyGC(Display *d,GC src, unsigned long valuemask,GC dst)
```

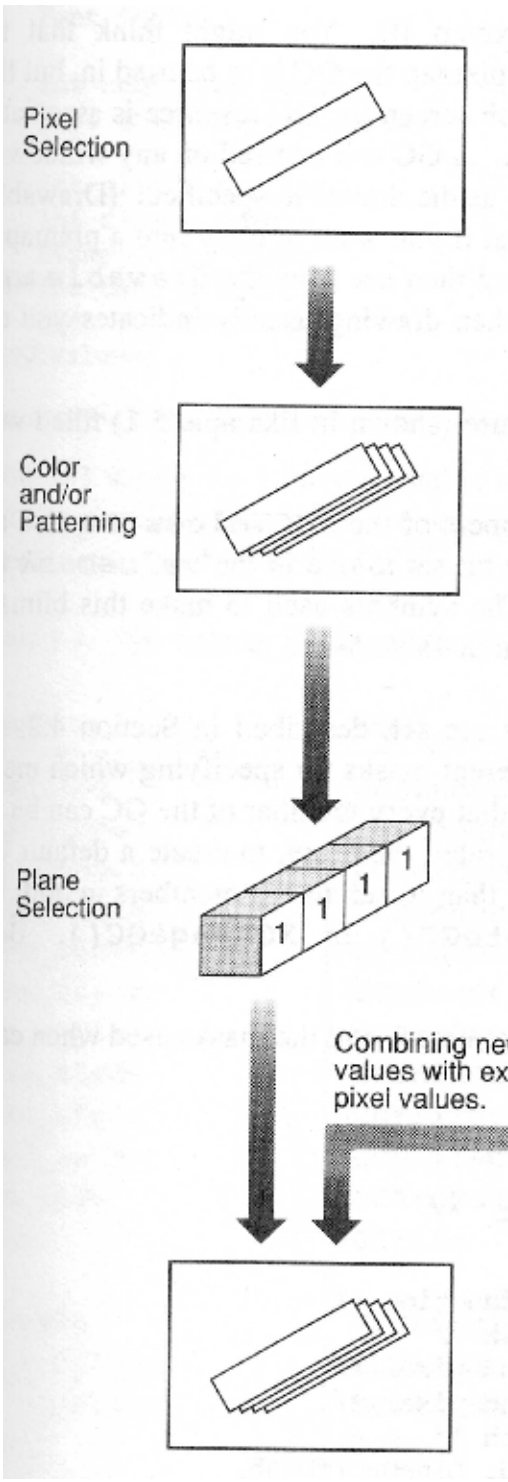
5.4. Liberar el recurso GC

- Usar `XFreeGC(Display *display,GC gc)`

5.5. Etapas en el proceso de dibujo

- Para predecir el efecto de los distintos campos del GC al dibujar con una primitiva gráfica, es útil dividir el proceso de dibujo en varias etapas, aunque realmente cada bit se dibuja de una sola vez.

1. **Selección de pixels**: Se lleva a cabo con la primitiva gráfica y en algunos casos con ciertos elementos del GC, como `line_width`, `clip_mask`, etc. El resultado de esta etapa es un bitmap con los pixels a dibujar con valor 1, y los que no se dibujan a 0.
2. **Color y/o patrón de relleno**: Esta etapa aplica uno o dos colores, o un patrón al resultado de la primera etapa, obteniendo un pixmap con la misma profundidad que el drawable donde se usa. La salida de esta etapa se conoce como la *fuente*.
3. Se aplica una *máscara de planos* para seleccionar los planos del drawable que pueden verse afectados por la primitiva gráfica. Por defecto esta máscara vale todo 1, con lo que no afecta en nada.
4. Los valores de pixel obtenidos en la etapa 3, se combinan con los que hay ya en el drawable, usando *funciones lógicas*



Primitive	GC Componenets Affecting Pixel Selection
lines	line_width, join_style, cap_style
text	font
filled arcs	arc_mode
filled polygons	fill_rule
all	clip_mask, clip_x_origin, clip_y_origin

fill_style	GC Components Used
FillSolid	foreground, background
FillTiled	tile
FillStippled	stipple, foreground
FillOpaqueStippled	stipple, foreground, background

line_style	GC Components Used
LineSolid	foreground
LineOnOffDash	foreground
LineDoubleDash	foreground, background

plane_mask = all ones

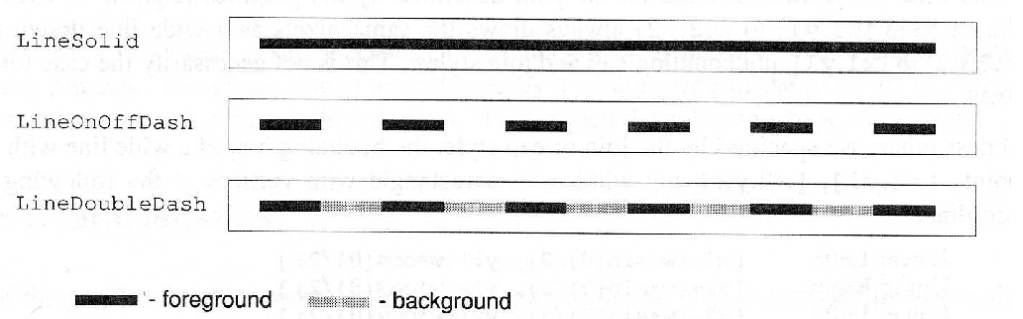
function = GXcopy

5.5.1. Componentes del GC en etapa 1: Selección de pixels

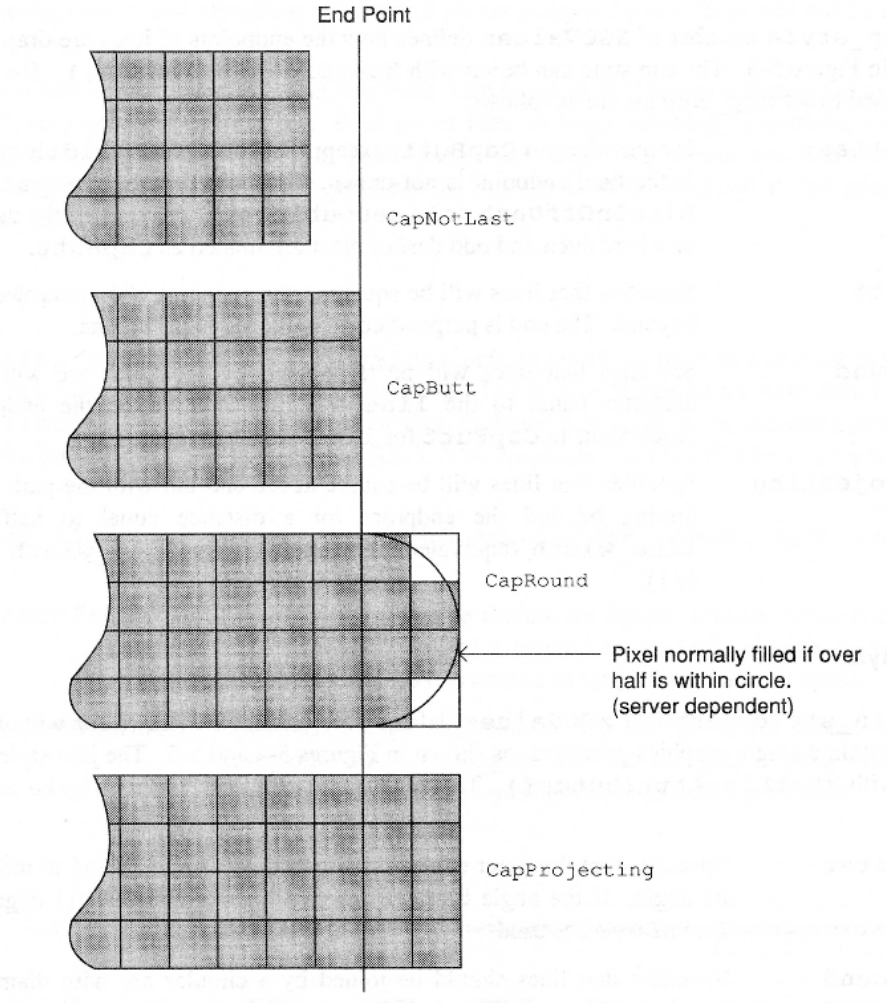
Características de líneas

Pueden definirse también con `XSetLineAttributes` y `XSetDashes`

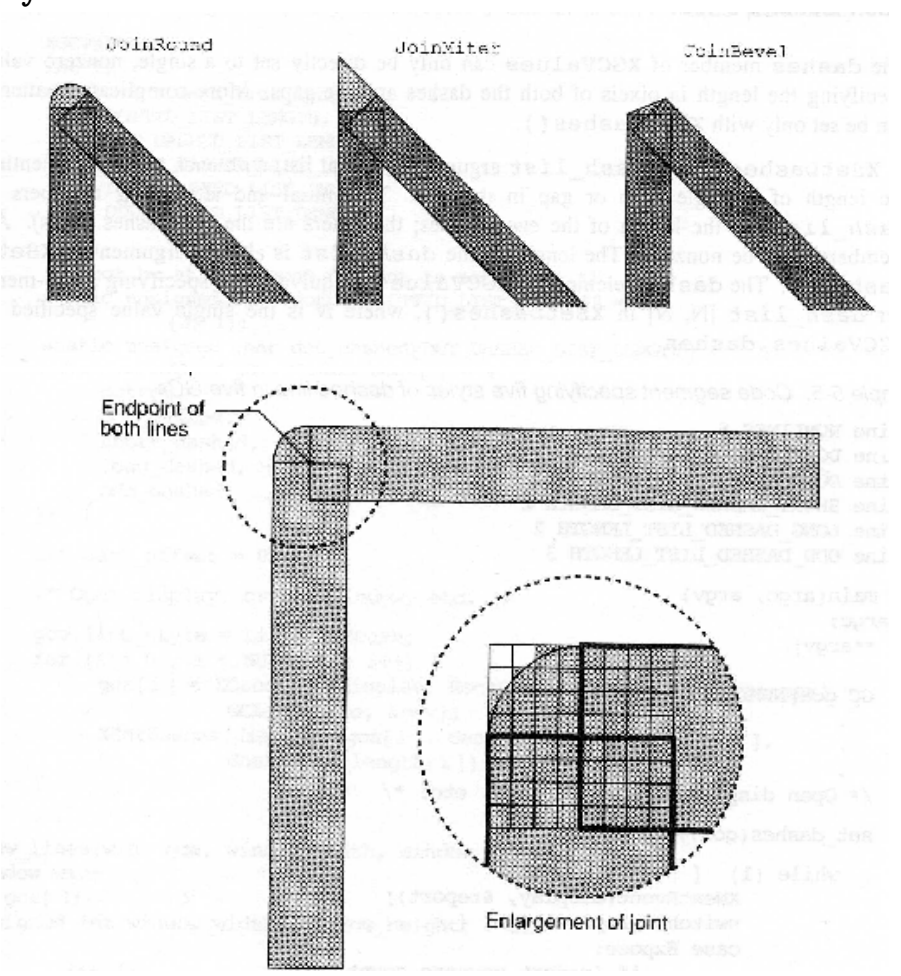
- `line_width`: (0,1, ...)
- `line_style`: (`LineSolid`, `LineOnOffDash` o `LineDoubleDash`).



- `cap_style`: Controla los finales de las líneas.

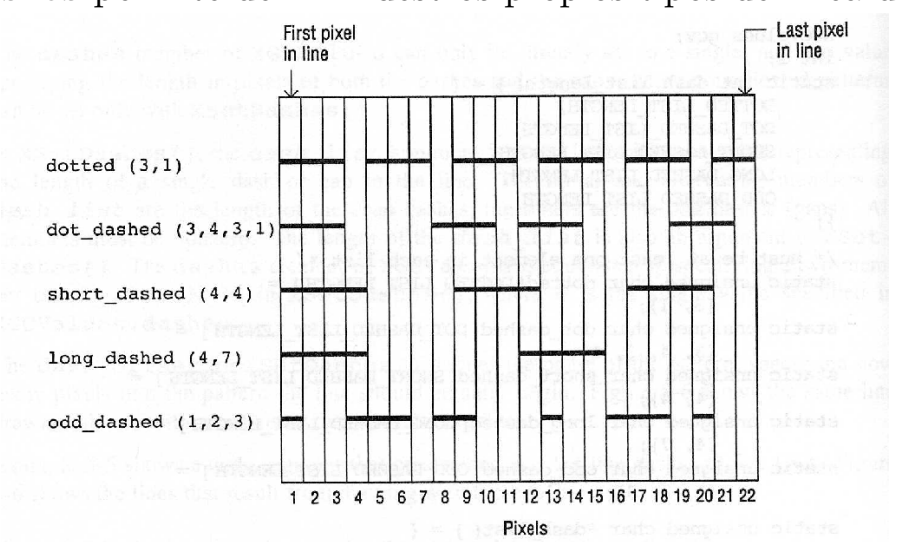


- `join_style`: Controla uniones de líneas consecutivas.



- `dashes`: Longitud de segmentos en líneas discontinuas.
- `dash_offset`: Desplazamiento en el comienzo del patrón de líneas discontinuas.

  `XSetDashes` permite definir nuestros propios tipos de línea discontinua.

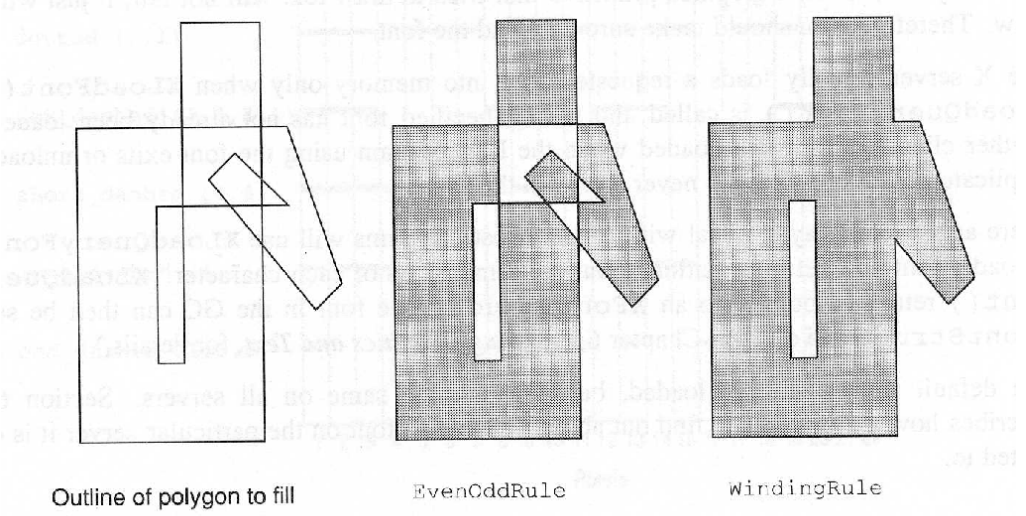


Font de texto

- Es seleccionado con el campo `font`.
- Puede seleccionarse también con `XSetFont`

Regla de rellenado

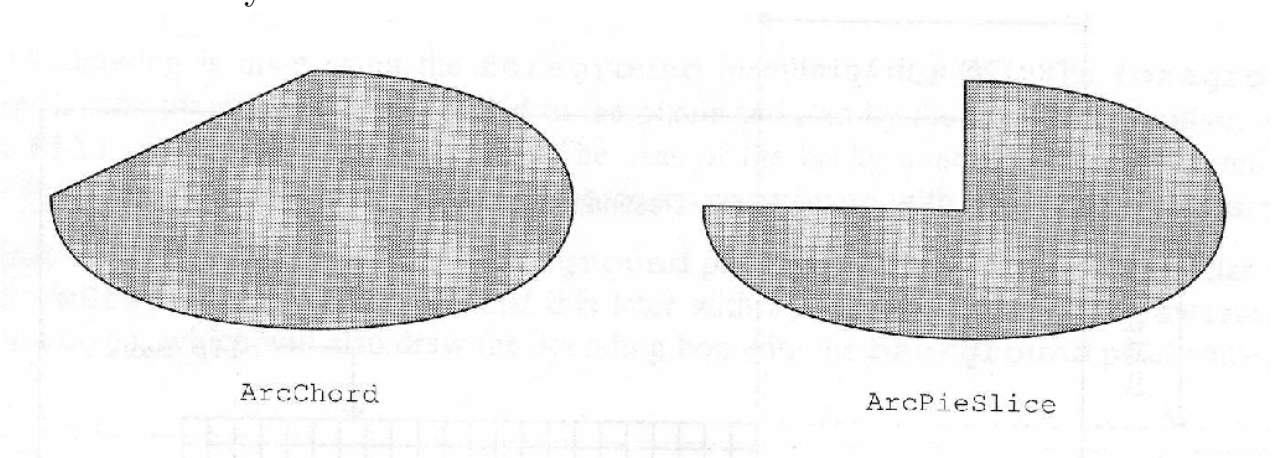
- El campo `fill_rule` (`EvenOddRule` o `WindingRule`) define las áreas que se rellenan al dibujar polígonos con `XFillPolygon`.



- Puede seleccionarse también con `XSetFillRule`

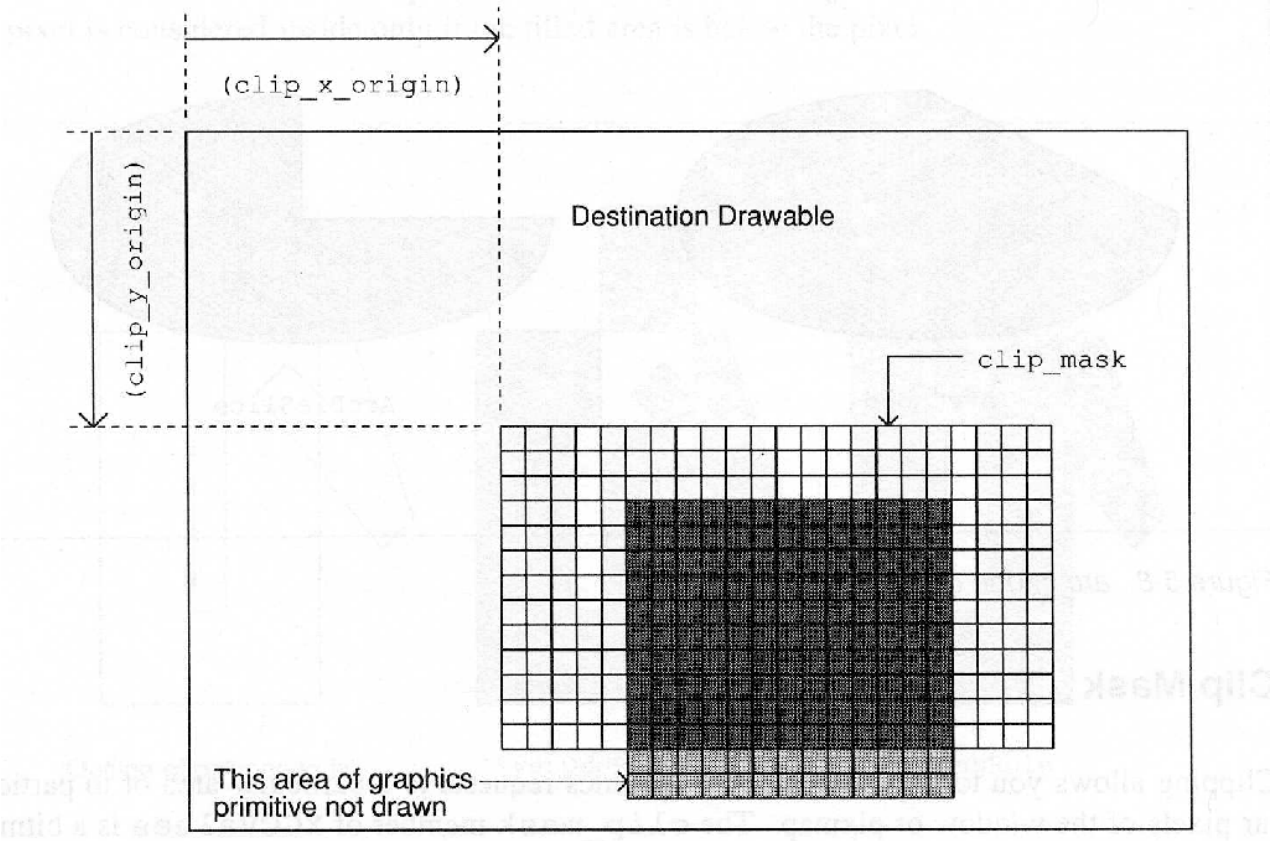
Modo de arco para rellenar

- El campo `arc_mode` controla el rellenado de arcos cuando se usa `XFillArc` y `XFillArcs`.



Máscara de recorte

- `clip_mask`: Bitmap que indica qué píxeles del drawable destino serán afectados.
  Se puede definir también con `XSetClipMask`, `XSetClipRectangles`, `XSetRegion` y `XUnionRectWithRegion`.

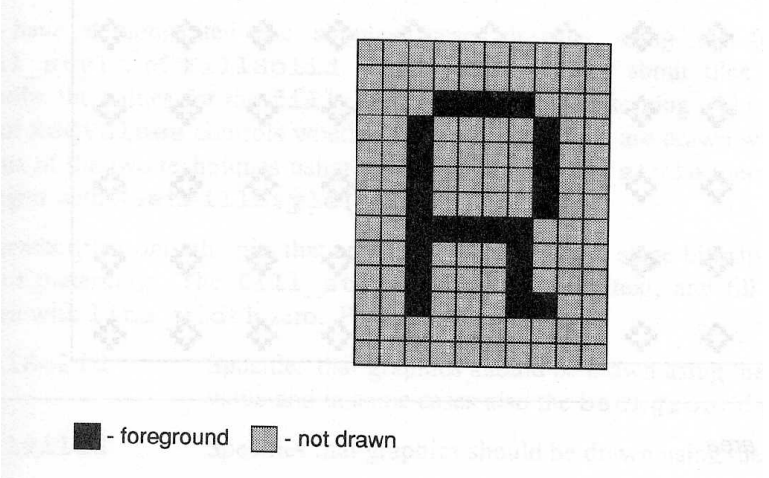


- `cliq_x_origin` y `cliq_y_origin`: Establecen el origen respecto al drawable destino de la máscara de recorte.
  Pueden definirse también con `XSetClipOrigin`.

5.5.2. Compontentes del GC en etapa 2: Control del color y rellenado

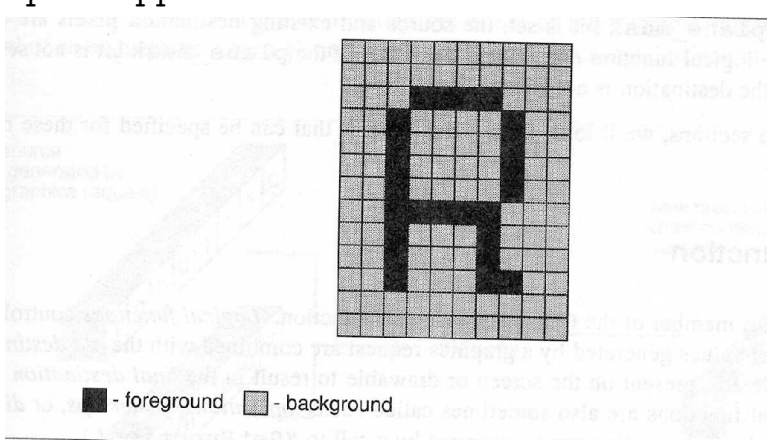
Foreground

- `foreground`: Color con el que se dibujan los pixels activados en fuente.



Background

- `background`: Valor del pixel para los bits no activados en el fuente de las siguientes primitivas gráficas:
  - `XDrawImageString`
  - `XCopyPlane`
  - Cuando se dibujan líneas con `line_style` de tipo `LineDoubleDash`
  - Cuando se rellenan áreas con `fill_style` de tipo `FillOpaqueStippled`

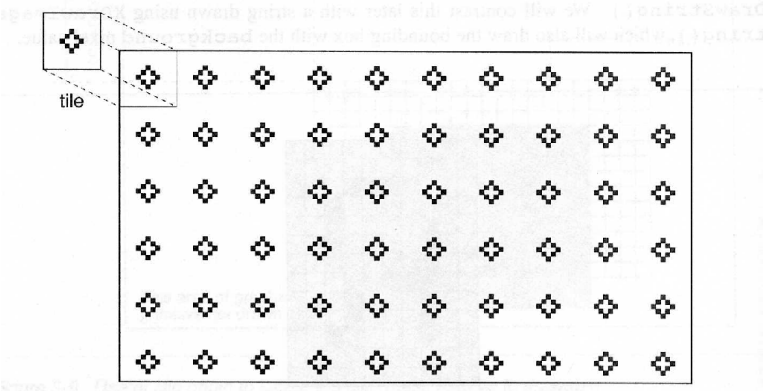


Patrones de rellenado

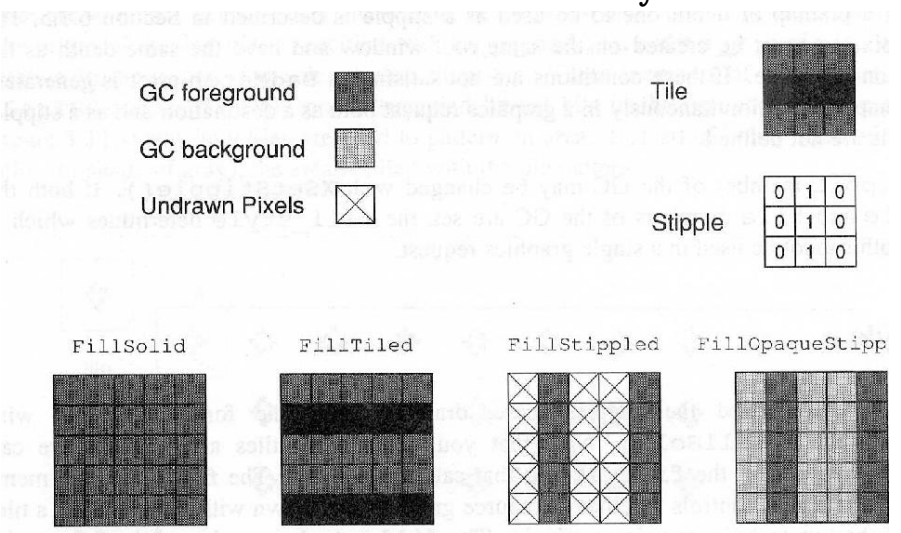
- Tiles: Pixmap usado como patrón de rellenado de áreas.
  Puede especificarse en el GC con `XSetTile`
- Stipples: Pixmap de profundidad 1 usado como patrón de rellenado de áreas. Puede especificarse en el GC con `XSetStipple`

Los siguientes campos del GC influyen en el tipo de patrón (además de los ya vistos `fill_rule` y `arc_mode`):

- `tile`: Pixmap usado como patrón de rellenado.
  Puede establecerse también con `XSetTile()`.



- `stipple`: Similar al tile, pero ahora un stipple tiene profundidad 1.
- `ts_x_origin` y `ts_y_origin`: Establecen el origen respecto al drawable destino del primer tile dibujado.
  El origen puede especificarse también con `XSetTSOrigin`.
- `fill_style`: Para rellenar con color sólido, tile o stipple.
  Puede seleccionarse también con `XSetFillStyle`.



5.5.3. Componentes del GC en etapas 3 y 4: Control del efecto de una primitiva gráfica

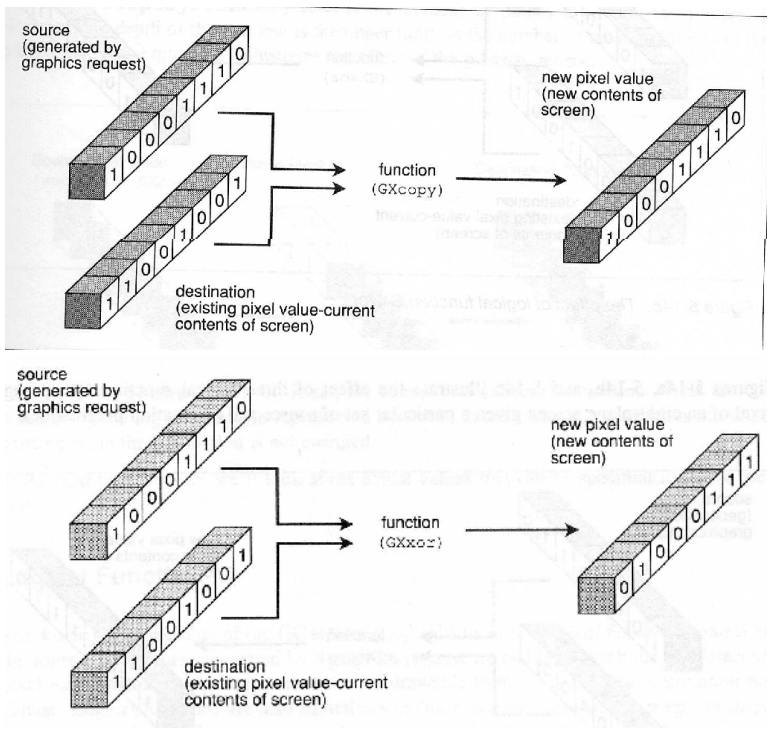
- El pixmap obtenido tras la etapa 2 se le conoce como *fuente*.
  Este pixmap se construye con dos colores: `foreground` y `background`
- Cada pixel fuente y destino se combinan con una función lógica (campo `function` del GC), bit a bit.
- El campo `plane_mask` (máscara de planos activos) restringe la operación a un subconjunto de planos, de forma que sólo algunos de los bits del fuente son usados.
- El campo `clip_mask` restringe la operación a un subconjunto de los pixels.
- Pixel fuente, pixel destino, y `plane_mask` se combinan con el siguiente algoritmo.

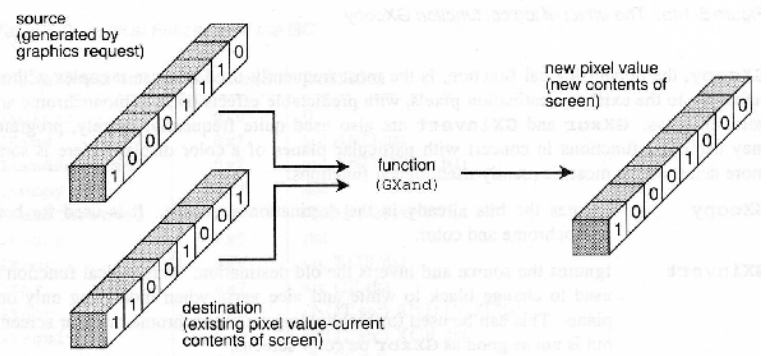
  Para cada bit de cada pixel seleccionado y coloreado en las dos primeras etapas, se aplica la siguiente expresión que indica si el bit se activa en el destino:
  `((src FUNC dst) AND plane_mask) OR (dst AND (NOT plane_mask))`
- O sea, si el `plane_mask` está activo, el fuente y destino se combinan con la función lógica `FUNC`. Si no lo está, el bit existente en el destino no se modifica.

Función lógica

- El campo `function` establece una función lógica.
- Este campo puede definirse también con `XSetFunction`

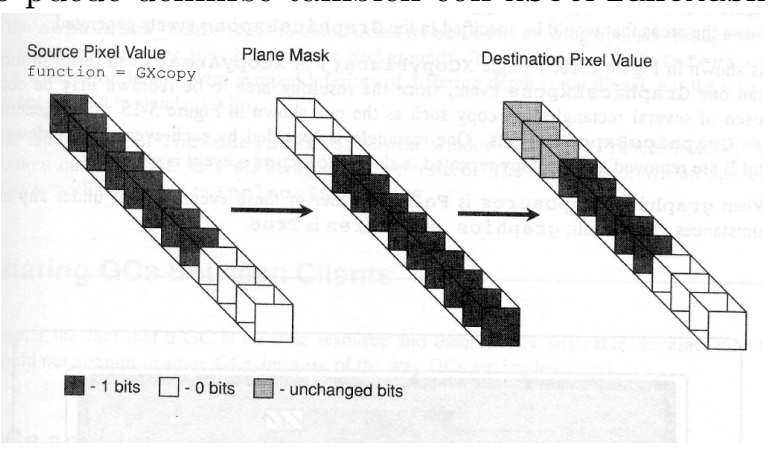
Función lógica	**Definición**
GXclear	0
GXand	src AND dst
GXandReverse	src AND (NOT dst)
GXcopy	src
GXandInverted	(NOT src) AND dst
GXnoop	src XOR dst
GXxor	src XOR dst
GXor	src OR dst
GXnor	(NOT src) AND (NOT dst)
GXequiv	(NOT src) XOR dst
GXinvert	(NOT dst)
GXorReverse	src OR (NOT dst)
GXcopyInverted	(NOT src)
GXorInverted	(NOT src) OR dst
GXnand	(NOT src) OR (NOT dst)
GXset	1





Máscara de planos activos

- El campo `plane_mask` determina los planos del destino que serán modificados.
- Por defecto se modifican todos.
- Este campo puede definirse también con `XSetPlaneMask`



◊ **Campo** `graphics_exposures`

- Al usar `XCopyArea()` y `XCopyPlane()` para copiar áreas de un drawable a otro, es posible que algunas zonas de la región fuente estén desmapeadas o tapadas por otras ventanas.
- En tal caso se generará un evento indicando que una o más áreas no pudieron copiarse.
- Si `graphics_exposures` es True entonces se generan eventos en tal caso.
  - Evento `GraphicsExpose` indica que una zona no se pudo copiar.
  - Evento `NoExpose` indica que la región fuente se pudo copiar sin problemas.

◊ **Campo** `subwindow_mode`

- Permite indicar un modo `IncludeInferiors` en el que al dibujar en la ventana, el resultado aparece incluso aunque en esa misma posición haya una ventana hija.
- El modo normal es `ClipByChildren`.
- Este campo puede establecerse también con `XSetSubwindowMode()`.

5.6. Líneas elásticas

```
win = XCreateSimpleWindow(display, RootWindow(display,screen_num),
  x, y, width, height, borderwidth, border_pixel,
  background_pixel);
XSelectInput(display, win, ExposureMask | ButtonPressMask|
  Button1MotionMask);
gc = XCreateGC(display, win, 0 , &values);
XMapWindow(display, win);
foreground_pixel = BlackPixel(display, screen_num);
while (1)  {
  XNextEvent(display, &report);
  switch  (report.type) {
  case Expose:
    XSetForeground(display, gc, foreground_pixel);
    XSetFunction(display, gc, GXCopy);
    XDrawLine(display, win, gc, 0, 0, currentX, currentY);
    break;
  case MotionNotify:
    XSetForeground(display, gc, foreground_pixel^background_pixel);
    XSetFunction(display, gc, GXxor);
    XDrawLine(display, win, gc, 0, 0, currentX, currentY);
    currentX=report.xmotion.x;
    currentY=report.xmotion.y;
    XDrawLine(display, win, gc, 0, 0, currentX, currentY);
  default:
    break;
  } /* end switch */
} /* end while */
```

6. Dibujo de gráficos y texto

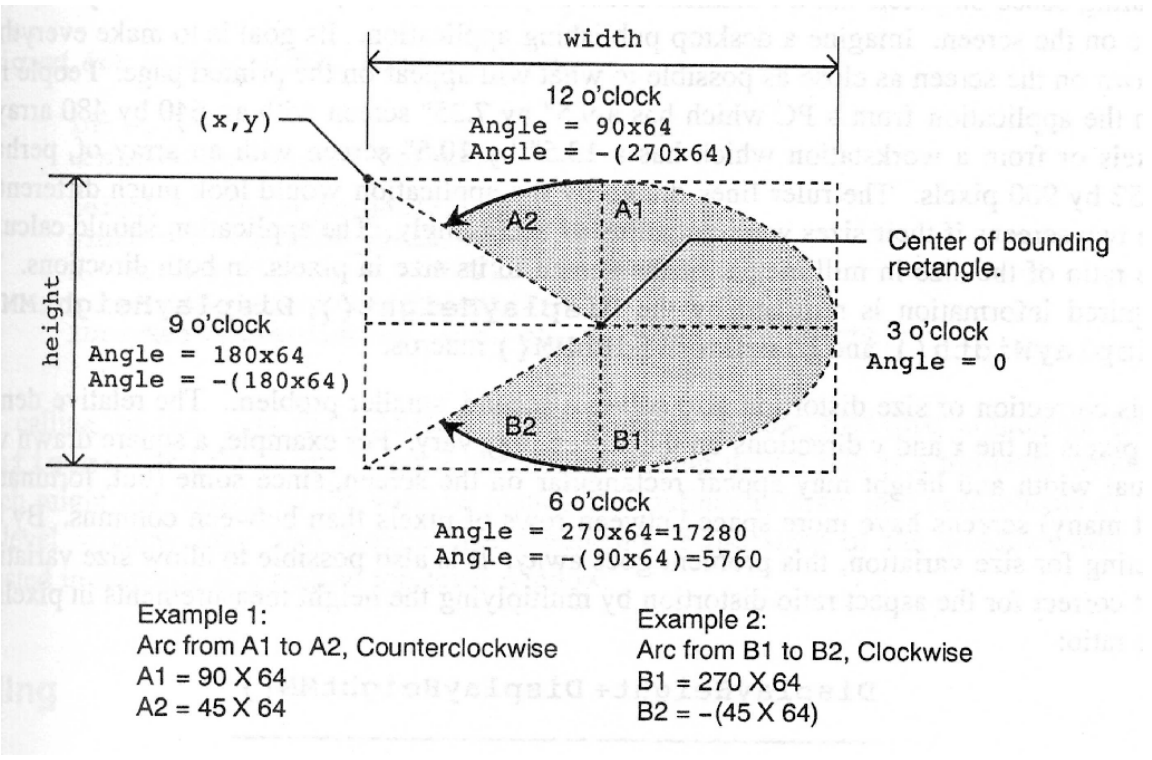
Para usar primitivas gráficas debe definirse un GC especificando al menos el background, foreground y font (si se dibuja texto).

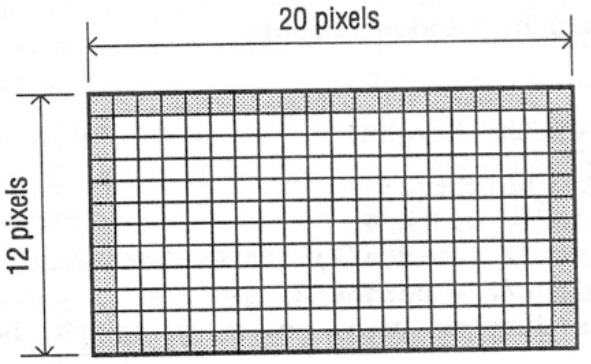
6.1. Primitivas gráficas

- Estas primitivas seleccionan los pixels fuente que luego se manejarán de acuerdo al GC.
- Utilizan el pixel como unidad de medida.
- Los gráficos deberían dibujarse con el evento `Expose`.

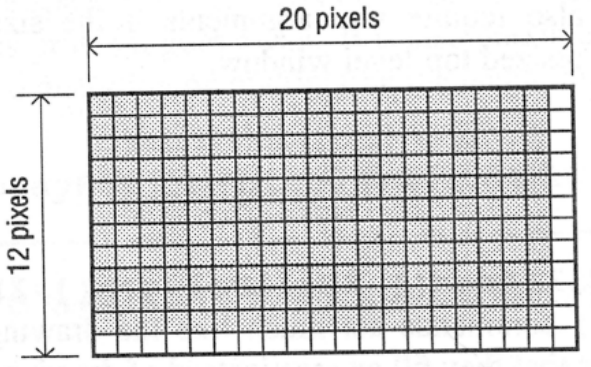
Dibujando puntos, líneas, rectángulos, arcos y polígonos

1. `XDrawPoint`: Dibuja un punto.
2. `XDrawPoints`: Dibuja un vector de puntos.
3. `XDrawLine`: Dibuja una línea.
4. `XDrawLines`: Dibuja una polilínea.
5. `XDrawSegments`: Dibuja un vector de segmentos (requiere un vector de pares de puntos: tipo `XSegment`).
6. `XDrawRectangle`: Dibuja el contorno de un rectángulo.
7. `XDrawRectangles`: Dibuja un vector de rectángulos.
8. `XDrawArc`: Permite dibujar arcos, círculos y elipses.
9. `XDrawArcs`: Permite dibujar varios arcos, círculos y elipses.
10. `XFillArc`: Rellena un arco.
11. `XFillArcs`: Rellena varios arcos.
12. `XFillPolygon`: Rellena un polígono.
13. `XFillRectangle`: Rellena un rectángulo.
14. `XFillRectangles`: Rellena varios rectángulos.






```
XDrawRectangle(display,
drawable, gc, 0, 0, 19, 11);
```




```
XFillRectangle(display,
drawable, gc, 0, 0, 19, 11);
```

6.2. Creación de bitmaps, pixmaps, tiles y stipples

- Todos ellos son de tipo `Pixmap`
- Los datos del pixmap pueden leerse en tiempo de compilación:
  1. `XCreateBitmapFromData`: Crea un pixmap (bitmap) a partir de un fichero (creado con `XWriteBitmapFile`) que es leido en tiempo de compilación, haciendo previamente un `#include` del fichero.
  2. `XCreatePixmapFromBitmapData`: Igual al anterior, pero haciendo que el pixmap tenga una determinada profundidad.
- También pueden leerse en tiempo de ejecución:
  1. `XCreatePixmap`: Crea un pixmap con contenido inicial indefinido.
  2. `XReadBitmapFile`: Lee un bitmap de un fichero y lo coloca en el pixmap que se le pasa como parámetro.
  3. `XReadBitmapFileData`: Lee un bitmap de un fichero, pero no crea el Bitmap. Sólo carga los distintos campos del bitmap en los correspondientes argumentos.
  4. `XWriteBitmapFile`: Escribe el contenido del bitmap en un fichero según el formato de ficheros bitmap de `X`.

6.3. Copiando y borrando áreas

1. `XClearWindow`: Pone el background de la ventana sin generar evento `Expose`.
2. `XClearArea`: Lo mismo pero en un área.
3. `XCopyArea`: Permite copiar un área de un drawable (ventana o pixmap) en otro. Origen y destino deben ser de la misma profundidad.
4. `XCopyPlane`: Para copiar un sólo plano. Puede servir para construir un pixmap a partir de varios bitmaps.

6.4. Fonts y texto

- **Font**: Conjunto de bitmaps que representan texto, cursores u otro conjunto de formas.
- Existen fonts de 1 byte (256 caracteres) y de 2 bytes (65536 caracteres).
- Un font debe cargarse en el servidor antes de usarse.
- Funciones relacionadas con fonts:
  1. `XLoadFont`: Carga el font en el servidor devolviendo su ID.
  2. `XSetFont`: Asocia el font con un GC.
  3. `XQueryFont` (comando `xlsfonts`): Devuelve la estructura `XFontStruct` de un font. Esta estructura permite conocer el tamaño de los caracteres del font.
  4. `XLoadQueryFont`: Carga el font devolviendo su `XFontStruct`.
  5. `XFreeFontInfo`: Libera una estructura de información de un font.
  6. `XUnloadFont`: Descarga un font del servidor (si no lo usa otra aplicación).
  7. `XFreeFont`: Hace las dos operaciones anteriores.
  8. `XListFonts`: Devuelve información sobre los fonts disponibles (como cadenas de caracteres).
  9. `XListFontsWithInfo`: Igual a anterior pero devolviendo los fonts con las estructuras `XFontStruct`
- `XQueryFont(display,XGContextFromGC(DefaultGC(display,screen)))`: Obtiene la información del font por defecto (que siempre está cargado en el servidor).

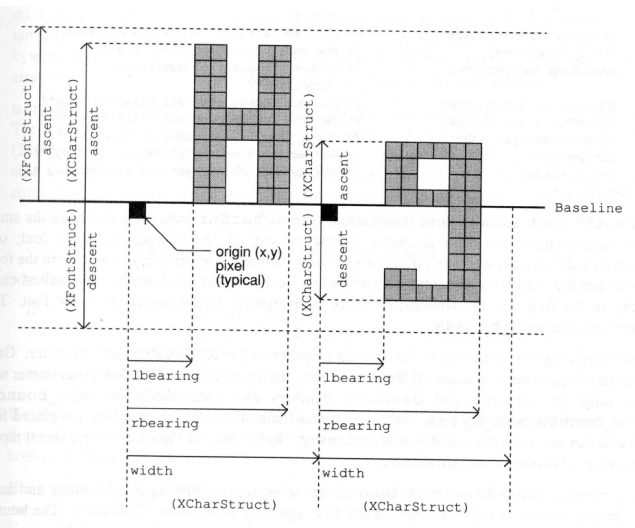
6.5. Métricas de un carácter

- El origen de cada caracter es la *línea base*.
- Hay dos estructuras que controlan la información de los fonts:
  - `XCharStruct`: Contiene información de un caracter del font.

```
typedef struct {
    short lbearing; /* origin to left edge of raster */
    short rbearing; /* origin to right edge of raster */
    short width;    /* advance to next char's origin */
    short ascent; /* baseline to top edge of raster */
    short descent; /* baseline to bottom edge of raster */
    unsigned short attributes; /* per char flags (not predefined) */
} XCharStruct;
```

  - `XFontStruct`: Contiene información de todo el font.

```
typedef struct {
    XExtData *ext_data; /* hook for extension to hang data */
    Font fid;  /* Font id for this font */
    unsigned direction; /* hint about direction the font is painted */
    unsigned min_char_or_byte2;/* first character */
    unsigned max_char_or_byte2;/* last character */
    unsigned min_byte1; /* first row that exists */
    unsigned max_byte1; /* last row that exists */
    Bool  all_chars_exist; /* flag if all characters have non-zero size*/
    unsigned default_char; /* char to print for undefined character */
    int n_properties;  /* how many properties there are */
    XFontProp *properties; /* pointer to array of additional properties*/
    XCharStruct min_bounds; /* minimum bounds over all existing char*/
    XCharStruct max_bounds; /* maximum bounds over all existing char*/
    XCharStruct *per_char; /* first_char to last_char information */
    int ascent; /* log. extent above baseline for spacing */
    int descent; /* log. descent below baseline for spacing */
} XFontStruct;
```



6.6. Dibujando texto

1. `XDrawImageString(16)`: Dibuja caracteres con el foreground del GC y el fondo con el background del GC. Utiliza coordenadas de `baseline`.
2. `XTextWidth(16)`: Devuelve anchura en pixels del string pasado por parámetro.
3. `XTextExtents(16)`: Permite obtener la anchura y altura de un string a partir de un `XFontStruct` pasado por parámetro.
4. `XQueryTextExtents(16)`: Lo mismo pero consultando al servidor.
5. `XDrawString(16)`: Sólo dibuja el foreground de caracteres.
6. `XDrawText`: Permite dibujar varios string con diferentes fonts y diferentes offset horizontal.