Writing Portable OpenGL ES 2.0

Tom Olson
Director, Graphics Research, ARM MPD
Chair, OpenGL ES Working Group
Writing Portable OpenGL ES 2.0

- Why is this a problem?

- What are the key portability issues in OpenGL ES 2.0?

- What are the best practices for managing them?

- Caveats
  - Just one spec geek’s view
  - Just a beginner’s guide
Portable Graphics: Problem Solved?

• OpenGL ES 2.0 is everywhere!
  - iOS
  - Android
  - WebOS
  - Symbian
  - etc

• So, everything is peachy now, huh?
  - Sadly, no.
  - Worse, it was done deliberately!
Why isn’t this a solved problem?

• Graphics APIs must trade off *portability* against *freedom to innovate*.

Ecosystem Viability

OpenGL ES 2.0 mandates *minimum* functionality, but allows implementations to exceed the minimum or offer new functionality by extension.

A single graphics API dictates functionality and performance completely. Monopolies emerge and innovation ceases.

Every vendor creates a proprietary API, optimized to exploit their platform’s unique features. Sadly, nobody writes code for any of them.

More portability vs Flexibility

More features
The One-Slide Guide to Portability

Implementation B

Implementations provide partially overlapping superset of spec-mandated core functionality

Implementation C

ES 2.0 spec mandated minimum functionality

Implementation A

Basic Recommendation

- Design your engine for the portable core
- Then, pull in implementation-specific functionality as time permits or necessity dictates
Portability Issues in OpenGL ES 2.0

• Performance characteristics

• Implementation options
  - Implementation-defined limits
  - Shader engine arithmetic precision
  - Shading Language restrictions
  - Error Behavior

• Extensions
  - Texture compression
  - “Silent” extensions
Performance Characteristics

• The bad news:
  - Huge variation even within a single GPU architecture / family
  - Each architecture has different strengths and weaknesses
  - Optimizing for one may hurt you on another

• Best Practices:
  - Focus on generic optimizations first
  - Get to know vendor DevRel teams and documents, and test on all your target GPUs early in the development cycle
  - Design your engine to adapt to device performance and capabilities
Implementation-defined limits

• **OpenGL ES 2.0 defines axes along which implementations may differ**
  - E.g., “how many vertex attributes can I use”?
  - Exposed as queriable read-only state, e.g. MAX_VERTEX_ATTRIBS
  - Specification mandates minimum value all implementations must support
    - See specification Table(s) 6.17-6.20

• **Best Practices**
  - Stay within the Chapter 6 minimum-maximum values if you can
  - Or, query and adapt
  - Or, know and respect the maxima for the platforms you target
Shader Engine Arithmetic Precision

• GLSL ES allows you to apply precision qualifiers to variables
  - Lowp (10-bit), mediump (16-bit), highp (24-bit)

• Qualifiers specify the minimum precision that must be provided
  - Implementations can provide more than you ask for
  - You can query via `GetShaderPrecisionFormat()`

• Support for highp is *optional* in the fragment shader

• Best Practices
  - Watch out for: large texture wrap factors, sensitive reflection vectors
  - Declare `#precision float mediump` in your fragment shader
    - If you use highp, provide a fallback for implementations that don’t support it
  - Test on implementations that (really) don’t provide highp
Shading Language Restrictions

- GLSL ES 1.0 allows implementations to provide less-than-full support for the core shading language
  - No virtualization – compilation can fail due to “out of resources”
  - Loop bounds may have to be known at compile time
  - Implementations may have restrictions on indexing
    - May require indices to be compile-time constants
  - See GLSL ES 1.0 specification, Appendix A

- Many vendors relax some of these restrictions, but there is no query

- Best Practices
  - Stay within Appendix A guidelines, or be prepared to fall back
  - Import vendor off-line shader compilers into your shader development pipe
Bad Code is Bad for Portability

• Invalid pointers
  - Attribute buffers

• Shading Language Errors
  - Divide by zero
  - NaN generation / propagation

• API restrictions
  - TexImage2D type / format / internalformat restrictions
Extensions

• Implementations can advertise novel features via *extension specs*
  - Defined as deltas to the core spec – can add features but not remove them
  - E.g., OES_texture_npot
  - Query using GetString[v]()

• Categories
  - OES: Written and approved by the ES Working Group, ratified, conformance tested
  - EXT: Supported by multiple vendors
  - Vendor (e.g. ARM): Supported by a single vendor, possibly proprietary

• Best Practices
  - AVOID if you can
  - If you use, prefer OES to EXT to Vendor
  - Always query and adapt
Texture Compression

• **Texture compression is great! Use it!**
  - Formats: ETC1, PVRTC, DXTn, ...

• **The bad news:**
  - There are no universally supported formats
  - To target both iOS and Android, you will need at least two art paths

• **Best Practices**
  - On iOS: Use PVRTC (vendor extension)
  - Everywhere else:
    - Use ETC1 (OES extension) if you can
    - Or, query for other vendor extensions

Photo: http://www.dicts.info
Fun with ETC1 textures

- **ETC1 is RGB-only. What if I need RGBA?**
  - Use two channels of ETC1
    - Second texture fetch is often free
    - If not using wrapping, pack RGB and A images into an atlas
  - Or, use ETC1 for RGB and pack A into another texture
    - E.g. use LA88 to store a height map plus alpha

- **What about normals?**
  - Use two channels of ETC1

- **In general, ETC1 has excellent resolution in luminance**
  - Works well when R/G/B are highly correlated
“Silent” Extensions

• OpenGL ES 2.0 restricts certain convenience features of OpenGL

  • Example: `glTexImage2D(T, L, internalformat, W, H, B, format, type, d);`
    - In ES, `format` must match `internalformat`

• Rules are not well tested by the ES conformance test
  - Some implementations allow features that they shouldn’t

• Best Practices
  - Know the rules
  - Yell at ES 2.0 implementors who break them
  - (I’ll help)
Summary

- **Writing portable OpenGL ES 2.0 is challenging!**
  - It’s the price we pay for rapid evolution of the technology

- **To manage the beast:**
  - Design for the portable core of the API
  - Know the specification
  - Debug your code
  - Don’t use implementation-specific features if you can avoid it
  - Know (and test on) all of the major implementations

- **Good luck!**
Acknowledgements

• **Thanks to:**
  - Maurice Ribble (Qualcomm)
  - Daniel Koch (Transgaming)
  - Acorn Pooley (NVidia)
  - Anders Lassen (ARM)
  - Jan-Harald Fredricksen (ARM)
  - Jakub Lamik (ARM)
  - Ed Plowman (ARM)
  - Rob Simpson (Qualcomm)

• **Feedback gratefully accepted at**
  - http://www.khronos.org/bugzilla/