Efficient OpenGL ES Programming

Nakhoon Baek
Kyungpook National University

9 Nov 2011 @ KGC 2011

Efficiency ?

- **from programmer's point of view:**
  - one source
  - multi-use

© Copyright Khronos Group, 2011 - Page 2

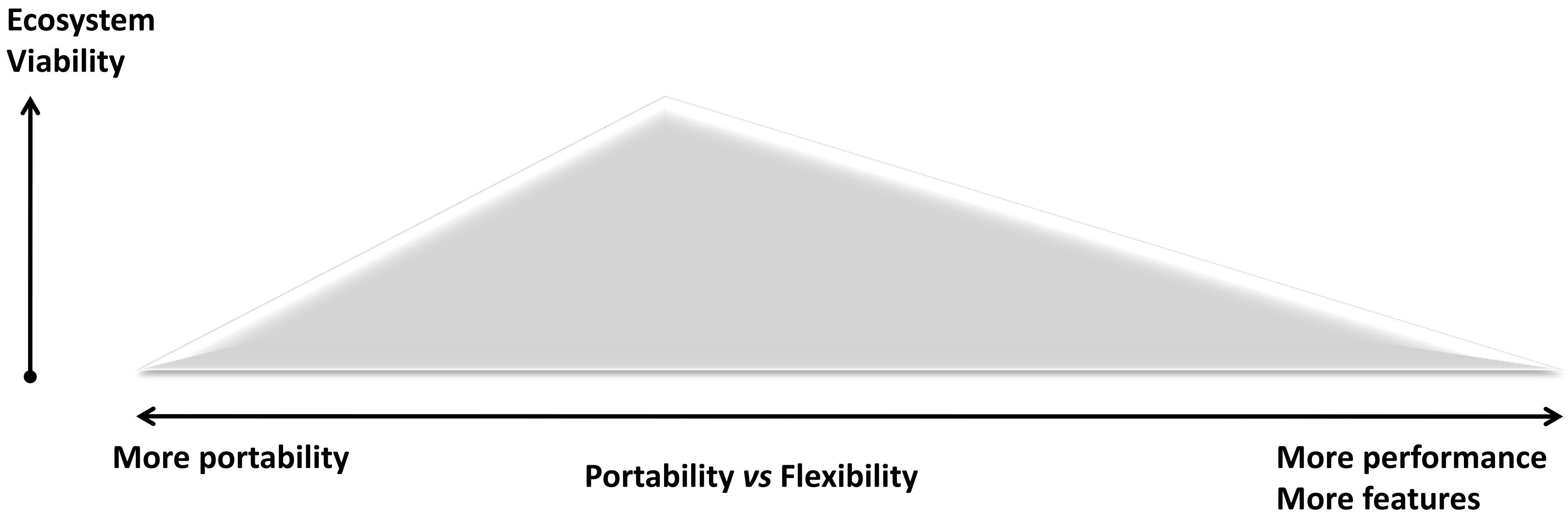
For OpenGL ES

- **one OpenGL ES program**
  - for Apple iPhone
  - for Google Android
  - for Samsung Bada
  - for WebOS
  - for Symbian
  - for Windows PCs

  - for any platform !

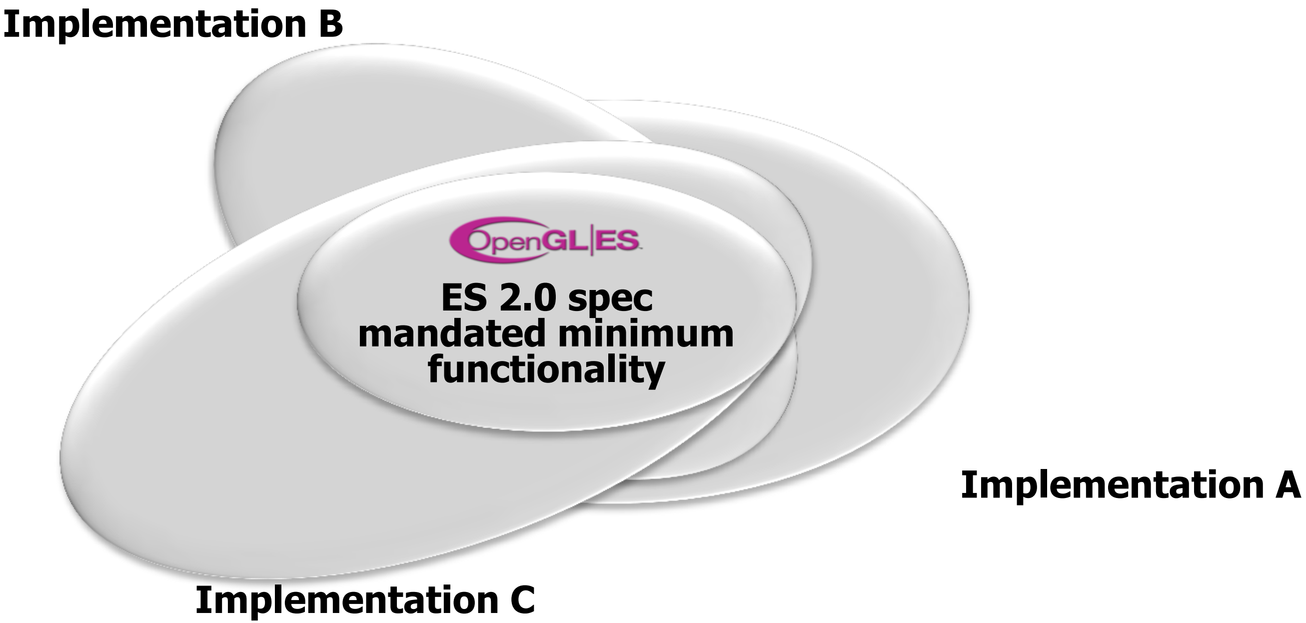
What is the problem?

- portability *vs.* freedom to innovate



OpenGL ES's solution

- **core features vs. extensions**
  - Implementations provide partially overlapping supersets of spec-mandated core functionality



© Copyright Khronos Group, 2011 - Page 5

Portability Issues in OpenGL ES 2.0

- **Performance characteristics**

- **Implementation options**
  - Implementation-defined limits
  - Shader engine arithmetic precision
  - Shading Language restrictions

- **Extensions**
  - texture compression

Performance Characteristics

- **current status:**
  - **Huge variation** even within a single GPU architecture / family
  - Technology is **evolving rapidly**
  - Optimizing for one may hurt you on another

- **strategy:**
  - Focus on **generic optimizations first**
  - Design your engine to adapt to device performance and capabilities

Implementation-defined limits

- **OpenGL ES 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**
  - Or, query and adapt

© Copyright Khronos Group, 2011

Implementation-defined limits

- **Chapter 6 from the OpenGL ES Specification:**

Get value	Type	Get Cmnd	Initial Value	Description	Sec.
SHADER_TYPE	Z_2	**GetShaderiv**	-	Type of shader (vertex or fragment)	2.10.1
DELETE_STATUS	B	**GetShaderiv**	*False*	Shader flagged for deletion	2.10.1
COMPILE_STATUS	B	**GetShaderiv**	*False*	Last compile succeeded	2.10.1
-	$0 * \times c$	**GetShaderInfoLog**	empty string	Info log for shader objects	6.1.8
INFO_LOG_LENGTH	Z^+	**GetShaderiv**	0	Length of info log	6.1.8
-	$0 * \times c$	**GetShaderSource**	empty string	Source code for a shader	2.10.1
SHADER_SOURCE_LENGTH	Z^+	**GetShaderiv**	0	Length of source code	6.1.8

Table 6.13. Shader Object State

Shader Engine Arithmetic Precision

- **GLSL ES precision qualifiers**
  - **lowp** (10-bit fixed), **mediump** (16-bit), **highp** (24-bit)
  - You can query via *GetShaderPrecisionFormat*()

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

- **Best Practices**
  - Declare **#precision float mediump** in your fragment shader
  - If you use **highp**, check for implementations that don't support it.

Shading Language Restrictions

- **GLSL ES allows implementations to provide less-than-full support for the core shading language**
  - 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**
  - **Stay within Appendix A guidelines**, or be prepared to fall back

Extensions

- **Implementations can advertise novel features via extension specs**
  - Defined as deltas to the core spec – can add features but not remove them
- **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
- ***prefer OES to EXT to Vendor-specific***

KHRONOS GROUP

Texture Compression

- **Use texture compression, if possible**
  - 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)

KHRONOS GROUP

Conclusion

- **Writing portable OpenGL ES 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**
  - Know (and test on) all of the major implementations
  - Don't use implementation-specific features if you can avoid it

Tip

- **Well Known OpenGL ES Optimization Tools**

  - gDebugger – OpenGL ES and OpenCL Debugger
  - Download - http://www.gremedy.com/
  - It's free !