



Accelerated Indirect GLX

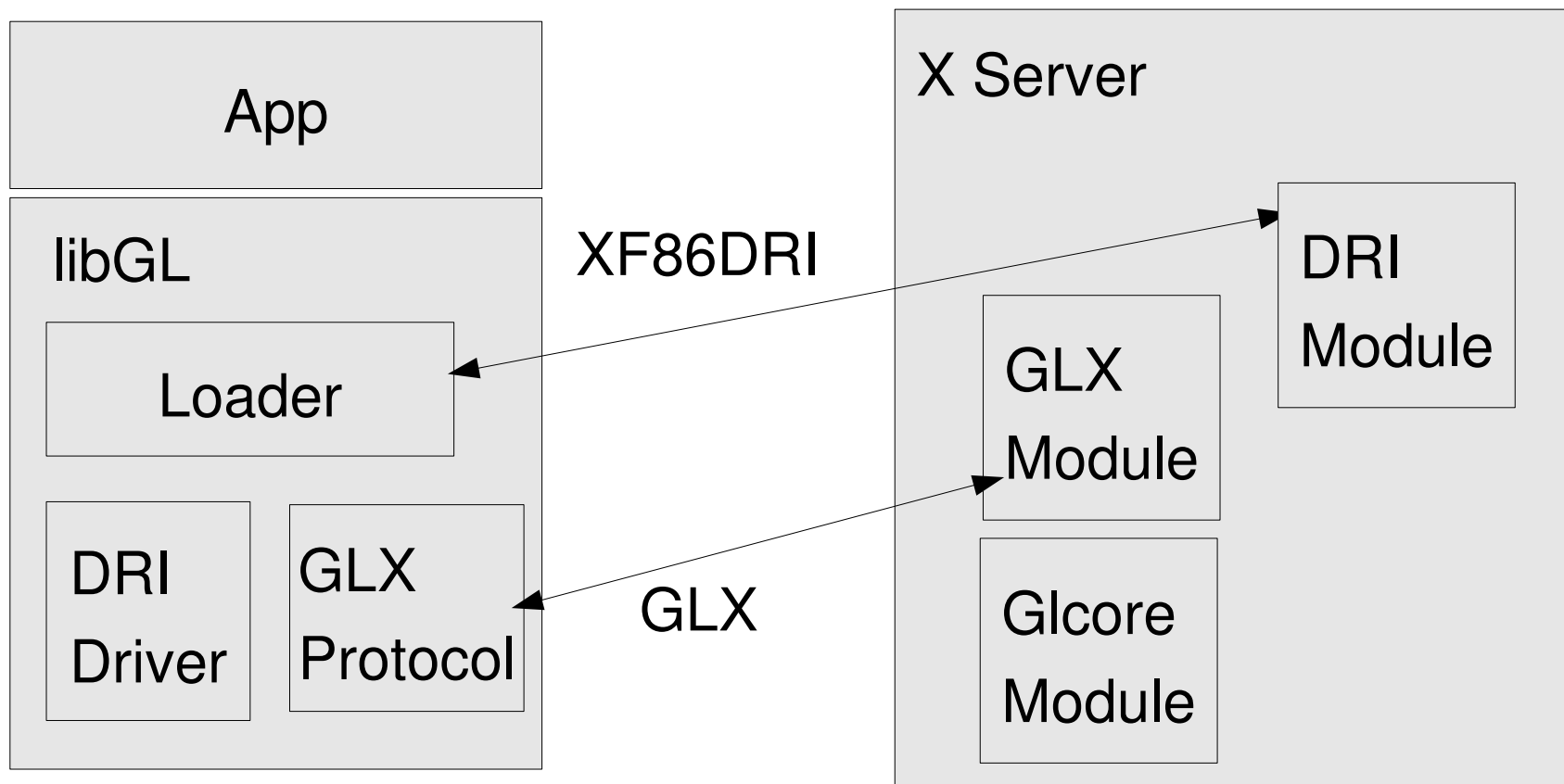
Kristian Høgsberg

XDevConf

9 Feb 2006

Current GLX Stack

- Client apps link to libGL
- Two modes of operation: direct and indirect



Direct Rendering

- Using the XF86DRI extension, the loader determines that the X server supports direct rendering.
- The loader asks the X server which DRI driver to load and where the framebuffer and sarea are located.
- The loader loads the specified DRI driver into the client using `dlopen()` and calls into the driver to create per screen data, drawables and GL contexts.
- All rendering is handled entirely in the client by calling into the driver which then programs the 3D hardware. No rendering requests are marshalled to the server.
- The X server only gets involved when moving windows or changing clip lists.
- Has become synonymous with accelerated rendering

Indirect rendering

- If the server doesn't support DRI, if part of the DRI setup fails or if the users specifically asks for it, libGL will fall back to indirect rendering.
- In indirect mode, all OpenGL requests are marshalled to the server and libGL is reduced to just a protocol layer.
- In the server, the GLX module maintains an OpenGL context on behalf of the client and forwards the demarshalled requests to the context.
- The current implementation pulls parts of Mesa into the Glcore module to provide a software backed GL context.
- Synonymous with software/slow rendering

Motivation

- If you can have accelerated direct rendering, then why would you want accelerated indirect?
- GLX_EXT_texture_from_drawable
 - Making X pixmaps available as OpenGL textures is a central operation in an OpenGL based compmgr
- With accelerated rendering is available inside the X server, it is possible to accelerate XRender Xv or other functionality using OpenGL (e.g. glitz) server side.
- Looking towards Xgl on egl, this will provide some of the same possibilities for clients and in the server, but implemented as a evolutionary step within Xorg.

Accelerated Indirect Rendering

- It's easy: just move loader from libGL into GLX module.
- The GLX module talks directly (in-process) to the X server DRI extension.
- We `dlopen()` the DRI driver and load it into the server.
- The loader provided callbacks (`__DRIinterfaceMethods`) are also just implemented by calling directly into the DRI code.
- For context and drawable creation, binding drawables to contexts and buffer swaps, the GLX module calls into the DRI driver.
- Thanks to Ian Romanicks dispatch table work, all other GLX requests are automatically forwarded into the currently active context dispatch table.

Visual Initialization Process

- Still a mess, but less so.
- At visual init time the GLX module adds various combinations of the GL properties for each of the existing visuals
- When the DRI driver is loaded, we ask the it what visuals it supports.
- The intersection of the DRI driver modes and the GLX computed modes is what we export.
- We don't get fbconfigs for 32-bit ARGB visuals provided by COMPOSITE, since they're added too late in the visual setup process.

The DRI Lock

- DRI uses a global lock to serialize access to the card.
- DRI drivers are designed with this lock in mind, so they respect it and take it when they need to touch the card.
- XAA, however, does not know about the DRI lock, so the DRI module in the X server installs block and wakeup handlers to always release and take the lock, respectively.
- Thus, when a direct rendering client runs, the X server sits in `select()`, and conversely, when the X server is active, all direct rendering clients are blocked on the DRI lock.

Deadlock

- With accelerated indirect rendering, we're using the DRI driver from within the X server.
- When we reach `glXDispatch()` in the GLX module, the wakeup handler has already taken the DRI lock, and we get a deadlock when the DRI driver tries to take it.
- So, we release the DRI lock while dispatching GLX commands and take it again when we're done, so the block handler can release it.
- With EXA we have a change to make things work better since we have `PrepareAccess` and `FinishAccess`, but it's probably too much work to make XAA play along.

Texture from drawable

- With the DRI driver in the X server address space, we can implement the `GLX_EXT_texture_from_pixmap` extension.
- The simple implementation is to assume pixmaps are in host memory (`XaaNoOffScreenPixmap`) and just call `glTexImage2D()` with pixmap data from the `Pixmap` structure. This is what we have now.
- A better implementation uses damage to track pixmap changes and uses `glTexSubImage2D()` to only uploads the parts that have changed when the texture is rebound.
- The best solution is of course to set things up so we can texture directly out of the pixmap data, saving memory and time – needs per driver work.

Software Rendering

- We still need a software solution for systems without supported 3D hardware.
- Current implementation makes protocol layer talk directly to DRI driver API.
- DRI software driver
 - de-couple mesa and xorg build processes
 - simplifies glx module significantly
 - Conflicts with Xgl work?
- To support non-double buffered rendering:
 - Need to fake sarea without DRI, but since software driver and X server run in same address space we can just malloc() the sarea.
 - DRI Software renderer needs to wrap all GL rendering commands and poll the sarea before rendering anything – just like the hardware DRI drivers do.

Software Rendering

- Alternative, and probably better approach: New abstraction layer in GLX module
- Would look roughly like DRI interface.
- Will be implemented by:
 - Small amount of glue code on top of DRI driver API
 - Using software mesa by shuffling the current GLX glue code around.
 - In Xglx by forwarding to underlying GL stack.
- Once this is done, the accel_indirect_glx branch should be mergeable.

Future work

- More debugging, testing with a wider range of chipsets.
- DRI driver interface suggestion: make `createNewDrawable` not add the drawable to the hash, make a `bindContext()` call that takes `__DRIdrawable` pointers instead of drawable ids.
- ...

Demo

- Glxgears
- Quake
- Metacity with composite manager
 - Wobbly windows are so 2005...

Questions?