TECNOLOGÍA: HACIA UNA INTERNET DE SERVICIOS INTEGRADOS

PROGRAMACIÓN ACTUAL

www.prensatecnica.com

995 pts.
PORTUGAL 990 ESC (CONT)
5,98 €

AÑO 3 • NÚMERO 25

+ de 600 Mb en el CD-Rom

NIVEL BÁSICO
Delphi: eventos y mensajes
Operaciones matemáticas en C++

LAS EMPRESAS DEMANDAN
Diagramas de casos de uso en UML
El entorno de ejecución de las páginas ASP
Servicios de impresión en NT

BAJO NIVEL
3D: consiga mayor velocidad con un nuevo engine
Contextos gráficos en Linux

CAMPUS ACTUAL
Configuración remota de una red
Creación de menús en el mIRC

EN EL CD-ROM

PROGRAMACIÓN
- DirectX 6.1 SDK
- Microsoft DirectX Media 6.0
- JDK 1.2 y documentación
- Dreamweaver
- Windows Registry Guide

HERRAMIENTAS INTERNET
- Internet Explorer 5
- Netscape Comunicator 4.51
- GetRight 3.3.3
- mIRC 5.51

WINDOWS
- ACDSee 2.4
- Paint Shop Pro 5.01

LINUX
- Kernel 2.2.3
- WordPerfect 8

Programación de VIDEOJUEGOS con DirectX

Prens@ Técnic
00025
8 413042 087779

Autor: Santiago Romero

Gráficos en XLIB (II)

Como ya se introdujo en nuestra anterior entrega, en X Window (y más concretamente en *Xlib*) disponemos de una serie de funciones gráficas elementales tales como el trazado de pixels, líneas, círculos, rectángulos, pixmaps, bitmaps, etc. Cada una de estas primitivas gráficas tiene una serie de atributos que lo definen, como puede ser el color y las coordenadas para el píxel, la anchura de la línea, o el patrón y color para una operación de rellenado. Todos estos atributos, parámetros al fin y al cabo, hemos de pasarlos a las funciones de dibujo para que éstas realicen la operación deseada con los resultados previstos. El estudio de este mes de los contextos gráficos cubre todo ese área de la programación de gráficos.

Los Contextos Gráficos (GCs) son estructuras de datos que definen la manera que X Window tiene de especificar los atributos (colores, formas, anchura, etc.) para las operaciones gráficas (líneas, rellenados, pixels) sobre ventanas y, en general, sobre cualquier superficie dibujable ("*drawables*").

CONTEXTOS GRÁFICOS

En lugar de pasar a cada función (ej: *XDrawLine*, *XDrawRectangle*, etc.) toda la lista de parámetros necesaria para definir la apariencia de dicha primitiva (el color, el tipo de rellenado, etc.), el camino que se tomó al diseñar X Window fue el de crear una estructura de datos (el contexto gráfico o *GC-Graphics Context*) que contenga todos los parámetros que pueden ser necesarios para todas las funciones gráficas de X-Lib. Esto significa que antes de llamar a una función gráfica se deberá crear un GC y rellenar sus diferentes campos con los valores apropiados (por ejemplo, su color a un valor determinado) antes de llamar a la primitiva gráfica a utilizar. Con esto se consigue que la llamada a la función se realice pasando un identificador a todos los datos en lugar de pasar todos los datos necesarios para el dibujado.

Tabla 1. Valores por defecto en la creación de un GC ("XGCValues")

Campo	Valor por defecto
plane_mask	1
foreground	0
background	1
line_width	0
function	GXcopy
line_style	LineSolid
cap_style	CapButt
join_style	JoinMiter
fill_style	FillSolid
fill_rule	EvenOddRule
arc_mode	ArcPieSlice
tile	pixmap relleno con fondo
stipple	pixmap de 1s
ts_x_origin	0
ts_y_origin	0
subwindow_mode	ClipByChildren
graphics_exposures	True
clip_x_origin	0
clip_x_origin	0
clip_mask	None

Los Contextos Gráficos (GCs) son estructuras que definen las propiedades de las primitivas gráficas

Esto no quiere decir que cada vez que realicemos una operación de dibujado tengamos que crear y rellenar un GC. Aunque esto es perfectamente posible, lo que se suele hacer en lugar de enviar un GC con cada petición de dibujado es enviar los atributos deseados al servidor, y enviar peticiones de dibujado utilizando dichos atributos, lo cual reduce mucho la sobrecarga (*overhead*) del sistema o la red. Como veremos, los GCs pueden crearse (lo hacen en el servidor) mediante *XCreateGC*, así como modificarse, de forma que podremos optar por crear varios GCs para diferentes primitivas de dibujo, o bien crear un sólo GC (es lo mínimo para utilizar funciones de dibujo) e ir cambiando sus valores antes de dibujar cada primitiva (función *XChangeGC* o funciones individuales). Esto es así porque cada vez que llamemos a una función que dibuje algo (en una ventana, en memoria, etc.) será necesario pasarle entre sus parámetros el GC que le indica al servidor cómo debe realizar el trazado.

Como ya hemos comentado, los GC se crean y almacenan en el servidor, de modo que cuando uno de ellos es creado, se nos da un número identificador (*identificador del GC*) que nos permitirá decirle al servidor cuál de todos los GC definidos allí queremos usar.

DIBUJANDO GRÁFICOS

A la hora de dibujar gráficos en cualquier recurso de X Window hay que seguir una serie de pautas que ya hemos introducido en los anteriores párrafos. Estas pautas son:

A) Obtener un GC a un recurso o crear un GC nuevo. Esto se puede hacer principalmente de tres formas. Por una parte se puede crear un nuevo GC con lo cual el servidor nos devolverá su ID para que lo usemos al referirnos a él. Esta es la opción más común, aunque también se puede obtener el ID del GC por defecto creado en el servidor X, o utilizar XCopyGC para obtener una copia de un GC ya creado.

B) Modificar los valores del GC para definir los atributos que deberá usar X Window cuando llamemos a la función gráfica adecuada.

C) Llamar a la función gráfica deseada pasándole el GC creado u obtenido.

CREACIÓN DE GCS

Los GCs se crearán asociados a un *drawable*, que definiremos como un recurso del servidor X Window donde pueden trazarse gráficos. Ejemplos de drawables son los pixmaps (y bitmaps), las ventanas (que en el

fondo son pixmaps), y similares. Este concepto lo utilizaremos a continuación en la llamada que se utiliza para la creación del GC, que normalmente asociaremos a la ventana donde queremos dibujar:

*GC XCreateGC(Display *display, Drawable drawable, unsigned long valuemask, XGCValues *values);*

Esta función hace que el servidor cree un nuevo contexto gráfico y que nos devuelva un valor entero que lo identifique, de forma que lo utilizaremos en las funciones de dibujo para que el servidor sepa cuál de toda la lista de GCs que posee es el que deseamos utilizar para el trazado de la primitiva gráfica pedida.

Los Contextos Gráficos se crean en el servidor mediante "XCreateGC()"

Mediante esta función no sólo se puede crear el GC sino que también pueden establecerse algunos de los valores de este atributo mediante el uso de *valuemask* y *values*, mediante los cuales le indicaremos qué atributos deseamos establecer y con qué valores. Si no nos interesa darle valores iniciales a este GC, se puede especificar 0 como *valuemask* y NULL como *values* de forma que el GC será creado con los valores por defecto del servidor (ver tabla 1).
El modo de actuación del servidor se basa en trazar la función gráfica *pedido* utilizando los atributos del contexto gráfico, lo cual afectará a determinados pixels del lugar donde se desea dibujar. Sobre estos pixels se aplicará la máscara de planos y la función lógica deseada (AND, XOR, o combinaciones de ellas como se podrá ver más adelante) entre los valores calculados y los existentes en el lugar destino. Los elementos que define el contexto gráfico son los siguientes:

- Colores: de primer plano (*foreground*) y de fondo (*background*). El primero indicará el color de la mayoría de operaciones gráficas (pixels, líneas, polígonos), mientras que el segundo se utilizará para el fondo de dichas primitivas (líneas discontinuas, fondos de los rellenados, etc.).
- Texto: fuente de texto a utilizar para operaciones de texto (*font*).
- Líneas: anchura (*line_width*), forma de los extremos (*cap_style*) y de las uniones (*join_style*), así como, obviamente, el estilo de la propia línea (*line_style*).
- Rellenados: estilo de rellenado (*fill_style*), que puede variar entre *sólido* (usando *foreground* como color de rellenado), *tile* (pixmap que se usa como patrón de relleno), *stipple* (se usa un patrón de relleno con pixels transparentes que no se dibujan) o *stipple opaco*, donde los pixels transparentes se dibujan con el color de fondo. Puede definir además el modo de rellenado de polígonos (*fill_rule*) o de arcos (*arc_mode*).
- Recorte: mediante *clip_ask* puede limitarse el trazado a una determinada área del *drawable* (recortando lo que quede fuera de ella, como ocurriría si parte de una imagen se saliera fuera de la pantalla, pero definiendo lo mismo en cualquier rectángulo de *drawable*).
- Función: permite especificar una función lógica (*function*) a aplicar entre los pixels que dibujamos y los ya existentes, para poder hacer XOR, AND, OR, etc., lo cual puede dar mucho juego a nuestras operaciones lógicas.

Nuestro próximo objetivo es saber cómo debemos indicarle a *XCreateGC()* o funciones similares qué valores deseamos para cada una de estas

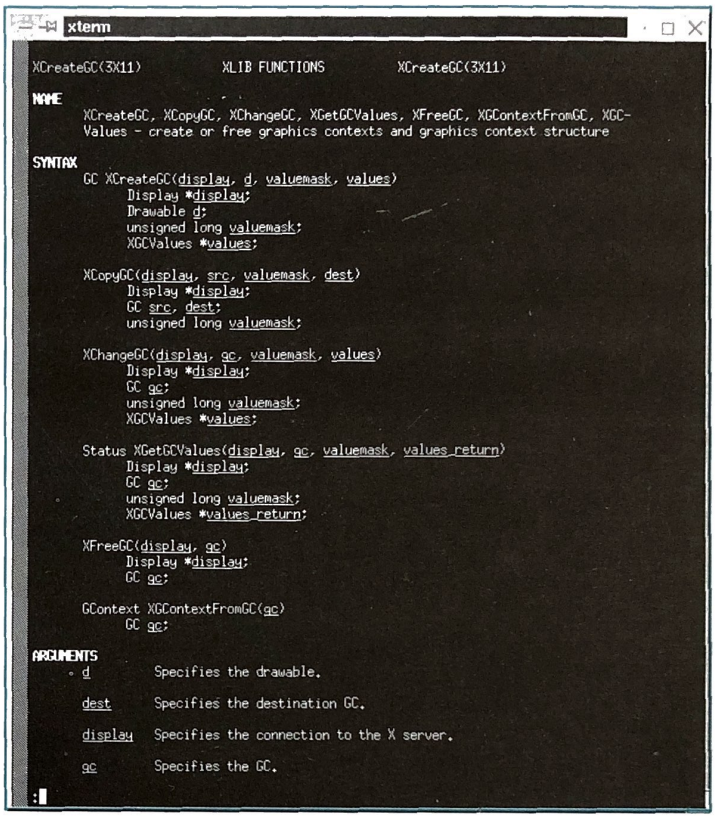

Figura 1. Página man de "XCreateGC()".

características (es decir, cómo cambiar el tipo de rellenado, cambiar el color de fondo o primer plano, etc.). Esto se conseguirá fácilmente mediante los dos últimos parámetros de la función, cuyo significado exacto analizaremos a continuación.

Los GCs se crean asociados a un "drawable" (dibujable)

En cualquier momento (por ejemplo, antes de salir del programa) podemos liberar los GCs previamente creados mediante la función *XfreeGC()*:

*XFreeGC(Display *display, GC gc);*

ESTRUCTURA GC - XGCVALUES

El GC almacena 23 atributos gráficos y todos ellos pueden especificarse o modificarse mediante una estructura de tipo *XGCValues*. Esta estructura tiene una serie de campos donde deberemos poner los valores deseados para los diferentes estilos y patrones, y que pasaremos a la función *XcreateGC* para que tome de ahí los valores para el GC que se va a crear. Como hemos dicho, no tienen porqué modificarse o establecerse todos los atributos del GC, sino que podemos especificar sólo los deseados gracias al parámetro *valuemask*. Los bits de este parámetro indicarán qué atributos del GC hemos puesto en *XGCValues* y queremos que sean utilizados. Cada bit del 0 al 22 indica un determinado parámetro, y si este bit está a 1 indica que queremos que el valor de dicho parámetro se lea desde la estructura *XGCValues* que le pasamos a la función, mientras que si está a 0 ese valor se tomará de la lista de valores por defecto. Podemos especificar uno o más parámetros mediante la operación lógica OR.
Para simplificar la modificación de la variable máscara existen unos *#define* en los ficheros de cabecera que facilitan enormemente la tarea de especificación de atributos, pues

es más sencillo acordarse de estas constantes que del bit asociado a cada una de ellas:

```
#define GCFunction      (1L<<0)
#define GCPlaneMask     (1L<<1)
#define GCForeground    (1L<<2)
#define GCBackground    (1L<<3)
#define GCLineWidth     (1L<<4)
#define GCLineStyle     (1L<<5)
#define GCCapStyle      (1L<<6)
#define GCJoinStyle     (1L<<7)
#define GCFillStyle     (1L<<8)
#define GCFillRule      (1L<<9)
#define GCTile          (1L<<10)
#define GCStipple       (1L<<11)
#define GCTileStipXOrigin
(1L<<12)
#define GCTileStipYOrigin
(1L<<13)
#define GCFont          (1L<<14)
#define GCSubwindowMode
(1L<<15)
#define GCGraphicsExposures
(1L<<16)
#define GCClipXOrigin   (1L<<17)
#define GCClipYOrigin   (1L<<18)
#define GCClipMask      (1L<<19)
#define GCDashOffset    (1L<<20)
#define GCDashList      (1L<<21)
#define GCArcMode       (1L<<22)
```

En la tabla 2 podemos ver la estructura XGCValues completa, y en la tabla 3 la lista de funciones para el parámetro *function* de dicha estructura. Ejemplo:

```
Display *display;
Window window;
GC gc;
XGCValues valores;

(... codigo de inicializacion, etc ...)

valores.foreground = WhitePixel
(display, pantalla);
valores.line_width = 10;

gc = XCreateGC( display, window,
                GCForeground |
                GCLineWidth,
                &valores );
```

Como puede verse en este ejemplo, hemos creado un GC (*XCreateGC*) preparado para dibujar en el *drawable window* (la ventana principal del programa), donde queremos que el color de primer plano sea el blanco y la anchura de línea de 10 unidades. El resto de valores deberá ser los que pone por defecto el servidor.

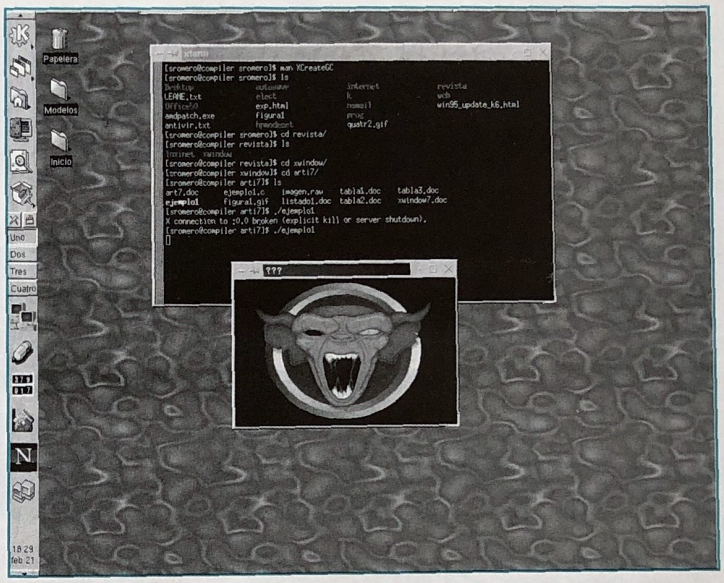

Figura 2. Salida del programa "ejemplo1.c".

Los valores de los contextos gráficos pueden modificarse mediante "XChangeGC()"

Lo que se ha hecho ha sido crear una estructura de tipo *XGCValues* donde hemos modificado los dos campos deseados, y luego mediante la máscara de *(GCForeground | GCLineWidth)* se han puesto los 2 bits correspondientes a 1 de forma que el servidor ha tomado los valores de color de primer plano y de ancho de línea de *XGCValues*, y el resto de los valores por defecto. Si ahora utilizamos una función de dibujar líneas con este GC, estaremos escribiendo en la ventana líneas de color blanco y de anchura 10. Este es el significado y funcionamiento de los GCs: la especificación del formato de las primitivas de dibujo.

CAMBIANDO GCS

Una vez hemos creado un GC en cualquier momento podemos cambiar sus valores (y por tanto el modo en que dibujamos en pantalla) mediante la función *XChangeGC*, que vuelve a hacer uso de la máscara *valuemask* y de la estructura *XGCValues* para especificar qué atributos deseamos cambiar y a qué valores queremos hacerlo.

```
XchangeGC ( Display *display,
    GC gc, unsigned long
mascara,
    XCGValues *valores );
```

Como puede verse en el siguiente ejemplo, su uso es muy similar al de la función *XCreateGC()*:

```
valores.foreground =
BlackPixel(display, pantalla);
valores.line_width = 5;

gc = XChangeGC( display, gc,
    GCForeground | GCLineWidth,
    &valores );
```

Aparte de disponer de *XChangeGC()* también pueden utilizarse una serie de funciones individuales que permiten cambiar un sólo atributo sin necesidad de utilizar *XChangeGC* ni de crear y modificar estructuras como las anteriores. Ejemplos de este tipo de funciones son las siguientes:

```
XSetForeground( Display *display,
GC gc, unsigned long pixel );
```

Tabla 3. Campos de la estructura "XGCValues"

```
typedef struct {
    int function; /* operación lógica a realizar */
    unsigned long plane_mask; /* máscara del plano */
    unsigned long foreground; /* color de 1er plano */
    unsigned long background; /* color de fondo */
    int line_width; /* ancho de la línea */
    int line_style; /* Estilo de línea */
    int cap_style;
    int join_style;
    int fill_style; /* Estilo de rellenado */
    int fill_rule;
    int arc_mode; /* Tipo de arco */
    Pixmap tile; /* pixmap en operaciones tile */
    Pixmap stipple; /* pixmap en operaciones stipple */
    int ts_x_origin; /* desplazamiento en ambas */
    int ts_y_origin;
    Font font; /* fuente de texto por defecto */
    int subwindow_mode;
    Bool graphic_exposures; /* Indica si debe haber EXPOSE */
    int clip_x_origin; /* origen del clipping */
    int clip_y_origin;
    Pixmap clip_mask; /* máscara de recorte */
    int dash_offset;
    char dashes;
} XGCValues;
```

```
XSetBackground( Display *display, GC gc,
    unsigned long pixel );
```

Estas dos funciones permiten modificar el color de primer plano y de fondo del GC que se le pasa como parámetro. El color es especificado mediante un *unsigned long* que suele ser resultado de operaciones de búsqueda de color como las vistas en nuestra primera entrega de gráficos en X-Window.
Estos colores (primer plano y fondo) afectarán al trazado de figuras, líneas (continuas y discontinuas), rellenados (*stipples*), texto, texturas, etc.

Los "drawables" son todos aquellos recursos susceptibles de ser dibujados (ventanas, pixmaps, etcétera.)

En cuanto el GC tiene el valor que se quiere para la operación ya puede utilizarse la función gráfica deseada:

```
XDrawLine( display, window, gc, 10, 20, 100, 110 );
```

Esto dibujaría una línea desde (10,20) hasta (100,110) en la ventana apuntada por *window* con los atributos (color, ancho de línea, etc.) especificados en el contexto gráfico *gc*.

COPIA DE GCS

Si queremos obtener una copia de los atributos (de nuevo varios o todos) de un determinado GC puede hacerse mediante una simple llamada a la función *XcopyGC()*, cuyos parámetros son los siguientes:

```
XCopyGC(Display *display, GC src,
    unsigned long valuemask, GC dest);
```

Con esta función se copian determinados atributos del GC SRC en DEST, y de nuevo gracias a *valuemask* podemos copiar todos los atributos o bien sólo algunos de ellos, dejando el resto inalterado.

LISTADO 1. EJEMPLO DE TRAZADO DE PIXELS

```
/* Ejemplo1.c -> Manejo de atributos gráficos */

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <strings.h>

unsigned long Color ( Display *,
    unsigned short, unsigned short,
    unsigned short, Colormap );
void CargaImagen( void );

Display *display;
Window window;
GC gc;
XEvent evento;
int pantalla;
char paleta[768], *buffer;

main()
{

 display = XOpenDisplay( NULL );
 pantalla = DefaultScreen( display );

 window = XCreateSimpleWindow( display,
    DefaultRootWindow( display ), 350, 0, 320,
      200, 1, WhitePixel(display,pantalla),
      BlackPixel(display, pantalla) );

XSelectInput( display, window,
    ButtonPressMask | KeyPressMask );

XMapWindow( display, window );

gc = XCreateGC( display, window, 0, NULL );
CargaImagen();

 while( 1 )
 {

  XNextEvent( display, &evento );
 switch( evento.type )
 {
     case ButtonPress: exit(0);
        break;
   case KeyPress:    exit(0);
        break;
   }
 }
 }

void CargaImagen( void )
{
  FILE *fp;
  int x, y;

  unsigned long pix;

  fp=fopen("imagen.raw", "rb");

  if( fp==NULL )
  {
    printf("\nError cargando
       fichero imagen.raw\n");
    exit(1);
  }

  buffer = (char *) malloc(64000);
  if( buffer == NULL )
  {
    printf("\nError asignando memoria
       para imagen.raw\n");
    exit(1);
  }

  fread( &paleta, 1, 768, fp );
  fread( buffer, 1, 64000, fp );
  fclose(fp);

  for(y=0; y<200; y++) {
    for(x=0; x<320; x++) {
     pix=Color ( display,
       paleta[(buffer[y*320+x]*3)]* (65536/64),

       paleta[(buffer[y*320+x]*3)+1]*(65536/64),

       paleta[(buffer[y*320+x]*3)+2]*(65536/64),
            DefaultColormap( display, pantalla
));
       XSetForeground( display, gc, pix);

       XDrawPoint( display, window, gc, x, y);
    } }

  free(buffer);
}

unsigned long Color (
    Display *display,
    unsigned short red,
    unsigned short green, unsigned short blue,
    Colormap mapa )
{
  XColor color;

  color.red=red;
  color.blue=blue;
  color.green=green;
  XAllocColor( display, mapa, &color );

 return( color.pixel );
 }
```

Tabla 2. Posibles valores para el campo "function" de "XGCValues" y su significado

Nombre Función	Valor	Operación
GXclear	0x0	0
GXand	0x1	src AND dst
GXandReverse	0x2	src AND NOT dst
GXcopy	0x3	src
GXandInverted	0x4	(NOT src) AND dst
GXnoop	0x5	dst
GXxor	0x6	src XOR dst
GXor	0x7	src OR dst
GXnor	0x8	(NOT src) AND (NOT dst)
GXequiv	0x9	(NOT src) XOR dst
GXinvert	0xa	NOT dst
GXorReverse	0xb	src OR (NOT dst)
GXcopyInverted	0xc	NOT src
GXorInverted	0xd	(NOT src) OR dst
GXnand	0xe	(NOT src) OR (NOT dst)
GXset	0xf	1

ALGUNAS FUNCIONES GRÁFICAS

Aunque el grueso total de funciones gráficas las veremos en la próxima entrega, este mes vamos a introducir algunas de ellas para que el lector pueda ir practicando mediante algún programa simple que utilice todo lo visto hasta ahora (como el listado de ejemplo de este mes).

```
XDrawPoint( Display *display,
    Drawable drawable, GC gc, int x, int y )
```

Dibuja un punto del color especificado en *GC.foreground* en las coordenadas (x,y) del *drawable* indicado (por ejemplo una ventana). Existe una función similar llamada *XDrawPoints()* que permite dibujar *n* puntos simultáneamente:

```
XDrawPoints( Display *display,
    Drawable drawable, GC gc,
    XPoint *puntos, int numpuntos,
    int modo );
```

A esta función se le pasa (aparte del GC, *display* y *drawable*), un array de puntos contenidos en una estructura de tipo *XPoint*, el número de puntos a dibujar, y el *modo* de dibujado, que puede variar entre *CoordModeOrigin* (coordenadas absolutas tomando (0,0) como la esquina de la ventana, mientras que con el modo *CoordModePrevious* el nuevo origen de coordenadas es la posición del punto anterior (coordenadas relativas). La estructura *XPoint* tiene la siguiente definición:

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

EJEMPLO

El ejemplo de este mes simplemente abre un fichero de gráficos llamado *imagen.raw*, en formato 320x200 a 256 colores, y de él extrae su paleta y el cuerpo de la imagen, utilizando *XDrawPoint* y los GC para trazar la imagen en pantalla. Este ejemplo es simplemente ilustrativo, no ha sido optimizado para facilitar su lectura, de modo que su velocidad de representación será muy baja, pero demuestra cómo pueden usarse los GCs y para modificar los diferentes pixels de una ventana. Más adelante veremos técnicas para obtener velocidades similares a las de los juegos profesionales, pero por ahora este ejemplo es perfectamente ilustrativo sobre el uso de GCs, aunque el uso que hace de las primitivas gráficas es bastante rústico (trabajar con funciones tipo *PutPixel()* puede resultar muy muy lento en contraposición a trabajar directamente con los buffers de dibujo antes de volcarlos a pantalla). Las partes principales de este ejemplo son:

```
gc = XCreateGC( display, window, 0, NULL );
```

Crea un GC con sus valores por defecto. Este GC será utilizado más adelante.

```
    fread( &paleta, 1, 768, fp );
    fread( buffer, 1, 64000, fp );
```

Tras la apertura del fichero, leemos en memoria la paleta gráfica (*colormap*) y el cuerpo de la imagen (los valores de los pixels dentro de la paleta, como se comentó en la primera entrega de gráficos en X Window).

```
for(y=0; y<200; y++)
{
  for(x=0; x<320; x++)
{

    pix=Color( display,
paleta[(buffer[y*320+x]*3)]*(65536/64),
paleta[(buffer[y*320+x]*3)+1]*(65536/64),
paleta[(buffer[y*320+x]*3)+2]*(65536/64),
DefaultColormap( display, pantalla ));
    XSetForeground( display, gc, pix);
    XDrawPoint( display, window, gc, x, y);
    }
}
```

Con esto dibujamos los 64.000 puntos de la imagen (es una imagen de 320x200 pixels) para lo cual cogemos cada color, buscamos el color más parecido disponible en el *ColorMap* por defecto (función *Color*()), ponemos como color de dibujado del GC el color encontrado, y dibujamos el píxel en la posición deseada mediante *XDrawPoint()*. Gracias al bucle esto se hace para los 64.000 pixels con los que obtenemos la imagen completa en la ventana.

El estado (0/1) de los bits de "ValueMask" especifica los atributos que deseamos modificar

Dicho de otro modo, creamos una ventana con el mismo tamaño en pixels que la imagen, de modo que cada píxel del fichero se corresponderá con un píxel concreto de la ventana. Lo que hacemos es leer cada píxel y preparar el GC para que al dibujarlo éste tenga el color que le corresponde. Como la imagen es de 256 colores, tiene una paleta que define las componentes RGB de cada uno de estos colores, y es por eso que no utilizamos el valor leído del fichero directamente como un píxel, sino que éste representa una referencia a la tabla de 256 componentes RGB que es la paleta. Esta paleta se almacena en el fichero (como comentamos en las imágenes de 256 colores en nuestra primera entrega de gráficos) en formato de 0 a 63, mientras que nuestra función de búsqueda de colores (y X Window) esperan valores de 0 a 65536, de modo que lo que hacemos es multiplicar por 65536/64 para escalarlas antes de llamar a la función *Color*. Tras esto dibujamos el píxel con el color que nos devuelve esta función y pasamos al siguiente píxel.

En resumen

En nuestro siguiente capítulo analizaremos la mayoría de funciones gráficas disponibles en *Xlib* (líneas, polígonos, arcos, texto, etc.), así como también un análisis de la necesidad y modo de gestión del evento de *EXPOSE* para la organización del código en los programas en X Window.

Santiago Romero
sromero@arrakis.es
http://members.xoom.com/sromero