EN ESTE NÚMERO: PROGRAMACIÓN DE COMPONENTES CON VISUAL BASIC 6

PROGRAMACIÓN ACTUAL

www.prensatecnica.com

995 pts.
PORTUGAL 1250 ESC (CONT)
5,98 €

AÑO 3 • NÚMERO 27

NIVEL BÁSICO
DELPHI: bibliotecas de enlace dinámico
Biblioteca estándar de plantillas en C++

LAS EMPRESAS DEMANDAN
Oracle 8I, la apertura de las bases de datos al mundo de Internet
Compartición de recursos en Windows NT
Configuración de redes: tuning de la red

NIVEL INTERMEDIO
Programación en Internet: transmisión de audio y vídeo por la Red

CAMPUS ACTUAL
La API de Xlib al descubierto
Creación de un script con mIRC

CONCURSO
Sorteamos 3 MICROSOFT VISUAL BASIC 6.0 PROFESIONAL en Pag. 19

EN EL CD-ROM

PROGRAMACIÓN
- Librerías de Visual Basic
- Librerías de Cobol
- Librerías de DELPHI
- InstallShield 5.5
- DemoShield 5.43
- PackageFor the Web
- InstallShield Express 2.11
- InstallShield for Windows CE

HERRAMIENTAS INTERNET
- Copérnico 99
- Eudora Pro 4.2
- mIRC 5.6
- ICQ Plus 2.01

WINDOWS
- ACDSee 2.41
- WinAmp 2.2
- WinZip 7

Cómo programar con Pentium III

Aprovecha todas las nuevas posibilidades

8 413042 087779 00027

Autor: Santiago Romero

Gráficos en XLIB (IV)

En esta última entrega sobre la API de gráficos de Xlib aprenderemos a trazar texto en pantalla (ya sea transparente u opaco), así como a manejar los eventos de exposición generados por las ventanas ante solapamientos o peticiones de redibujado, lo cual nos ayudará a organizar y acelerar el código del programa.

Las fuentes de texto son elementos muy importantes pues constituyen la manera más estándar de dar información al usuario, de igual forma que se hace en los modos de texto: mediante la utilización de un juego de caracteres que en Xlib se conoce como *font* y que no es más que un array de bitmaps, con un bitmap para cada letra. Existen multitud de fuentes en X-Window para el uso del programador, todas ellas localizables en el directorio */usr/lib/X11/fonts* y de extensión *.PCF*.

CARGANDO LAS FUENTES DE TEXTOS

Antes de usar las fuentes de texto es necesario cargarlas en memoria para que el servidor X sepa qué fuente debe usar para la representación de una determinada cadena de caracteres. Para realizar esta tarea disponemos de la función *XloadFont()*, con el siguiente prototipo:

*Font XloadFont(Display *display, char *NombreFuente);*

Las fuentes de texto constituyen la manera más estándar de informar al usuario

Esta función carga la fuente especificada por la cadena *NombreFuente* y devuelve el ID de la propia fuente, que utilizaremos posteriormente para asociarla a un GC y usar éste al imprimir texto en pantalla.
Los nombres fuentes más habituales en todo servidor X son los siguientes (aunque pueden haber y de hecho suelen haber muchas más):

10x20, 8x16rk, clR6x13, gb16st, 12x24, 9x15, clR6x6, gb24st, 12x24rk, 9x15B, clR6x8, hanglg16, 5x7, clB6x10, clR7x10, hanglm16, 5x8, clB6x12, clR7x12, hanglm24, 6x10, clB8x10, clR7x14, heb6x13, 6x12, clB8x12, clR7x8, heb8x13, 6x13, clB8x13, clR8x10, jiskan16, 6x13B, clB8x14, clR8x12, jiskan24, 6x9, clB8x16, clR8x13, k14, 7x13, clB8x8, clR8x14, nil2, 7x13B, clB9x15, clR8x16, olcursor, 7x13euro, cll6x12, clR8x8, olgl10, 7x13euroB, cll8x8, clR9x15, olgl12, 7x14, clR4x6, cursor, olgl14, 7x14B, clR5x10, deccurs, olgl19, 7x14rk, clR5x6, decsess, vga, 8x13, clR5x8, 8x13B, clR6x10, 8x16, clR6x12, gb16fs.
En el nombre puede verse el tamaño de cada carácter (10x20= caracteres de 10 pixels de ancho por 20 de alto) y, además, su nombre completo en X-Window puede consultarse en los ficheros */usr/X11R6/lib/X11/fonts/misc/fonts.dir* y */usr/X11R6/lib/X11/fonts/misc/fonts.alias*. Mediante el programa *xfd* incluido en X-Window pueden consultarse los diferentes caracteres de las fuentes (ver Figura 1).

Una fuente no es más que un array de bitmaps, con un bitmap para cada letra

Si nos fuera necesario saber las características (anchura, altura, etc.) de la fuente para organizar nuestras funciones (no escribir fuera de pantalla, saber cuántos caracteres caben en un determinado espacio, etc.), podríamos cargar la fuente al mismo tiempo que obtenemos información sobre ella mediante la función *XLoadQueryFont()*:

*XFontStruct *XLoadQueryFont(Display *display, XID font_ID);*

Esta función carga la fuente especificada por el nombre *font_ID* (de tipo *font*) y devuelve una estructura de tipo *XFontStruct* que contiene la siguiente información:

```
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
```

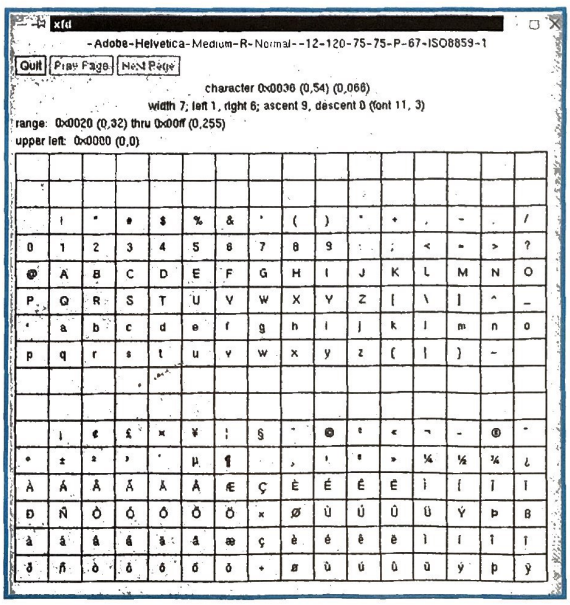

Figura 1. Aspecto de las fuentes de texto con 'xfd'.

*last_char information */*
int ascent; / log. extent above baseline for spacing */*
int descent; / log. descent below baseline for spacing */*
} XFontStruct;

Otra posibilidad hubiese sido utilizar la función *XQueryFont()* para obtener de una fuente cargada con *XLoadFont()* los mismos parámetros que en *XLoadQueryFont()*. La información de estos parámetros de fuente, así como otras funciones de interés están disponibles en la página *man* de *XLoadQueryFont()* y son las siguientes:
*XFontStruct *XQueryFont(Display *display, XID font_ID);*
*XFreeFont(Display display, XFontStruct *font_struct);*
*Bool XGetFontProperty(XFontStruct **font_struct, Atom atom, unsigned long *value_return);*
*XUnloadFont(Display *display, Font font);*
Lo fundamental es que con la estructura de información que nos devuelve *XLoadQueryFont()* es posible obtener la anchura de un determinado texto en pixels mediante la siguiente función:

*int XTextWidth(XFontStruct *font_struct, char *string, int count);*

donde *font_struct* es la estructura de fuente que nos devolvió al cargarla; *string* es la cadena cuya anchura en pixels queremos conocer, y *count* es el número de caracteres de la cadena. El valor de retorno será la anchura en pixels de *string*. En principio, si siempre vamos a utilizar una determinada fuente (9x15, 8x16, etc.) no será necesario el uso de estas funciones y bastará la carga de la fuente mediante *XLoadFont()*. Nótese también que podemos cargar todas las fuentes que nos sean necesarias y, posteriormente, antes de imprimir en pantalla cada cadena, indicar en el GC qué fuente debe utilizarse para el trazado.

Las fuentes se cargan en memoria mediante 'XLoadFont()'

DIBUJO DE TEXTO

Al igual que se hace con el resto de primitivas gráficas, antes de trazar el texto en pantalla hemos de especificar en el GC las características del mismo (color, fondo, etc.), así como la fuente a utilizar:

*XSetFont(Display *display, GC gc, Font fuente);*

Esta función establece el atributo de fuente del GC indicado como la fuente especificada (el ID devuelto por *XloadFont*). Si cargamos la fuente con *XLoadQueryFont()* disponemos del ID en el campo *fid* de la estructura devuelta. Tras esto ya podemos utilizar dicho GC para trazar texto en pantalla con las siguientes funciones:

*XDrawString(Display *display, GC gc, Drawable drawable, int x, int y, char *string, int length);*

Esta función escribe el texto *string*, de longitud *length*, en el *drawable* especificado (ID de ventana o de pixmap) con los atributos especificados en GC (color, fondo, fuente) y situando el píxel base del primer carácter de la frase en las coordenadas x,y del *drawable*. Los pixels de fondo de la cadena no se dibujan, es decir, las letras son transparentes y no borran el fondo sobre el que son dibujadas.

*XDrawImageString(Display *display, GC gc, Drawable drawable, int x, int y, char *string, int length);*

Realiza la misma función que *XDrawString()* pero traza los pixels del fondo de las letras con el color de fondo del GC, sobreescribiendo el fondo del *drawable* y dibujando, de este modo, pixmaps cuadrados.

Pueden obtenerse las características de la fuente al cargarla mediante 'XLoadQueryFont()'

Además de estas dos funciones también puede hacerse uso de *XDrawText()*, que hace posible dibujar texto con varias fuentes (de la misma forma que se podían dibujar varias líneas o pixels simultáneamente). También se dispone de las siguientes funciones para el uso de caracteres internacionales de 16 bits:

*XDrawString16(Display *display, GC gc, Drawable drawable, int x, int y, XChar2b *string, int length);*
*XDrawImageString16(Display *display, GC gc, Drawable drawable, int x, int y, XChar2b *string, int length);*

LISTADO 1. Trazado de texto en pantalla

```
/* Ejemplo1.c -> Trazado de texto en pantalla          */
/* gcc -o ejemplo1 ejemplo1.c -lX11 -L/usr/X11R6/lib   */
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <stdio.h>

unsigned long ColorPorNombre( Display *, char *);

/*-----------------------------------------------*/
void main()
{
  Display *display;
  Window window;
  GC gc;
  Font fuente;
  char cadena[] = "Esto es una prueba";

  display = XOpenDisplay( NULL );
  window = XCreateSimpleWindow( display,
          DefaultRootWindow( display ),
          100, 100, 300, 300, 1, 0, 1 );
  XMapWindow( display, window );

  gc = XCreateGC( display, window, 0, NULL );
  fuente = XLoadFont( display, "9x15" );
  XSetForeground(display, gc, ColorPorNombre(display, "green"));
  XSetFont( display, gc, fuente );
  XDrawString( display, window, gc, 50, 100, cadena, strlen( cadena ) );

  XFlush( display );
  getchar();
}

/*--- Rutina para localizar colores por nombre ----------*/
unsigned long ColorPorNombre( Display *display, char *Nombre )
{
  XColor color, temp;
  XAllocNamedColor( display,
     DefaultColormap(display,DefaultScreen(display)),
     Nombre, &color, &temp );
  return( color.pixel );
}
******** FIN LISTADO
```

En el Listado 1 (ver su salida en la Figura 2) disponemos de un ejemplo que muestra texto en pantalla utilizando las funciones vistas anteriormente. Si se hubiese querido que el texto fuera de color verde sobre fondo azul (en vez de ser de fondo transparente), podría haberse hecho lo siguiente:

```
color = ColorPorNombre( display, "green" );
fondo = ColorPorNombre( display, "blue" );
XSetFont( display, gc, fuente );
XSetForeground( display, gc, color );
XSetBackground( display, gc, fondo );
(...)
XDrawImageString( display, window, gc, 10,
10, cadena, strlen( cadena ) );
```

Mediante 'XTextWidth()' podemos obtener la anchura en pixels que ocupará una cadena

De esta manera ya somos capaces de trazar cualquier texto, sobreescribiendo texto anterior o haciéndolo transparente. Como gracias a los eventos podemos tomar entrada del teclado o ratón, cualquier tipo de programa que utilice texto puede hacerse con los conocimientos adquiridos hasta ahora, máxime cuando sabemos cómo dibujar sencillas primitivas gráficas que podrían ser utilizadas para dibujar botones, fáciles menús y similares (posteriormente lo haremos con *widgets* de librerías de más alto nivel).

MANEJO DE EVENTOS DE 'EXPOSE'

Una vez vistas las principales primitivas (tanto de texto como gráficas), hemos de pasar a ver la forma en que los eventos gráficos de la ventana deben procesarse.

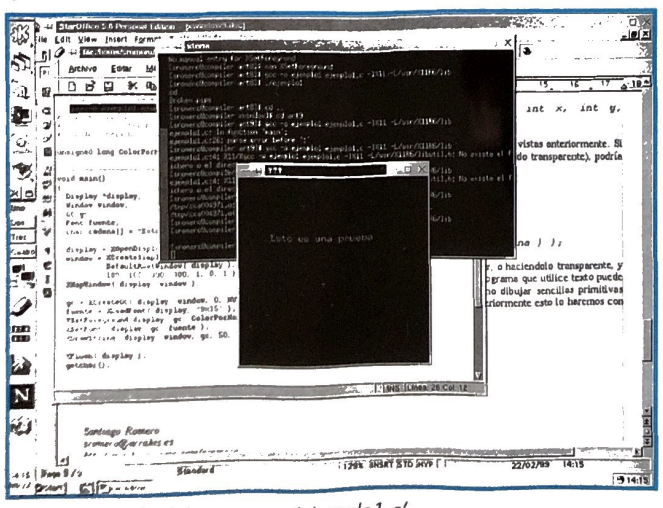

Figura 2. Salida del programa 'ejemplo1.c'.

Fijémosnos en todos los ejemplos vistos hasta ahora que tracen gráficos en pantalla: si en esos programas cambiamos de ventana (o movemos una ventana sobre ésta) y volvemos a cambiar de nuevo al ejemplo, veremos que parte de los gráficos (si la hemos solapado en parte) o toda la ventana (si cambiamos de escritorio) han desaparecido. Esto es debido a que al tapar nuestra ventana hemos perdido la información gráfica que había en ella y que debemos recuperar cuando ésta vuelva a pasar a primer plano (Figura 3).
Resulta pues necesario el almacenamiento de los contenidos de la ventana para poder restaurarlos cuando sea necesario gracias al evento *Expose* (que luego más tarde se denominó en los entornos Windows como mensaje *WM_PAINT*). Éste nos avisará de cuándo es necesario redibujar nuestra ventana y qué parte de la misma es necesario redibujar. Estos eventos se producen cuando una parte de la ventana que estaba tapada por otra se hace visible (o la ventana entera), o cuando se mapea una ventana, etc. Entonces nos proporciona una oportunidad de restaurar los contenidos de la misma para que se disponga de la información gráfica actualizada.

'XDrawString()' dibuja letras transparentes mientras que 'XDrawImageString()' las traza de forma opaca

Con el fin de recibir este tipo de eventos es necesario haber especificado mediante *XSelectInput()* la máscara *ExposureMask* (junto con otras que se necesiten, como las de teclado, ratón, etc.):

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

La estructura de evento utilizada es *XExposeEvent*, cuyos campos más importantes son los siguientes:

* *XExposeEvent.x* y *XExposeEvent.y* -> Coordenadas x,y de inicio del bloque que es necesario redibujar.
* *XExposeEvent.width* y *XExposeEvent.height* -> Anchura y altura del bloque a redibujar.
* *XExposeEvent.count* -> Contador de *Expose*.

La gestión de los eventos *Expose* es, en principio, bastante sencilla: por cada porción de la ventana que deba ser restaurada (áreas rectangulares) se recibirá un evento *Expose* para que dibujemos la sección rectangular de dibujo que debe ser retrazada. La sección rectangular quedará definida por los campos *(x,y)* y *(x+width,y+height)*.
La manera de restaurar los gráficos (saber qué datos hay que dibujar en cada área) puede abordarse de diferentes formas. Por una parte, cada vez que recibamos un *Expose* podemos ignorar el área a redibujar y volver a dibujar la ventana completa (este proceso es lento, pero idóneo si nuestro programa no requiere velocidad o no hace un uso intensivo de los gráficos). Por otra parte, también es posible (incluso recomendable) tener siempre en memoria (en un pixmap de la misma profundidad de color que la ventana) una copia de toda la ventana, de forma que cuando nos llegue un evento *Expose* podamos usar las funciones de copia de áreas (*XCopyArea()*) de Xlib para restaurar el trozo de ventana necesario. Esto hace el refresco de la pantalla mucho más rápido (sólo restauramos aquello que es necesario). Si hubiera que restaurar varios rectángulos de la ventana, recibiríamos entonces varios eventos *Expose*, uno por cada sección rectangular a retrazar. Si la zona no fuese estrictamente rectangular, el servidor nos generaría los *Expose* necesarios (descomponiendo la zona a redibujar en rectángulos) para retrazar toda la zona necesaria. Este número de rectángulos o eventos Expose son contados en el campo *count* de la estructura *XExposeEvent*. Este campo irá decreciendo desde n hasta 0 por cada evento *Expose* que recibamos, de forma que cuando leamos un 0 en él quiere decir que este *Expose* es el último de todos los rectángulos generados para redibujar las zonas necesarias.
En el fichero *ejemplo2.c* del CD puede verse un programa que tiene en cuenta los eventos de exposición para redibujar la ventana cuando sea necesario. Realice la prueba de tapar intencionadamente la ventana con otra ventana y observe cómo al quitar esta última la primitiva gráfica es redibujada. Si probamos a hacer lo mismo sin el evento *Expose* (coméntelo con /* y */ y recompile) puede verse cómo la porción de ventana que es solapada no es restaurada. La parte clave del programa es

la siguiente:

```
while ( 1 )
{
 XNextEvent( display, &evento );
 switch(event.type)
 {
  case Expose:
    XFillRectangle( display, window, gc, 20,
20, 260, 260 );
    break;
  case KeyPress:
    exit( 0 );
    break;
 }
}
```

Cabe decir que no es necesario dibujar la primitiva gráfica antes de entrar en el bucle de eventos pues, al mapear la ventana, como se comentó anteriormente, se genera un evento *Expose* para toda ella.

Es necesario un control del contenido de la ventana para restaurarla cuando sea necesario

En el anterior ejemplo, cada vez que se produce un evento *Expose* estamos redibujando toda la ventana, en vez de redibujar sólo la porción que nos indican los campos *x*, *width* y *height* de la estructura del evento (es decir, tenemos esta información disponible en *evento.xexpose.x*, *evento.xexpose.y*, etc.).

PIXMAPS PARA PANTALLAS VIRTUALES

La manera más eficiente de refrescar las partes de la ventana que deban ser retrazadas es el uso de pantallas virtuales. Consiste en crear un pixmap del mismo tamaño que la ventana y dibujar allí las primitivas gráficas, volcando luego todo el pixmap a pantalla. Cuando se produzca un evento *Expose*, simplemente se copia el área (*XCopyArea()*) que debe ser restaurada desde el pixmap a la ventana. Para poder hacerlo vamos a detenernos primero en la manera de crear, cargar y manipular pixmaps (funciones que además podrán utilizarse para multitud de tareas).
Un pixmap se crea de la siguiente manera:

```
Display *display;
unsigned int anchura, altura;
unsigned int profund_color;
Pixmap pixmap;
pixmap = XCreatePixmap( display, ventana,
anchura, altura, profund_color);
```

Esta función devolverá *NULL* en caso de error y *distinto* en caso de éxito. Tras la creación ya es posible utilizar el pixmap como *drawable* en cualquier función de dibujo, pudiendo trazar imágenes, texto, copiar áreas sobre y desde ella, etc. Como muestra lo primero que deberemos hacer será crear un GC para el pixmap con el

El evento 'Expose' nos avisará cuando sea necesario restaurar parte de una ventana

color *foreground* necesario para borrar el pixmap (por ejemplo con color negro) mediante una orden como la siguiente:

```
XFillRectangle( display, pixmap, gc, 0, 0,
anchura, altura );
```

Finalmente, cuando ya no se utilicen más los pixmaps (antes de salir del programa) se hace necesario liberar los datos emplazados en el servidor mediante la función *XfreePixmap (Display *display, Pixmap pixmap)*.
Por otra parte, con las funciones *XReadBitmapFile()* y *XWriteBitmapFile()* podemos cargar y grabar bitmaps al disco en ficheros Ascii con un formato determinado. Esto, en conjunción con la función *XCreateBitmapFromData()*, nos provee de suficientes herramientas para el uso de bitmaps.
Basándose en estas funciones y en *XCopyArea* se ha desarrollado el Listado 2 (ejemplo 3 en el CD), que utiliza un pixmap para almacenar los contenidos de la ventana y poder realizar el refresco con mucha sencillez y velocidad. Este programa incluye también algunas secciones de código que muestran el uso de los eventos como repaso (como el tratamiento de *ConfigureNotify* para saber en todo momento las coordenadas y dimensiones de la ventana), así como las nociones del uso de pixmaps y tratamiento de *Expose*.

SOLAPAMIENTOS EN COPIAS

Al realizar copias entre *drawables* (porciones de un pixmap a una ventana, o viceversa), es posible que mediante las funciones de copia habituales (ej: *XCopyArea()*)

'Expose' utiliza la estructura 'XexposeEvent'

podamos incurrir en un error si el área a copiar está tapada por otras ventanas (si dicho área destino está ocupado por una ventana o simplemente está oculto no debemos dibujar ahí). Para solucionarlo en Xlib se han definido las siguientes estructuras:

```
typedef struct {
 int type;
 unsigned long serial;
 Bool send_event;
 Display *display;
 Drawable drawable;
 int x, y;
 int width, height;
 int count;
 int major_code;
 int minor_code
} XGraphicsExposeEvent;

typedef struct {
```

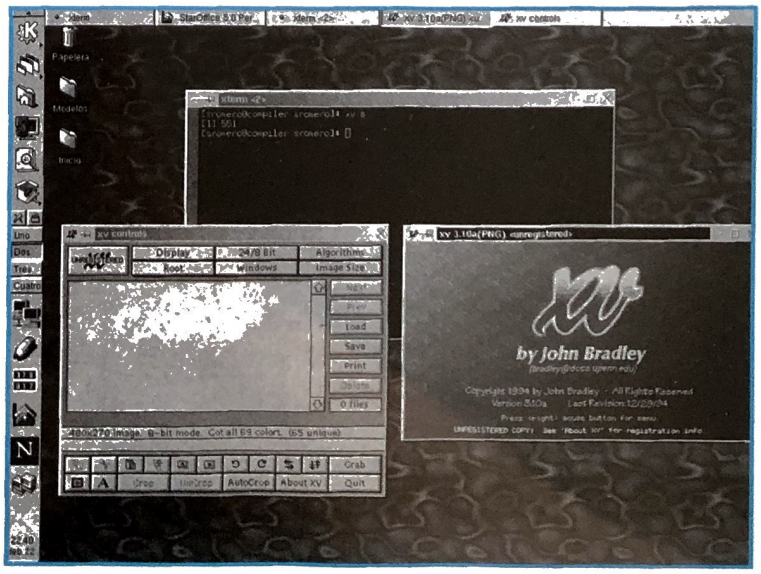

Figura 3. Solapamiento de ventanas (eventos 'Expose')

LISTADO 2. Gestión avanzada de 'expose', 'bitmaps' y 'eventos'

```
/* Ejemplo3.c -> Gestión avanzada de expose, bitmaps y eventos.    */
/* gcc -o ejemplo3 ejemplo3.c -lX11 -L/usr/X11R6/lib               */
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <stdio.h>

void Redibuja( Display *, Drawable, GC, Pixmap, int, int, int, int);
int XVentana, YVentana, AnchoVentana, AltoVentana;

/*-------------------------------------------------------------------*/
void main()
{
  Display *display;
  Window window;
  int pantalla;
  Pixmap pixmap;
  GC gc;
  XEvent evento;
  Font fuente;
  char cadena[] = "Uso de pixmaps.";

  display = XOpenDisplay( NULL );
  window = XCreateSimpleWindow( display,
         DefaultRootWindow( display ),
         100, 100, 300, 300, 1, 0, 1 );
  pantalla = DefaultScreen(display);
  XSelectInput( display, window, KeyPressMask|ExposureMask|StructureNotifyMask );
  XStoreName(display, window, "Gestion de expose.");
  XMapWindow( display, window );
  pixmap = XCreatePixmap( display, window, 300, 300,
                          DefaultDepth(display, pantalla));
  if( pixmap == (Pixmap) NULL )
  {
    printf("\nError creando Pixmap\n");
    XCloseDisplay(display);
    exit(1);
  }

  gc = XCreateGC( display, pixmap, 0, NULL );
  XSetForeground(display, gc, 0 );
  XFillRectangle( display, pixmap, gc, 0, 0, 300, 300 );

  fuente = XLoadFont( display, "9x15" );
  XSetForeground(display, gc, ColorPorNombre(display, "green"));
  XSetFont( display, gc, fuente );
  XDrawString( display, pixmap, gc, 50, 120, cadena, strlen( cadena ) );
  XSetForeground(display, gc, ColorPorNombre(display, "blue"));
  XDrawString( display, pixmap, gc, 20, 40, cadena, strlen( cadena ) );
  XSetForeground(display, gc, ColorPorNombre(display, "red"));
  XDrawString( display, pixmap, gc, 100, 240, cadena, strlen( cadena ) );
  XDrawRectangle(display, pixmap, gc, 10,10,280,280);

  while ( 1 )
  {
    XNextEvent( display, &evento );
    switch(evento.type)
    {
     case Expose:
       Redibuja(display, window, gc, pixmap,
              evento.xexpose.x, evento.xexpose.y,
              evento.xexpose.width, evento.xexpose.height );
       XFlush(display);
      break;
     case ConfigureNotify:
       XVentana = evento.xconfigure.x;
       YVentana = evento.xconfigure.y;
       AnchoVentana = evento.xconfigure.width;
       AltoVentana = evento.xconfigure.height;
       printf("Posición: %d,%d  y Dimensiones: %dx%d\n",
              XVentana, YVentana, AnchoVentana, AltoVentana );
      break;
     case KeyPress:
       XFreePixmap(display, pixmap);
       XCloseDisplay(display);
       exit( 0 );
       break;
    }
  }

}

/*— Función que redibuja la porcion de pantalla deseada ——*/
void Redibuja( Display *display, Drawable drawable, GC gc, Pixmap pixmap,
           int x, int y, int anchura, int altura  )
{
  XCopyArea(display, pixmap, drawable, gc, x, y, anchura, altura, x, y);
}
******** FIN LISTADO
```

```
int type;
unsigned long serial;
Bool send_event;
Display *display;
Drawable drawable;
int major_code;
int minor_code
} XNoExposeEvent;
```

Estas dos nuevas estructuras contienen los datos necesarios para dos nuevos eventos que se denoniman *GraphicsExpose* y *NoExpose*. El primero se recibe cuando el área que se va a copiar está tapada, mientras que el segundo se recibe en caso de que no lo esté. Para hacer posible la recepción de estos eventos no existe una máscara como en el caso de *ButtonPress* y similares, sino que se activan desde el atributo *GC.graphics_exposures* (1= envía eventos, 0= no envía). Gracias a ellos podemos

Es recomendable el uso de un pixmap como pantalla virtual desde la que restaurar los gráficos con 'XCopyArea()'

Las funciones *XCopyArea()* y *XCopyPlane()* pueden copiar un área específica de una ventana sobre un *dibujable*. Si dicho área está tapada por otras ventanas, la imagen no se puede copiar de forma completa. Para solucionar este problema se usan los eventos especiales *GraphicsExpose* y *NoExpose*. El evento *GraphicsExpose* se envía desde el servidor si el área que se copia está tapada parcial o totalmente por otras ventanas, en caso contrario se envía al evento *NoExpose*. Para más información se puede consultar la página *man* correspondiente. Hay que destacar que los campos tienen el mismo significado que en los eventos *Expose*, con la adición de *major_code*, que puede tomar los valores *X_CopyArea* o *X_CopyPlane*, dependiendo de la función que provocó dicho evento. También puede consultarse la página *man* de *XSetGraphicsExposures()*.

EN RESUMEN

Vistas las funciones básicas de Xlib, en la próxima entrega nos introduciremos en las extensiones X-Window, analizando varias de ellas que nos permitirán hacer programas gráficos de alta velocidad (como juegos o similares) con utilización de pantallas virtuales y acceso directo a la videomemoria.