Agenda

- OpenGL ES 2.0
  Brief Overview
- Tools
  - OpenGL ES 2.0 Emulator
  - RenderMonkey w/ OES 2.0 Support
- OpenGL ES 2.0 3D Engine Case Study
What is OpenGL ES 2.0?

- OpenGL for Embedded Systems
- OpenGL ES 2.0
  
  Fully shader-based
  Based on ES Shading Language
  Draft spec released at SIGGRAPH 05
  Spec ratified and released at GDC 07
OpenGL ES 2.0 – Widespread Industry Support

- OpenGL ES 2.0 support announced from many companies:
  - AMD
  - ARM
  - Imagination Technologies
  - NVIDIA
  - STMicro
  - Texas Instruments
  - ...and more...

- OpenGL ES 2.0 will become ubiquitous
ES 2.0 – The Problem for Game Developers

- Developers need to develop their game engines in advance of new hardware
- No hardware available today
- OpenGL ES 2.0 may require handheld developers to change their engines significantly
  - Enables more flexibility through programmability
ES 2.0 – A Development Solution

- **OpenGL ES 2.0 Emulator**
  OpenGL ES 2.0 implementation for Win32
  Allows developers to write their engines in advance of hardware

- **OpenGL ES 2.0 RenderMonkey**
  Develop OpenGL ES 2.0 shaders and effects
OpenGL ES 2.0 Emulator
OpenGL ES 2.0 Emulator

Goals

- Provide an OpenGL ES 2.0 development environment on the PC
  Minimize porting effort once hardware is available

- Leverage features/performance of desktop hardware
OpenGL ES 2.0 Emulator – What is it?

- OpenGL ES 2.0 – libGLESv2x.dll + lib
- EGL 1.3 – libEGL.dll + lib
- Khronos standard header files
- Example programs
- Utilizes desktop hardware for rendering
  Requires desktop OpenGL 2.0 hardware
OpenGL ES 2.0 Emulator – Usage Overview
OpenGL ES 2.0 Emulator - Features

- **OpenGL ES 2.0 Core API**
  - Full OpenGL ES 2.0 Implementation
  - Binary and Source Shaders

- **Optional Extensions:**
  - 10.10.10.2 Vertex/Texture Data
  - FP16 Vertices and Textures
  - 3D and Non-Power-2 Textures
  - Compressed Texture Formats
    - ETC1, ETC3, ETC5, ATI_TC
  - Occlusion and Conditional Queries
  - Depth Textures
OpenGL ES 2.0 – Demo
OpenGL ES 2.0 Emulator – Enables Developers

- More than just a prototyping tool
  Graphics code should move over easily from emulator to real hardware

- Mirrors top tier handheld developer approaches
  Prototype on the PC
  Move to handheld device as a final step
OpenGL ES 2.0 Emulator

Contact devrel@amd.com for more information

PowerVR also provides an emulator and SDK:

http://www.powervrinsider.com
RenderMonkey – OpenGL ES 2.0 Support
What is RenderMonkey?

- Shader Development Environment
- Rapid Prototyping of Shader Effects
- Multiple Shading Languages
  - OpenGL ES Shading Language
  - OpenGL Shading Language
  - DirectX HLSL
  - DirectX Assembler
RenderMonkey – Why use it?

- **Full IDE for shader effect development**
  - Programmer and artist view for rapid iteration

- **Easy integration into game pipeline**
  - Plug-in SDK for custom import/export
    - Effects, models, textures, variables, etc.
  - Support for many standard formats
    - DDS, BMP, TGA, X, OBJ, 3DS, FX

- **Encompasses all effect resources**
  Render state, texture state, variables, render targets, textures, models, etc…
RenderMonkey – What’s new?

- Support for OpenGL ES 2.0
- ES Shading Language v1.00
- ES syntax highlighting
- ES render/sampler states
- Large suite of ES examples
- User editable vertex attribute names
What is Different with ES Shaders?

- Generic vertex attributes
- User varyings
What is Different with ES Shaders? (cont’d)

Most built-in uniforms removed
- e.g. gl_ModelViewMatrix
- RenderMonkey provides equivalent user named uniforms

Default precision qualifier required for FS

Various limitations:
- Loop constructs
- Relative addressing

Extension enabling with #extension:
- 3D Textures, derivatives
What is Different with ES Effects? (cont’d)

- Reduced render state
  - Alpha test must be done with *discard*
  - No polygon fill mode
  - No fixed-function state: fog, point size, etc.

- Reduced sampler state
  - Less texture wrap modes
  - No fixed-function LOD bias
  - No texture border color
RenderMonkey - Demo
OpenGL ES 2.0 – 3D Engine Case Study
Sushi Demo Engine

- AMD’s Demo Engine

Support for:
- DX9
- DX10
- OpenGL
- OpenGL ES 2.0
Key Challenges

- Designing an engine to target multiple APIs with different feature sets
- Designing a shader-based engine
- Platform compatibility
  - Large variance in handheld platform capabilities
  - Limitations make portability a challenge
Abstracting the Graphics API

Challenge: what level to abstract the 3D API?

- Support all features of DX9/DX10/GL/GLES?
- Support common set of features?
- How to handle different shading languages?
State of the APIs - 2005

- 10.10.10 vertex data
- FP16 vertex data
- Multisample RT’s
- R/RGB texture formats
- ... 

- AA Lines/Points
- Edge Flags
- Polygon Fill Modes
- Texture borders
- Two-side polygons
- ... 

In 2005, we abstracted the DX9 feature set.
We used extensions to support missing features in OpenGL.
The choice is no longer so easy.
Especially if you add game consoles to the mix…
Abstracting the API – How We Decided

- Driven by requirements:
  - Demos must use the latest features of all APIs
  - Exposing the lowest-common denominator not an option
- Running the same demo on each API not a requirement
- Let content – rather than API abstraction – define the feature set
Abstracting the API – What We Did

- Our API abstraction looks a lot like DX10
  - Resources
  - Views
  - Geometry Shaders
  - Stream Out
  - All the latest and greatest features…

- Each API implementation supports a subset of these features
API Abstraction – Fallback Paths

- Demo engine is based off a scripting system using Lua
- Lua script provides fallback rendering paths
- Trade off: High end features vs. Content portability
  
  For Sushi, this was a fair tradeoff to make
  It might not be for you…
Sushi - Effect System

- Encapsulate essential information about rendering techniques
- Essential part of shader-based engine
  - Develop our own?
  - Use someone else’s?
    - Microsoft .FX
    - COLLADA FX
    - CgFX
- At the time, no existing solution fully fit our requirements
Sushi – Effect System Goals

- Multiple API / Shading Language Support
  HLSL, GLSL, ES SL
- Flexible support for advanced rendering techniques
  The effect system is the foundation that all the demos are built on
Sushi Effects – Cross-API Effect System

- Similar to Microsoft .FX, but multiple API support
- Expresses the following data:
  - Shaders
  - Render State
  - Passes
  - Techniques
  - Variable Bindings
Shader Authoring

- Many of our shaders authored in HLSL
- Needed a way to convert to:
  - OpenGL Shading Language
  - OpenGL ES Shading Language
- Wrote a tool for this purpose:
  - HLSL2GLSL
HLSL2GLSL

- Command-line tool and library
- Converts SM 3.0 HLSL shaders to:
  - GLSL v1.10.59 shaders
  - ES SL v1.00 shaders
- Open-source:
  - http://sourceforge.net/projects/hlsl2glsl
  - Very flexible BSD license
Sushi – Platform Portability

Handheld platforms have many constraints:

Examples:
- No Standard Template Library
- No C++ Exceptions
- Manual Cleanup Stack
- Incomplete Standard Libraries
- No Global Variables
- Limited Memory Footprint
- No Floating Point Unit
Sushi - Portability

- Standard abstraction layers
  Math, I/O, Memory, Window, etc.
- Custom template classes
  Lists, vectors, maps, etc.
- Constrained use of C++
  No exceptions
  No STL
Portability - OpenKODE

- OpenKODE
  Set of portable APIs
  Increased source portability
  Reduced platform fragmentation
  Provisional 1.0 Spec released Feb 2007

- Take a look at OpenKODE before designing your own platform abstraction layer…
Summary

Tools
- OpenGL ES 2.0 Emulator
- RenderMonkey w/ OES 2.0 Support

OpenGL ES 2.0 3D Engine Case Study
- Graphics API Abstraction
- Effects System
- Portability
Questions?

devrel@amd.com