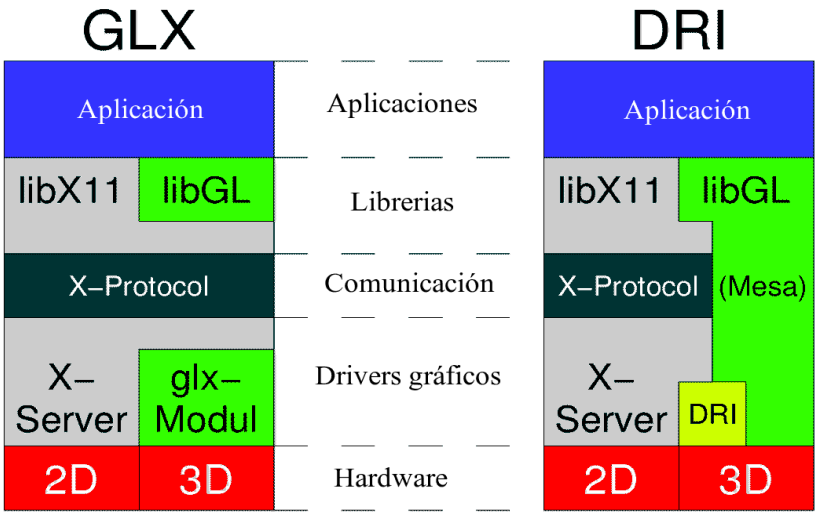
Tema 3

3.1. GLX

Son las iniciales en inglés de Extensión de OpenGL para el Sistema de Ventanas X. Fue desarrollado por Silicon Graphics. En el momento de la redacción de este trabajo se encuentra en la versión 1.4. Forma parte del servidor gráfico de la fundación X.Org desde la versión X11R6.7.0. GLX proporciona los enlaces necesarios para conectar las X con OpenGL, permitiendo que los programas incorporen llamadas a OpenGL dentro de una ventana creada el servidor gráfico y manejada por el gestor de ventanas. Por ejemplo, GLUT, en su implementación para GNU/Linux usa GLX.

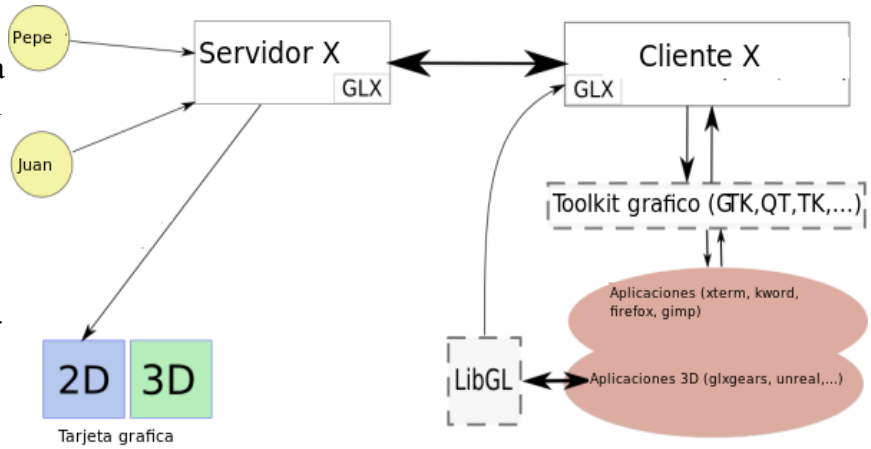


Esquema del funcionamiento de GLX frente DRI

GLX consiste fundamentalmente en tres partes de muy bajo nivel.
Por un lado proporciona un API con funciones de OpenGL a una aplicación X, para que esta pueda abrir contextos OpenGL en su ventana.
En segundo lugar, una extensión del X protocol, que permite al cliente (la aplicación OpenGL) enviar instrucciones de renderizado al servidor X. Como uno se puede imaginar, esto es lento, pues el trabajo es realizado, en parte, por el procesador. De mejorar esto se encarga AIGLX, que permite

que el cliente envie las instrucciones de renderizado directamente a la librería OpenGL.
Por último, GLX nos provee de una extension del propio servidor X que recibe las instrucciones de rendering del cliente y se las pasa a la libreria OpenGL instalada. Esta extensión será saltada por AIGLX.

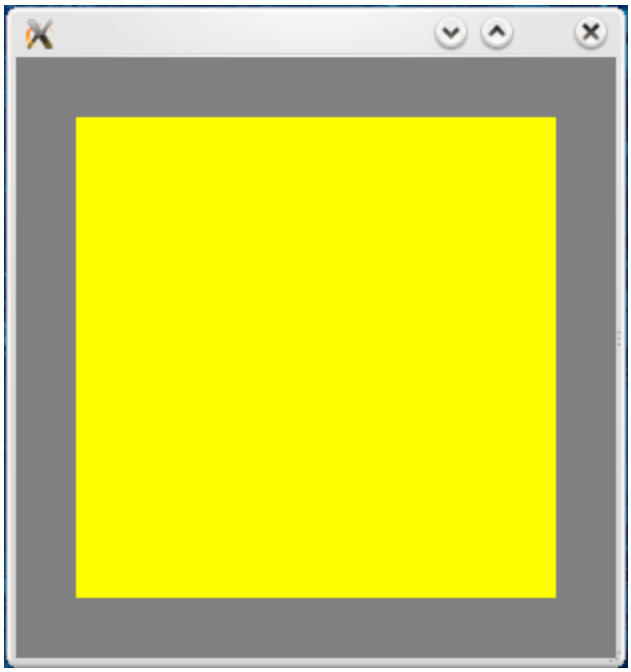
Además si disponemos de una tarjeta gráfica con aceleración 3D y del driver apropiado, la extensión del X protocol y de las X, pueden ser obviadas usando DRI (Direct Rendering Infraestructure), permitiendo al cliente acceder directamente al hardware gráfico.



Ni ATI ni nVidia usan la implementación de GLX que trae X.Org. Han desarrollado las suyas propias, que incluyen en sus drivers privativos. De ahí los problemas que surgen cuando, por ejemplo, no funcionan las extensiones de GLX como deberían, suponiendo claro que estas existan en el driver. En el caso de los drivers libres de los que dispone X.Org, como los que proporcionan

fabricantes como Intel, usan la implementación estándar de GLX lo que facilita el desarrollo de aplicaciones que hagan uso de GLX. De hecho, el primer hardware sobre el que funcionaron los gestores de ventanas con composición fue sobre las tarjetas Intel y aquellas ATI que funcionan bajo el driver libre de X.Org.
En el caso de no disponer de aceleracción 3D el renderizado se realiza a través de Mesa, la implementación de OpenGL que trae el servidor X de X.Org.

Ejemplo de código que usa GLX, obtenido de glxdemo.c, disponible en el paquete mesa-utils. Dibuja un cuadrado amarillo en una ventana. Es apreciable la complejidad de usar una librería de tan bajo nivel como GLX en lugar de una abstracción como GLUT.



```
#include <GL/gl.h>
#include <GL/glx.h>
#include <stdio.h>
#include <stdlib.h>


static void redraw( Display *dpy, Window w )

{
   printf("Redraw event\n");

   glClear( GL_COLOR_BUFFER_BIT );

   glColor3f( 1.0, 1.0, 0.0 );
   glRectf( -0.8, -0.8, 0.8, 0.8 );

   glXSwapBuffers( dpy, w );
}

static void resize( unsigned int width, unsigned int height )
{
   printf("Resize event\n");
   glViewport( 0, 0, width, height );
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity();
   glOrtho( -1.0, 1.0, -1.0, 1.0, -1.0, 1.0 );
}

static Window make_rgb_db_window( Display *dpy,
                                  unsigned int width, unsigned int height )
{
   int attrib[] = { GLX_RGBA,
                    GLX_RED_SIZE, 1,
                    GLX_GREEN_SIZE, 1,
                    GLX_BLUE_SIZE, 1,
                    GLX_DOUBLEBUFFER,
```

```
                                None };
  int scrnum;
  XSetWindowAttributes attr;
  unsigned long mask;
  Window root;
  Window win;
  GLXContext ctx;
  XVisualInfo *visinfo;

  scrnum = DefaultScreen( dpy );
  root = RootWindow( dpy, scrnum );

  visinfo = glXChooseVisual( dpy, scrnum, attrib );
  if (!visinfo) {
    printf("Error: couldn't get an RGB, Double-buffered visual\n");
    exit(1);
  }

  /* window attributes */
  attr.background_pixel = 0;
  attr.border_pixel = 0;
  attr.colormap = XCreateColormap( dpy, root, visinfo->visual, AllocNone);
  attr.event_mask = StructureNotifyMask | ExposureMask;
  mask = CWBackPixel | CWBorderPixel | CWColormap | CWEventMask;

  win = XCreateWindow( dpy, root, 0, 0, width, height,
                       0, visinfo->depth, InputOutput,
                       visinfo->visual, mask, &attr );

  ctx = glXCreateContext( dpy, visinfo, NULL, True );
  if (!ctx) {
    printf("Error: glXCreateContext failed\n");
    exit(1);
  }

  glXMakeCurrent( dpy, win, ctx );

  return win;
}

static void event_loop( Display *dpy )
{
  XEvent event;

  while (1) {
    XNextEvent( dpy, &event );

    switch (event.type) {
          case Expose:
            redraw( dpy, event.xany.window );
            break;
          case ConfigureNotify:
```

```
            resize( event.xconfigure.width, event.xconfigure.height );
            break;
        }
    }
}
```

Observando el código vemos muchas similitudes con GLUT, sin embargo toda la parte que este nos oculta sobre la inicialización de la ventana aquí aparece. Por no mencionar que dibujar un cuadrado con GLUT requeriría bastante menos código.

3.2. Enfoques para lograr la composicion de ventanas

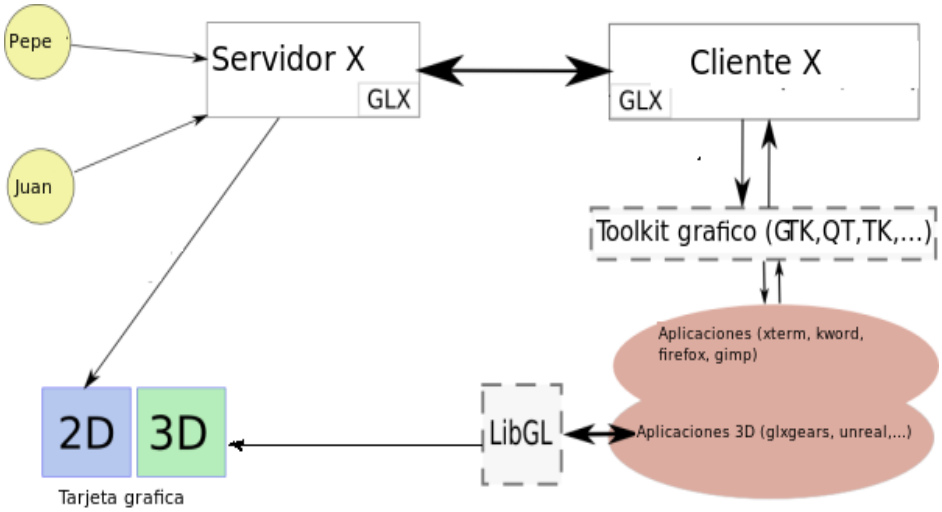
Hay dos formas de conseguir que un sistema de ventanas permita a OpenGL acceder a la tarjeta gráfica. La primera es especificar los comandos de OpenGL de una forma portable y neutral a la red usada, usando un enfoque similar al usado en el sistema de dibujado del protocolo X11. Este idea es indirecta, por el hecho de que las instrucciones de redenderizado se envian al servidor X y este a la tarjeta gráfica. Esto aumentaría la carga del procesador y provocaría que el renderizado fuera muchísimo más lento. El desarrollo de AIGLX solucionó estos problemas.

La segunda forma, que es la base de Xgl, consiste, en abrir una nueva ventana, y entonces permitir que el programa que la controle envie a la libreria OpenGL comandos de renderizado para la tarjeta gráfica. En este caso, los gestores de ventanas lo que harían sería crear una ventana inicial sobre la que dibujarian el resto de ventanas.

3.2.1. AIGLX

En inglés, Accelerated Indirect GLX ("AIGLX"), es decir, GLX indirectamente acelerado. Es un proyecto de software libre fundado por Red Hat y la comunidad de Fedora para añadir capacidades de aceleración indirecta de GLX a X.Org usando DRI.

De modo más sencillo, dispondremos de aceleración 3D usando el protocolo GLX.
Para ello es necesario acelerar el camino indirecto de OpenGL. Este método es transversal a la implementación del servidor X, pero, sin embargo, nos facilita mucho poder capturar el stream de datos de la ventana. Esta captura es trasformada por la extensión de GLX *texture_from_pixmap* (de la que hablaremos luego) a una textura, lo cual nos permite aprovechar el hardware específico de la tarjeta gráfica para el manejo de estas, liberando al procesador de una gran carga. Luego, el gestor de composición de ventanas simplemente deberá manejar texturas realizando las llamadas que proporciona OpenGL.



Han sido necesarios varios parches para el servidor X de X.Org para conseguir AIGLX. Estos se encuentran disponibles desde la versión 7.1 y están incluidos por defecto en el código de X.Org.
El soporte de hardware en estos momentos es bastante bueno, sobre todo usando los drivers privativos de nVidia, así

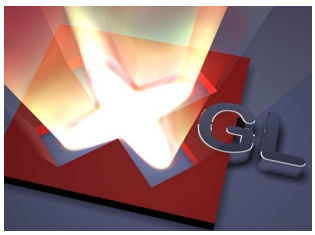
como los libres de Intel. El problema radica en que la implementación de GLX usada debe incorporar la extensión.

3.2.1.1 La extensión *GLX_EXT_texture_from_pixmap*

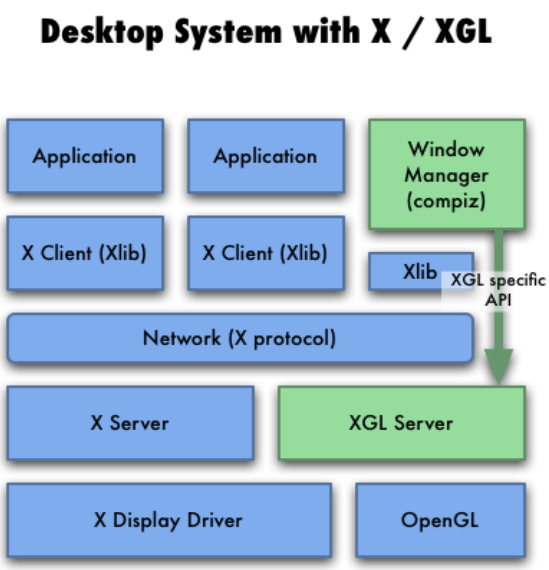
Esta extensión de GLX es la encargada de capturar los streams de dibujado que envian las aplicaciones al servidor X para ser tratados como texturas por GLX. De esta forma logramos aprovechar las capacidades 3D de la tarjeta gráfica, liberando al procesador de la carga que supone el dibujar y componer las ventanas.
Es necesaria pues, para los gestores de ventana con composición que usen AIGLX, como Compiz, Metacity o KWin4. Salida del comando $glxinfo mostrando la extensión *texture_from_pixmap*

juan@neboa:~$ glxinfo
name of display: :0.0
display: :0 screen: 0
direct rendering: Yes
server glx vendor string: NVIDIA Corporation
server glx version string: 1.4
server glx extensions:
GLX_EXT_visual_info, GLX_EXT_visual_rating, GLX_SGIX_fbconfig,
GLX_SGIX_pbuffer, GLX_SGI_video_sync, GLX_SGI_swap_control,
***GLX_EXT_texture_from_pixmap**, GLX_ARB_multisample, GLX_NV_float_buffer,*
GLX_ARB_fbconfig_float
client glx vendor string: NVIDIA Corporation
client glx version string: 1.4
client glx extensions:
GLX_ARB_get_proc_address, GLX_ARB_multisample, GLX_EXT_visual_info,
GLX_EXT_visual_rating, GLX_EXT_import_context, GLX_SGI_video_sync,
GLX_NV_swap_group, GLX_NV_video_out, GLX_SGIX_fbconfig, GLX_SGIX_pbuffer,
GLX_SGI_swap_control, GLX_NV_float_buffer, GLX_ARB_fbconfig_float,
*GLX_EXT_fbconfig_packed_float, **GLX_EXT_texture_from_pixmap,***
GLX_EXT_framebuffer_sRGB

3.2.2 Xgl



Xgl es una arquitectura de servidor X, aunque su implementación es una capa que se encuentra sobre OpenGL. Aprovecha las ventajas de las modernas tarjetas gráficas mediante sus controladores OpenGL, que soportan aceleración por hardware. Sin embargo, en el estado actual de desarrollo Xgl consiste en un backend llamado Xglx que requiere un servidor gráfico sobre el que ejecutarse. Esto causa una gran redundancia, por lo que se inició el desarrollo de un nuevo servidor gráfico diseñado especificamente para Xgl, llamado Xegl, aunque en estos momentos el desarrollo se encuentra parado.



El funcionamiento con Xglx es el siguiente:
En primer lugar se lanza el script de arranque de Xgl, este lanza el servidor X.Org. Una vez ahí, se enlaza Xgl a la librería de OpenGL en uso, creandose de esta forma un cliente OpenGL. Se solicita, entonces, un contexto directo de OpenGL, a través de GLX, para Xgl. Una vez asignado, se puede lanzar ya el servidor Xgl. Ahora ya se pueden lanzar otros clientes OpenGL, como Compiz, que obtendrán contextos indirectos OpenGL, salvo que dispongamos de drivers con soporte DRI y enlacemos manualmente las aplicaciones a través de este con la librería OpenGL en uso.

3.3 Comparación entre XGL y AIGLX

Nos encontramos ante tecnologias bastante similares en cuanto a rendimiento. Si bien es cierto que AIGLX tiene un mayor consumo de CPU, Xgl lo tiene de memoria, ya que, como dijimos, ejecuta un servidor X encima de otro.
Ademas el primer enfoque tiene la ventaja de requerir solo una relativamente pequeña extensión de GLX frente a tener que crear un servidor gráfico entero como en el caso de Xgl. Además usando AIGLX todo puede ser acelerado, sin embargo, en la implementación actual de Xgl, el hecho de saltarnos DRI provoca que perdamos esa posibilidad.
A esto hay que sumarle que AIGLX tiene la posibilidad de poder cambiar al modo tradicional de gestión de ventanas sin composición evitando tener que reiniciar el servidor X. Además se puede aprovechar toda la infraestructura ya creada en X.Org para el manejo de multiples monitores o gafas 3D, mientras que con Xgl todo eso se pierde. En cuanto al soporte de hardware los tres grandes frabricantes de graficas (NVIDIA, ATI y Intel) soportan tanto AIGLX como Xgl, aunque en el momento de escribir este trabajo el funcionamiento de los drivers de ATI en el primer caso aun dejaba bastante que desear.