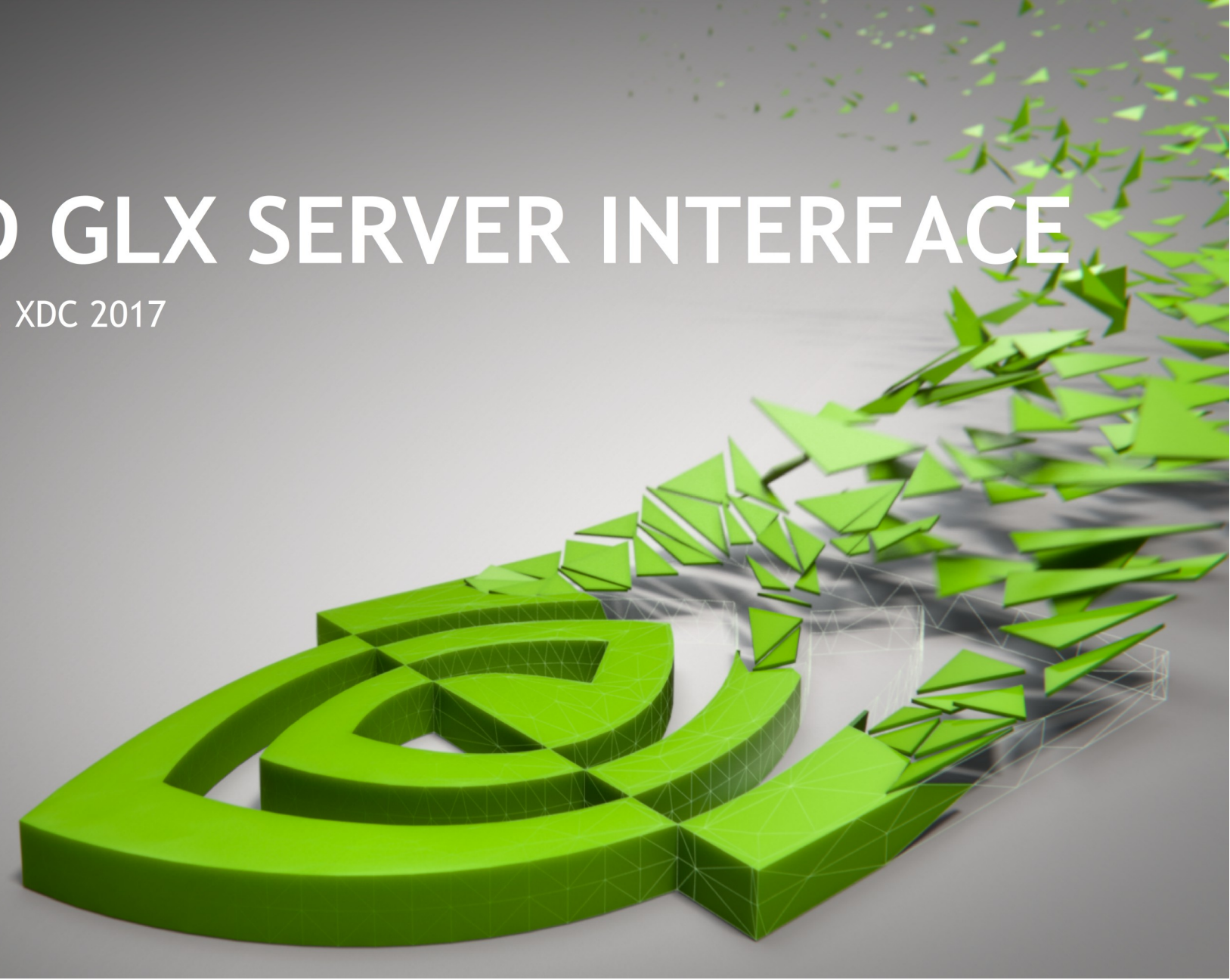
GLVND GLX SERVER INTERFACE

Kyle Brenneman, XDC 2017

NVIDIA



Agenda

Overview of client libraries and client-side dispatching.

Design for a new server-side interface.

Server-side vendor selection.

Server-side dispatching.

Open questions and future work.

GLVND CLIENT INTERFACE

Client-Side GLVND

GLVND defines a new ABI between OpenGL applications and implementations.

It allows multiple vendor implementations to coexist on the filesystem.

It allows multiple vendor implementations to coexist in the same process.

It allows EGL and GLX to coexist in the same process.

The client API libraries (OpenGL, OpenGL ES) work with GLX and EGL.

Client-Side GLVND, Continued

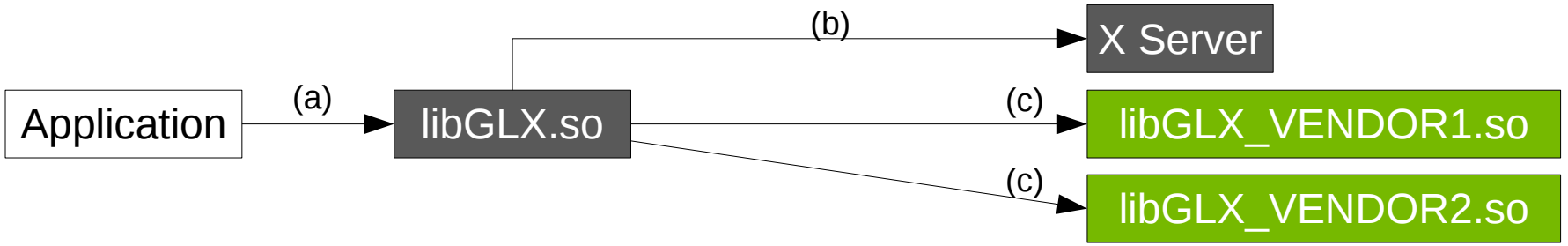
GLVND provides vendor-neutral versions of the application-facing libraries:

- libGLX.so
- libEGL.so
- libOpenGL.so
- libGLESv1_CM.so
- libGLESv2_CM.so
- libGL.so

Vendors provide implementation libraries:

- libGLX_${VENDOR}.so
- libEGL_${VENDOR}.so

Dispatching GLX Functions



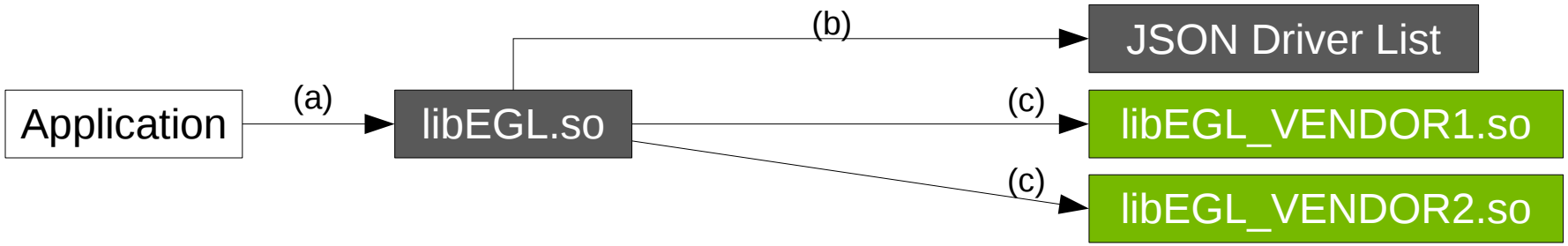
a) Application calls a GLX entry point in libGLX.so.

b) libGLX.so uses GLX_EXT_libglvnd to query the vendor name for each screen from the server.

c) libGLX.so loads the vendor library and forwards the GLX function call to it.

- Each GLX entrypoint has a dispatch function that figures out which screen number the call should go to.

Dispatching EGL Functions

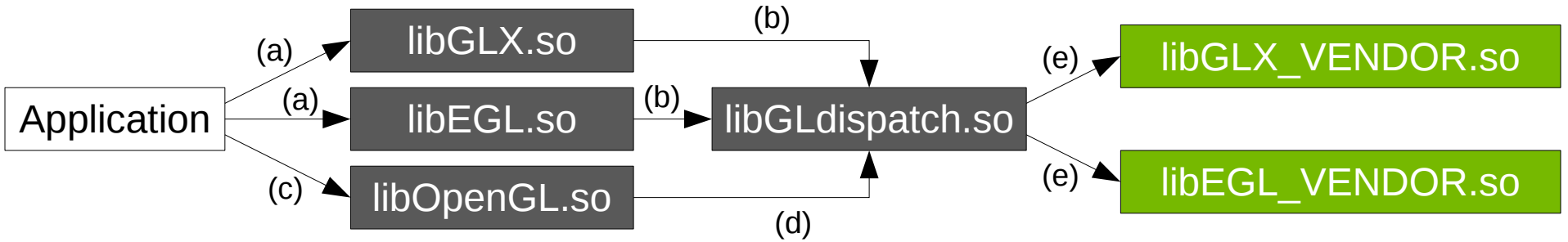


a) Application calls eglGetPlatformDisplay.

b) libEGL.so gets a list of vendors and loads each of them.
   - Each vendor registers itself by putting a JSON file into a well-known directory.

c) libEGL.so calls into each vendor's eglGetPlatformDisplay implementation until it finds one that succeeds.
   - libEGL.so will dispatch functions that take an EGLDisplay handle to whichever vendor created the handle.

Dispatching OpenGL Functions



a) Application calls glXMakeCurrent or eglMakeCurrent.

b) libGLX.so/libEGL.so creates a dispatch table for the vendor, and assigns it to the current thread in libGLdispatch.so.

c) Application calls an OpenGL entrypoint.

d) libOpenGL.so (or libGL.so, libGLESv2.so, etc) forwards the call to libGLdispatch.so.

e) libGLdispatch.so finds the dispatch table from (b), and jumps through it to the vendor library.

GLVND SERVER INTERFACE

Why do we need server-side dispatching?

Direct rendering only works if the client-side vendor library matches the server-side GLX module.

GLX in the server still exists as a normal extension module (libglx.so).

The server can only load one GLX module at a time, so you can't use different drivers on different screens.

Design Goals

Be able to handle any and all GLX requests correctly.

Allow (at least) a different vendor on each screen.

Keep the differences between a GLVND and non-GLVND driver to a minimum.

Overall Structure

The GLVND layer registers the GLX extension.

During initialization, the X drivers call into GLVND to assign a vendor to each screen.

When a GLX request comes in, GLVND figures out which vendor should handle the request, and forwards it.

Vendor Registration

The X driver registers an initialization callback in GLVND, which GLVND calls from InitExtensions.

The driver loads and initializes the appropriate GLX implementation from that callback.

The driver passes that implementation as a function table to GLVND, which returns an opaque vendor handle.

The driver then calls into GLVND to assign that handle to whichever screens it supports.

Since GLVND only needs a function table, the GLX implementation doesn't have to be a separate library.

Dispatching

Every request opcode has a dispatch stub function.

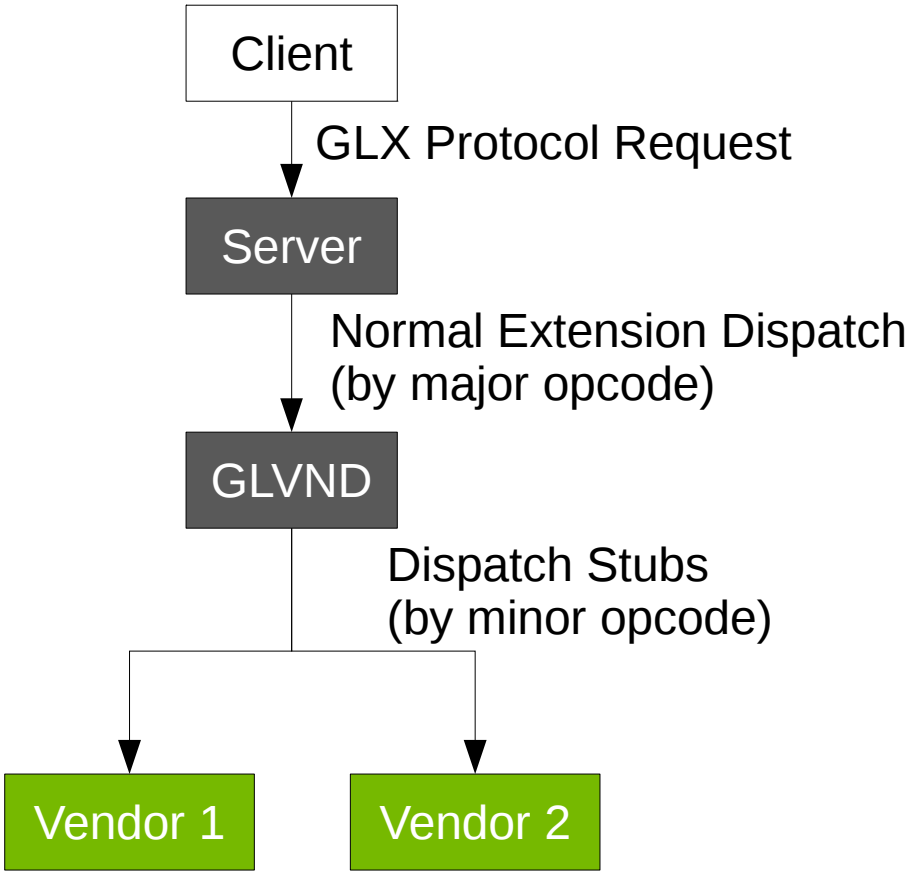
The dispatch stub looks at the request to find a screen number, context tag, or XID.

It uses that value to look up a vendor handle, then calls into GLVND to forward the request.

The vendor libraries provide dispatch stubs for any extension requests.

Most dispatch stubs can be generated.

Some requests need special handling.



Dispatching by screen

Each screen has a single vendor assigned to it, so just look up the vendor.

```
int dispatch_GLXQueryServerString(ClientPtr client)
{
    REQUEST(xGLXQueryServerStringReq);
    __GLXServerVendor *vendor = NULL;
    if (stuff->screen < screenInfo.numScreens) {
        ScreenPtr screen = screenInfo.screens[stuff->screen];
        vendor = GLVND->getVendorForScreen(client, screen);
    }
    if (vendor != NULL) {
        return GLVND->forwardRequest(vendor, client);
    } else {
        return BadValue;
    }
}
```

Dispatching by XID

GLVND keeps track of a vendor for each XID.

The dispatch stub is responsible for adding and removing client-specified XID's from that mapping.

The vendor has to add any server-generated XID's to the map during initialization.

One exception is GLXFBConfig ID's, because every request that has a GLXFBConfig also has a screen number.

GLVND can look up a screen number and vendor for regular X windows.

Lookup by XID – GLXQueryContext

```
int dispatch_GLXQueryContext(ClientPtr client)
{
    REQUEST(xGLXQueryContextReq);
    __GLXServerVendor *vendor = GLVND->getXIDMap(stuff->context);
    if (vendor != NULL) {
        return GLVND->forwardRequest(vendor, client);
    } else {
        return GLXBadContext;
    }
}
```

Adding an XID – GLXCreateContext

```
int dispatch_GLXCreateContext(ClientPtr client)
{
    REQUEST(xGLXCreateContextReq);
    __GLXServerVendor *vendor = NULL;
    LEGAL_NEW_RESOURCE(stuff->context, client);
    if (stuff->screen < screenInfo.numScreens) {
        ScreenPtr screen = screenInfo.screens[stuff->screen];
        vendor = GLVND->getVendorForScreen(client, screen);
    }
    if (vendor != NULL) {
        int ret;
        if (!GLVND->addXIDMap(stuff->context, vendor)) {
            return BadAlloc;
        }
        ret = GLVND->forwardRequest(vendor, client);
        if (ret != Success) {
            GLVND->removeXIDMap(stuff->context);
        }
        return ret;
    } else {
        return BadMatch;
    }
}
```

Removing an XID – GLXDestroyContext

```
int dispatch_GLXDestroyContext(ClientPtr client)
{
    REQUEST(xGLXDestroyContextReq);
    __GLXServerVendor *vendor = GLVND->getXIDMap(stuff->context);
    if (vendor != NULL) {
        int ret = GLVND->forwardRequest(vendor, client);
        if (ret == Success) {
            GLVND->removeXIDMap(stuff->context);
        }
        return ret;
    } else {
        return GLXBadContext;
    }
}
```

Context Tags

GLVND selects context tag values to ensure that they're unique across vendors.

GLVND keeps a (void *) pointer for each context tag, which is used to store arbitrary vendor-private data.

GLVND never dereferences that pointer, so vendors can also use it to hold whatever tag value they would have used in a non-GLVND version.

MakeCurrent Requests

Vendors provide a separate function to handle MakeCurrent requests.

If a request switches between vendors, then GLVND calls into both of them.

GLVND passes the old and new context tags, and the old private data to the vendor.

The vendor returns a new private data pointer.

After calling into the vendor, GLVND sends the reply.

Dispatching by Context Tag

```
int dispatch_GLXRender(ClientPtr client)
{
    REQUEST(xGLXRenderReq);
    __GLXServerVendor *vendor = NULL;
    GLVND->getContextTag(client, stuff->contextTag, &vendor, NULL);
    if (vendor != NULL) {
        return GLVND->forwardRequest(vendor, client);
    } else {
        return GLXBadContextTag;
    }
}
```

Other Special Case Requests

GLXQueryVersion – Doesn't take a screen number, but it's probably safe to assume that every driver supports GLX 1.4.

GLXSwapBuffers – It has to look up a vendor by context tag if the tag is non-zero, or else one vendor might try to dereference the private data from another vendor.

GLXClientInfo – The request has to get forwarded to every vendor, not just one.

Current Status

Implemented as a proof-of-concept right now.

Direct and indirect rendering work on two screens with different drivers, from a single client.

No changes are needed in the client library.

Open Questions

Where should the GLVND code live?

Should the GLVND interface stay as extension module?

What about Xvfb, Xwayland, etc?

Future Work

Xinerama

With a single driver, Xinerama should work like it does now.

With different drivers, requests would currently get dispatched to just one of them.

Cross-vendor GPU offloading

Per-client vendor selection.

How would the client and server coordinate to pick a vendor?

How do we deal with X visuals?

What happens if the same XID is used between multiple clients?

How does communication and presentation between drivers work?

Links and Acknowledgements

Thank you to everyone who's contributed patches, comments, and bug reports.

Special thanks to Emil Velikov and Adam Jackson for their help in hashing out the various interfaces.

GLVND is available at:

https://github.com/NVIDIA/libglvnd

Questions?