GLX, the protocol for networked OpenGL

Mark J. Kilgard
mjk@sgi.com
Silicon Graphics, Inc.

Connetathon '96
February 28, 1996

Jist of this talk...

Overview of
OpenGL's GLX protocol
for the
network engineer,
not the
graphics engineer:

What does GLX do?
When is it used?
How does it operate?
What's novel about it?

What does GLX do?

- o GLX is an X11 extension protocol.
- o GLX integrates the OpenGL 3D graphics standard with the X Window System.
- o GLX provides two things:

  1) Network protocol for network extensibility.

  2) Integration and programming interface for using OpenGL with X.

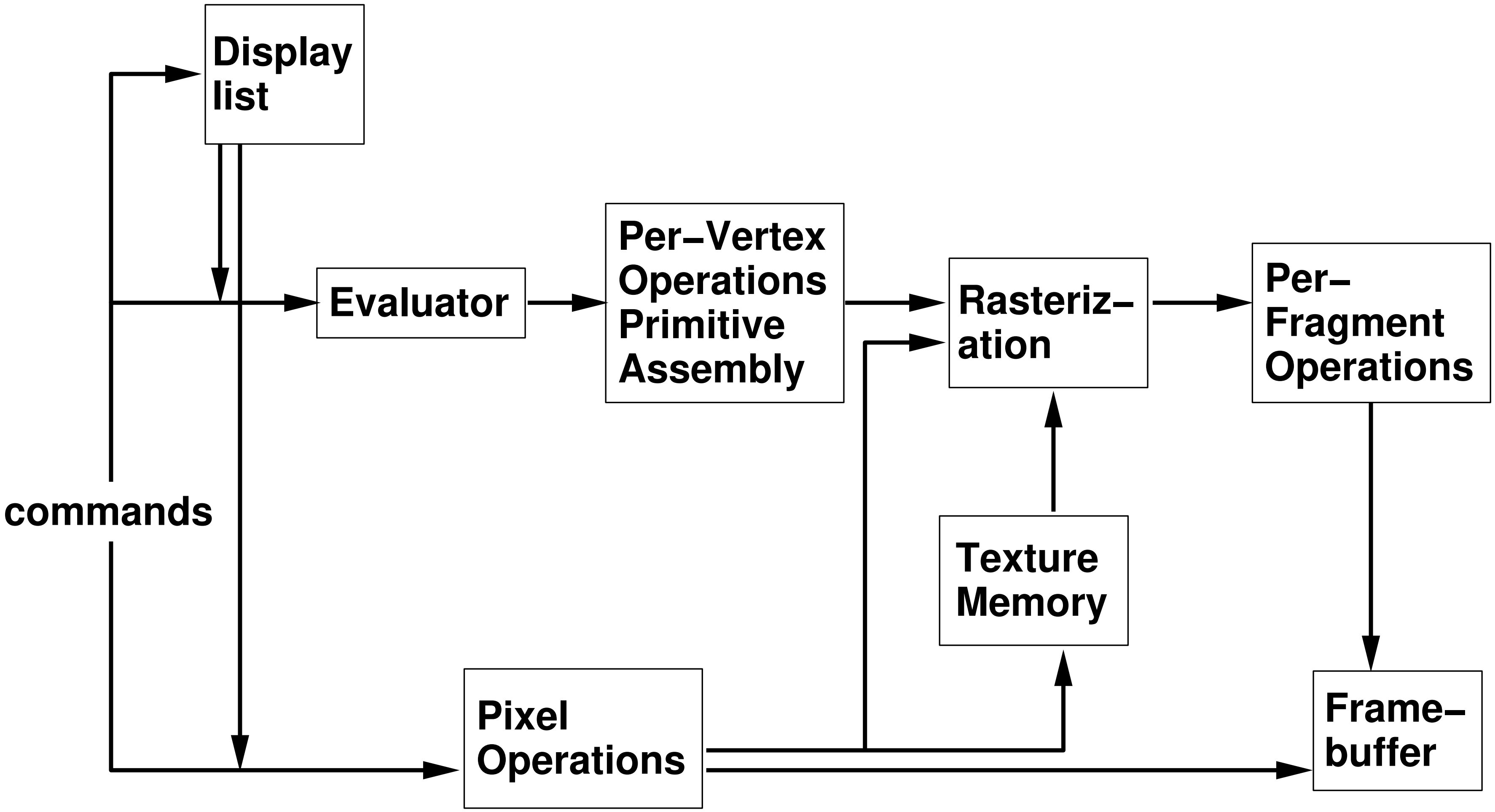
OpenGL (in two slides)

- Industry standard programming interface for interactive 2D and 3D graphics.
- Developed by Silicon Graphics.
- Overseen by Architectural Review Board.
- Low–level, procedural interface.
- High–performance, well–suited to hardware implementation.

OpenGL (almost done!)

- Mandated functionality; no subsetting of rendering capabilities.
- Advanced functionality: lighting, texture mapping, antialiasing, blah, blah.
- Conformance suite & well–defined specification provides compatibility.
- Window system independent rendering interface. X, NT, OS/2, Mac, etc.

Obligatory OpenGL State Machine



Integrating OpenGL with win–systems

- OpenGL's win–system independence requires integration to win–systems.
- Each window system defines its own interface model/API to use OpenGL.
- Functionality provided: creating rendering contexts, binding to windows, buffer swaps, etc.
- GLX is OpenGL's interface for X11.

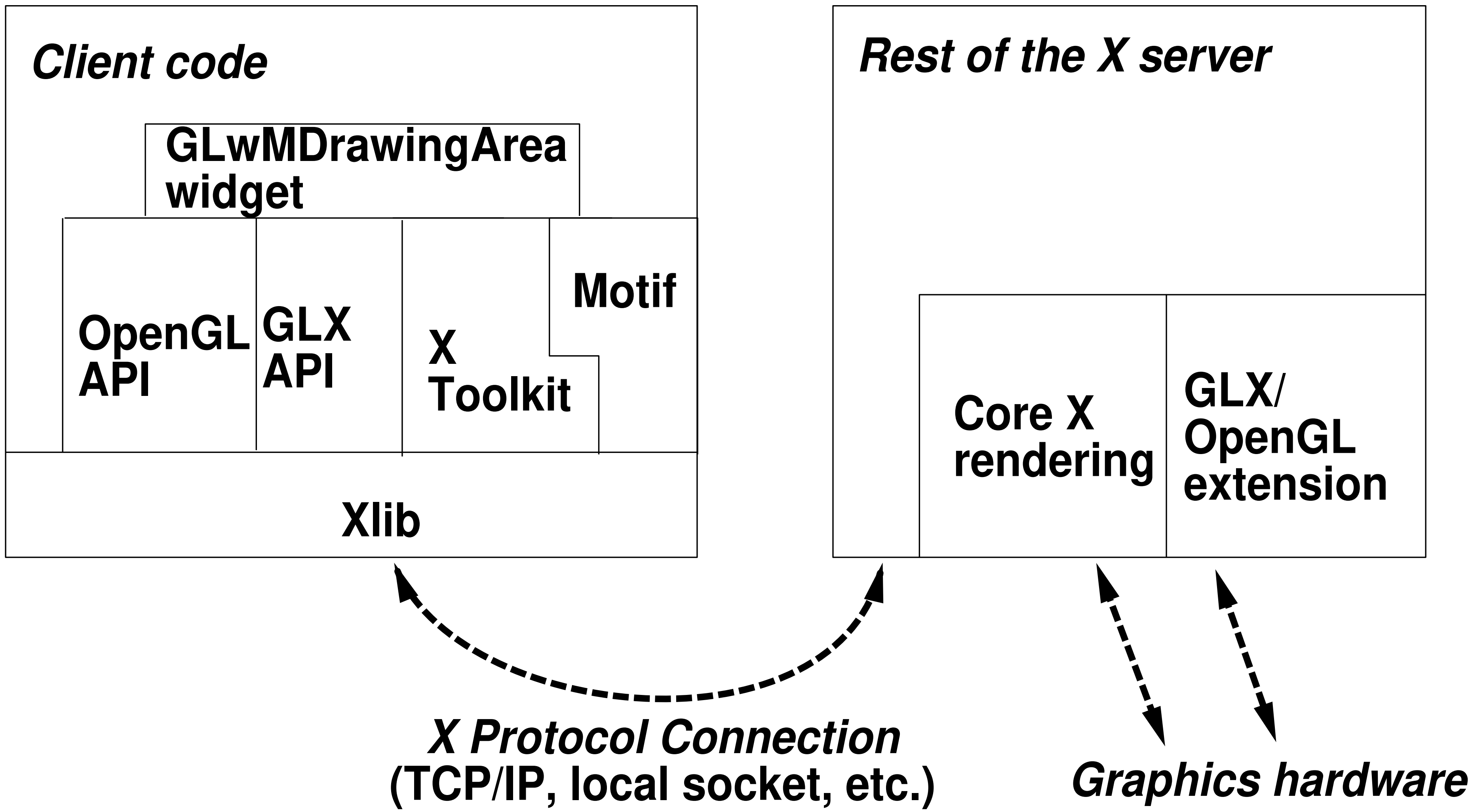
How does GLX do its job?

- GLX is implemented as X extension.
- Therefore, GLX protocol embedded in the X protocol stream.
- GLX also has API.
  Example: *glXCreateContext*
- Both GLX and OpenGL calls translated into GLX protocol requests.
- GLX protocol is standard & interoperable.

Example GLX calls

- *glXChooseVisual*
- *glXCreateContext*
- *glXCreateGLXPixmap*
- *glXSwapBuffers*
- *glXCopyContext*
- *glXDestroyContext*

GLX makes X server 3D render proxy



Distinction: GLX and OpenGL

- Logically, two distinct APIs (*gl*– vs. *glX*–).
- OpenGL = 3D rendering.
- GLX = managing OpenGL for X.
- Two APIs packed into the single GLX protocol.

A little about OpenGL rendering

- OpenGL routines take no context or destination: *Example:*

```
glRectf(1.0,2.0,1.0,3.0);
```

- GLX "binds" an OpenGL context and drawable to thread with *glXMakeCurrent*.

- OpenGL calls done with respect to current bound context and drawable.

- Lower overhead; win–sys independent API.

Generating GLX Requests

- Most GLX calls have corresponding GLX protocol request.
- OpenGL rendering calls (no replies) are packed into *glXRender* requests.
- OpenGL rendering calls (no replies) with >256KB data use *glXRenderLarge*.
- Non–rendering OpenGL calls each have corresponding GLX requests.

Displays Lists Minimize Bandwidth

- To reduce protocol overhead, OpenGL display lists are useful, ie. *glCallList*
- Display lists are uneditable sequences of OpenGL commands.
- Once created, they are in the X server.
- Display lists can call other display lists.
- Effective use minimizes protocol use.

GLX Context Tags

- Multiple client threads using shared X server connection?
- “Make current” managed per–thread, not X connection.
- Each OpenGL request sends a GLX Context Tag identifying thread.
- Context tag to use is returned by *glXMakeCurrent* request.

Value of GLX Context Tags

- Context tags distinguish distinct threads' current OpenGL binding.
- GLX API hides this "protocol detail" from the program calling GLX & OpenGL routines.
- In practice, value of this is dubious.

OpenGL & X Streams Independent

- OpenGL & X command streams independent.
- *glXWaitX* and *glXWaitGL* provide explicit ordering of the streams.
- Logically, these exist in both streams.
- More about why this might be later. . .

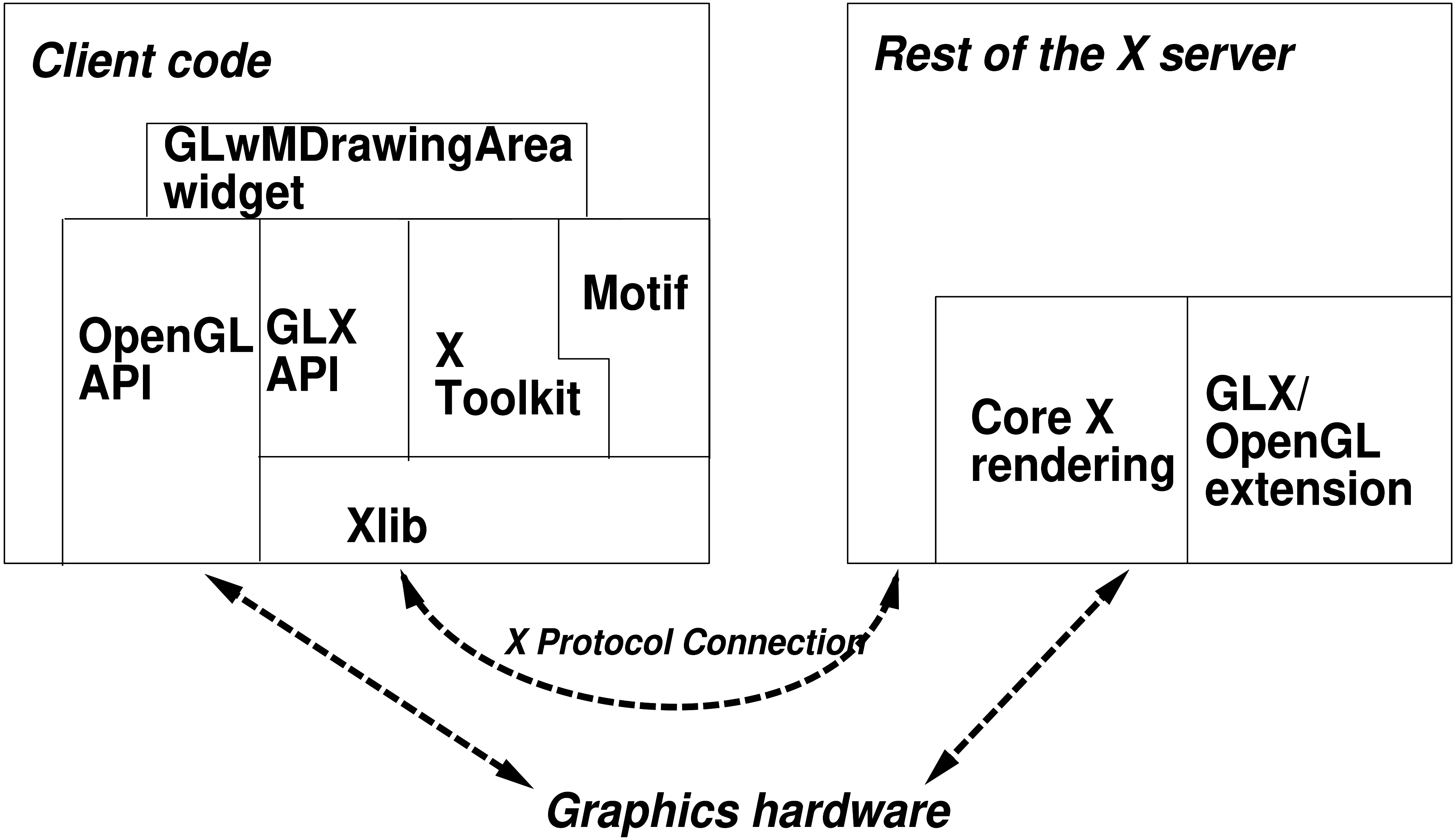
GLX & OpenGL Extensions Possible

- Yes, extensions of an X extension!
- *glXVendorPrivate* & *glXVendorPrivateWithReply*
- Op–code space assigned by OpenGL ARB for shared extensions; vendor ranges for vendor–specific extensions.
- Distinct GLX and OpenGL extensions possible.

Overhead of GLX Protocol Hurts OpenGL Performance

- 3D graphics & imaging = hi–bandwidth.
- Network extensibility is nice, but . . .
- For fast graphics hardware, GLX protocol “relay” starves graphics hardware.
- GLX permits bypassing of protocol for (local) direct access.

Architecture of Direct Rendering OpenGL (*locals only, dude!*)



Direct Rendering Bypasses X Server

- Optional for GLX implementations.
- *glXCreateContext* has option to try direct rendering.
- Need some support for virtualizing access to graphics.
  See “System Support for OpenGL Direct Rendering,” GI ’95.
- OpenGL calls = direct hardware access; no protocol (independent streams!).
- Connection to X server still exists.

Example of Local RPC Optimization

- Access to OpenGL's state machine can be considered *client/server*.
- Logically, OpenGL rendering commands are Remote Procedure Calls.
- GLX's direct rendering is an example of local RPC optimization.
- Direct rendering lets OpenGL render at hardware performance limits!

GLX Summary

- GLX supports OpenGL for X11.
- Makes OpenGL inter–operable & network extensible.
- GLX itself can be extended for GLX and OpenGL extensions.
- Direct rendering is very important optimization of GLX (by *eliminating* the rendering protocol!).