OpenGL's Immediate Mode Interface on Open-Source Platforms

Ian D. Romanick

idr@us.ibm.com

Agenda

- Where is OpenGL now?
- Where is OpenGL going?
- How does the current infrastructure fit in?
- How can the infrastructure adapt?
- Conclusion

Where is OpenGL now?

- OpenGL is an ornery 14-year old.
- The API has picked up a lot of “cruft” over the years
  - New features are added that improve performance / usability
  - Existing applications need the old interfaces kept around
    - Backwards compatibility is one of OpenGL's strengths!
  - There are now 4 different ways to submit vertex data
    - Burden for application developers: How to choose?
    - Burden for driver developers: Which to optimize?
  - DirectX “flushes” API periodically and doesn't have this problem
- OpenGL Architecture Review Board is aware of this problem.

Where is OpenGL going?

- Some interfaces will be removed from a future version
  - Follow the footsteps of OpenGL ES
- Compatibility for some interfaces will be provided by a “shim”
  - Thin layer between the application and the driver that emulates the deprecated functionality
  - Some versions of the shim may also provide debugging support

Where is OpenGL going? (cont.)

- There are too many different ways to submit vertex data
  - Immediate mode
  - Display lists
  - Client-side vertex arrays
  - Server-side buffer objects
- Follow the OpenGL ES lead and give immediate mode the axe!
- The shim layer would emulate immediate mode using either vertex arrays or buffer objects

Current infrastructure

- libGL provides thinnest possible layer between driver & app
- Function calls directed into the driver via dispatch functions and a dispatch table
  - Similar to C++ virtual functions
  - Adds measurable overhead to some applications

```
void glVertex3fv(const Glfloat *v)
{
    (*_glapi_Dispatch_tls->Vertex3fv)(v);
}
```

Adapting the infrastructure

- Existing library is obvious location for “shim”
- Implement immediate mode directly in libGL
    - Marshal data into vertex arrays
    - Submit data when glEnd is called
    - Eliminates dispatch overhead!
    - Similar to indirect rendering implementation
- Moves a *lot* of code from each driver into libGL
    - Violates “thinnest possible” principle, may be contentious

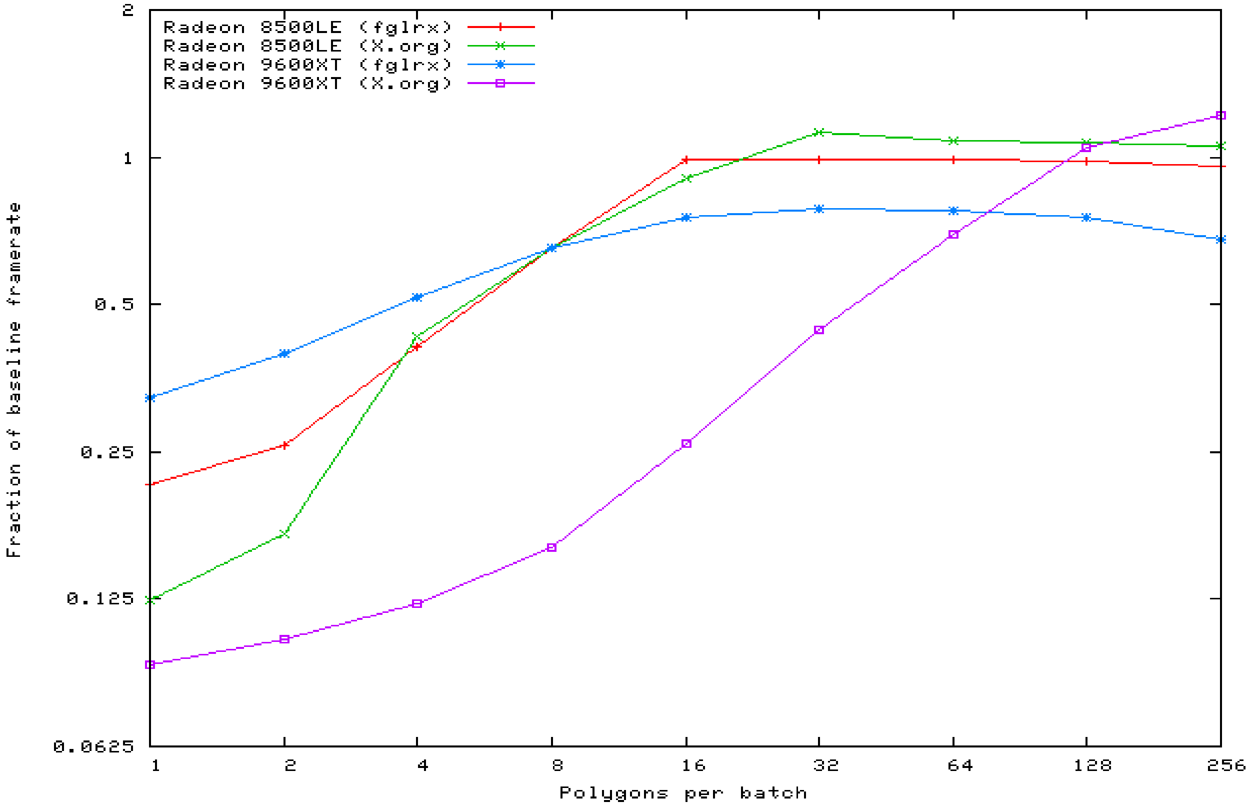
Pitfalls to implementation

- Non-array data
  - glMaterialf
- Non-uniform API usage
  - Mixing data types within a primitive
  - Mixing data counts within a primitive
  - Changing per-vertex data within a primitive
  - Mixing immediate mode and arrays
- Display lists
- Vendor extensions

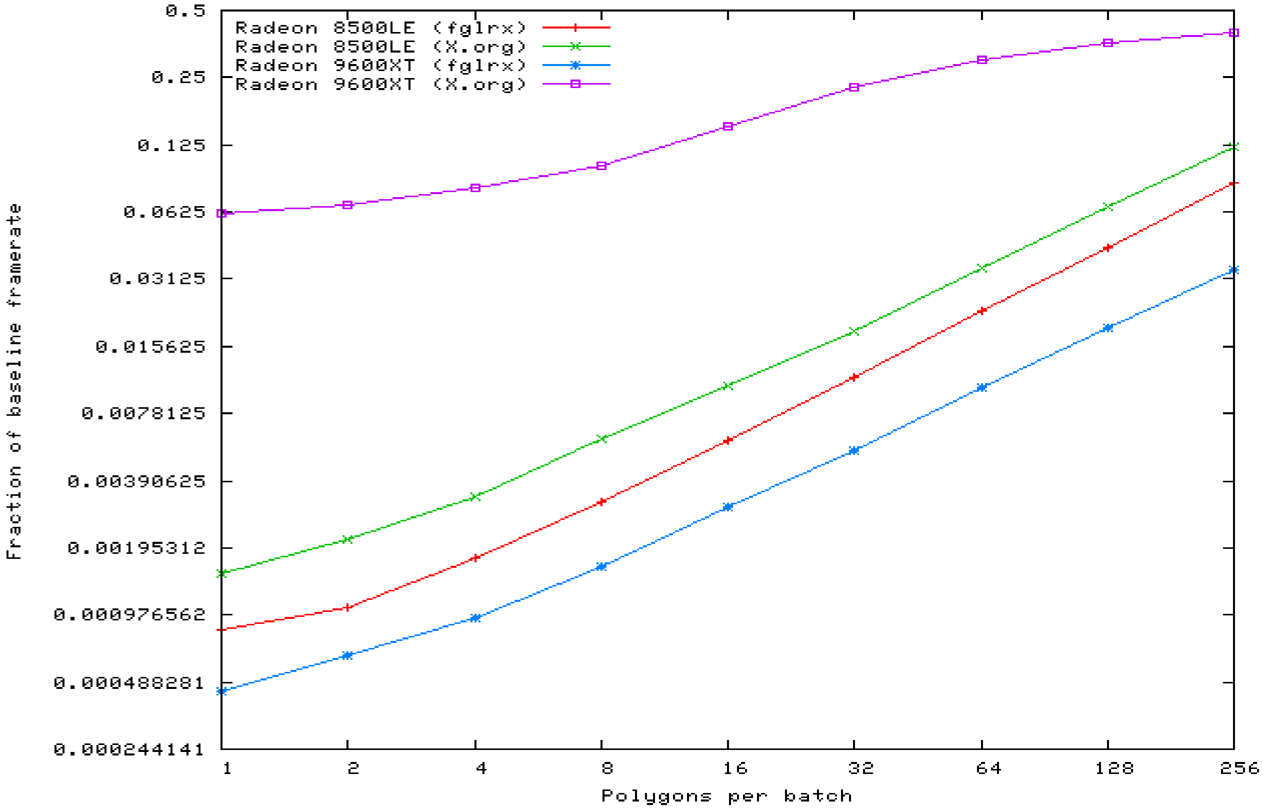
Projected performance

- Tested simple program with partial emulation layer
- Emulation layer can use several modes
  - Client-side vertex arrays
  - “Fire and forget” server-side buffer objects
    - This should be the optimal mode
  - Reused server-side buffer objects
- Tested two cards and two drivers
  - Radeon 8500LE with open source drivers and fglrx
  - Radeon 9600XT with open source drivers and fglrx

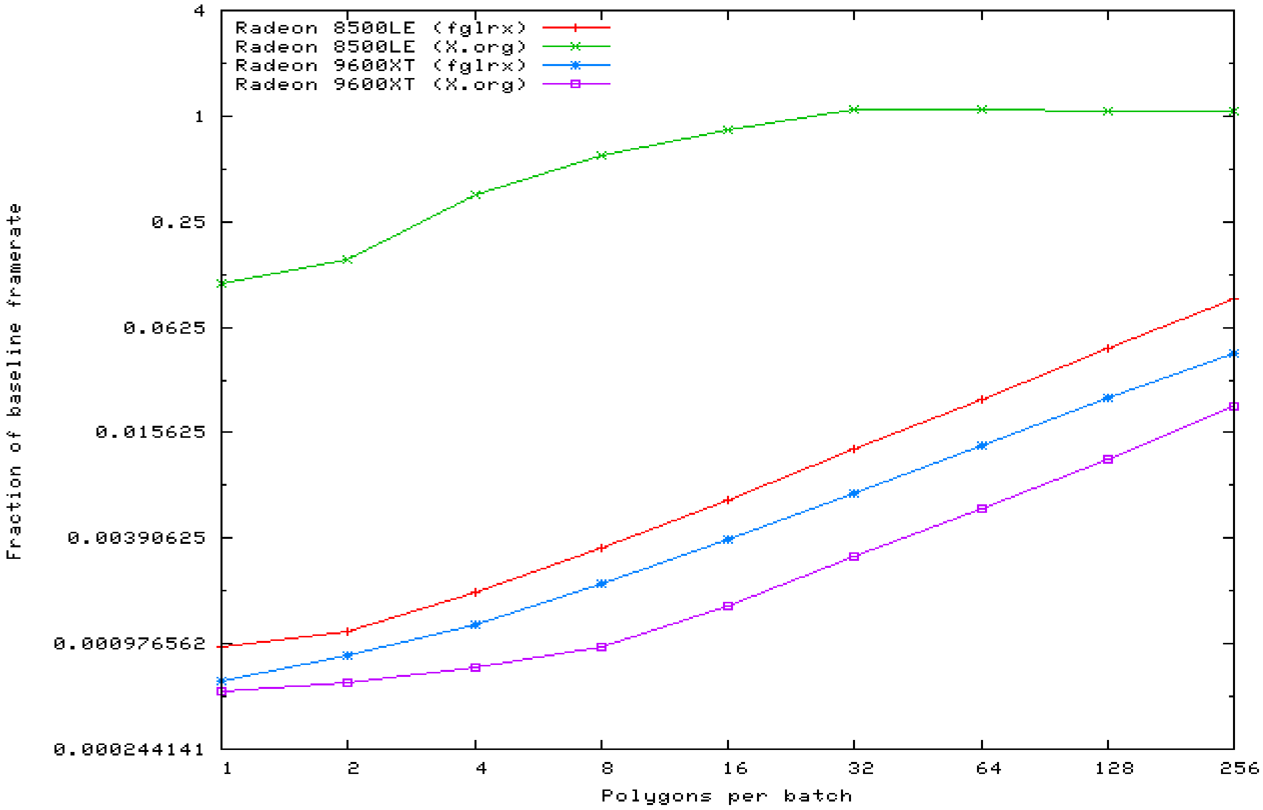
Performance with vertex arrays



Performance with buffer objects (fire & forget)



Performance with buffer objects (reused)

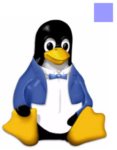


Surprising results!

- Buffer object performance inconsistent across implementations
  - “Fire & forget” performed poorly
    - By the design of buffer objects, this should be the optimal mode!
  - Neither usage pattern well suited to fglrx implementation!
- Gives insight into implementation specifics
  - fglrx implements buffer mapping by copying
  - Helps guide future interface designs

Future extensions to improve implementation

- True “zero” stride
  - Reuse data element for each vertex in a primitive
  - Extend to full instancing?
- Array state containers
  - Already proposed for future OpenGL version
  - GL_APPLE_vertex_array_object implemented in Mesa
- Flush callback
  - Driver notifies shim of state changes to improve batching
- Buffer object subrange unmap
  - Inform driver that a subrange of a mapped VBO was modified



Next steps

- Determine acceptability of “fattening” libGL
  - Doing this *right* will likely require significant changes to Mesa
- Rearchitect Mesa to move common front-end code
  - X.org libGL and pure software Mesa should share code
  - Should reduce maintainence burden on both paths

Questions?

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM.
- IBM is a registered trademark of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Microsoft, Windows, and DirectX are trademarks of Microsoft Corporation in the United States, other countries, or both.
- OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.
- Other company, product, and service names may be trademarks or service marks of others.