What is OpenGL ES?

• OpenGL for embedded & mobile devices
  - Gets rid of redundant / legacy features
  - Extensions to make it mobile-friendly
  - All the functionality of the desktop API

• Versions
  - ES 1.0 / 1.1: fixed function
  - ES 2.0: shader-based via GLSL ES

• A huge success!
  - The dominant native 3D API for mobile devices
  - Widely adopted for STB, DTV, automotive,…
  - Hundreds and hundreds of millions shipped
What’s New [at SIGGRAPH 2009]?

- **OpenGL ES 2.0 platforms are shipping!**
  - Available in several popular handsets
  - Coming soon to set-top boxes, navigation devices, etc.
  - SDKs, books available
What’s New [at SIGGRAPH 2010]? 

OpenGL ES 2.0 is everywhere!

• **On the desktop**
  - ARB_ES2_compatibility
  - ES 2.0 context creation in desktop drivers

• **In the browser**
  - WebGL

• ...and, of course, on mobile devices

• Awesome content is appearing
What’s New in 2011?

The year of high-end content

• Game Engines modified for ES 2.0
  - UE3
  - Unigine

• Demos / Promos
  - Rage
    http://www.idsoftware.com/rage-mobile/

See “Bringing UE3 to Apple’s iPhone Platform”, Josh Adams (Epic Games), GDC 2010
Working Group Activities

• **Next Generation OpenGL ES**
  - Working group’s main focus since mid-2009
  - Will be released when the market is ready
  - Design driven by high-end content needs

• **ARB / ES Convergence TSG**
  - Meeting weekly to align roadmaps and minimize incompatibilities
  - Many examples of spec negotiation and adjustments

• **Grand Unified Conformance Test**
  - Joint project of OpenGL ES WG and the ARB
  - Common test framework for OpenGL 4.x and next-gen OpenGL ES
  - Largest ever investment by Khronos
  - Will lead to significant improvements in portability
Portability in OpenGL ES 2.0

Tom Olson
Director, Graphics Research, ARM MPD
Chair, OpenGL ES Working Group
SIGGRAPH Asia 2011
The World of Portability

- Where do we want to go?
Looking further away...
Cross-API Portability for OpenGL ES 2.0

• **OpenGL ES 1.1**
  - Code for OpenGL ES 2.0 and port back
  - Going the other way is hard work!

• **OpenGL**
  - Program for the common subset

• **Direct3D**
  - Consider getting help, e.g. Transgaming
Back on the island...
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*.

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.

---

Ecosystem Viability

More portability

Portability vs Flexibility

More performance

More features
The One-Slide Guide to Portability

Implementations provide partially overlapping supersetsof spec-mandated core functionality

Implementation A

Implementation B

Implementation C

ES 2.0 spec mandated minimum functionality

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 – Driver Issues

• Deferred execution
  - Rendering
  - Texture Upload
  - Compilation

• Compilation Speed

• State Changes
  - State dependency triggered compiles

• Draw Call Overhead
Performance – Other Issues

• GPU Hardware
  - Lots of 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 - Arithmetic Precision

• **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 - Language Restrictions

• Stay within Appendix A guidelines if you can

• Or, be prepared to fall back

• Test using vendor off-line shader compilers
  - Integrate into your build system or shader authoring pipeline
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
  - CopyTexImage2D conversions
Extensions

• **Implementations can add new 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, PVR, ...): 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
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 features of OpenGL
  - “If it is going to be in software anyway, let the application do it”

• 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
  - Vendors are supposed to advertise extra features by extension
  - Vendors are NOT supposed to add features that the WG removed
Best Practices - “Silent” Extensions

• Test on multiple implementations

• 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/