Safety Critical

Chris Hall, Seaweed Systems

What do we mean by Safety Critical Software

- Anomalous behavior could result in loss of life and/or property
- As a result, must provide evidence that the software is safe
  - Must follow rigorous development procedures, and prove that you did so
  - Must prove that you have executed every code path
  - Must remove all unused code
  - Code execution should be deterministic

How can graphics be Safety Critical

- Graphics is the last link in the chain of information
- For example, Avionics
  - Incorrect altitude display could result in a crash
- For example, Medical Devices
  - Incorrect display could result in cutting good tissue, not bad.
- Graphics drivers typically have low-level system access.

We must subset OpenGL, in order to be able to certify it

- Full OpenGL is (practically) un-certifiable at the highest standards levels (e.g. DO178-B)
  - There is too much redundant functionality
  - There is too much functionality which requires complex software
  - Testing is too complex

Why you should care about OpenGL SC

- Subsets are the future of embedded OpenGL
- You get higher quality software
- You get an API which runs well (fast!) on all platforms
- There is no substantial loss in expressive power
- Future-proof your application

Key Goals and Philosophies

- A subset of OpenGL intended to address the needs of Safety-Critical markets should have the following properties:
  - Meet the functional requirements of target applications
  - Minimize redundancy in functionality
  - No unused functionality
  - Simplified requirements for implementors
  - Minimize dynamic allocation requirements
  - Avoid recursion

Practical Guidelines

- Where multiple functions could convey the same data to OpenGL, prefer functions which match native data types
- Where are variety of data type sizes, e.g. 8,16,24,32 bits are available, balance simplicity with the need to allow small footprint applications
- Where input parameters to functions include a range of functionality, retain only commonly used inputs

Practical Guidelines (2)

- Wherever possible, remove functionality which requires and implementation to make a decision
- Enable optimization of the entire graphics pipeline, without forcing complex logic on the application, or the implementation
- Avoid functionality which is not commonly used and/or supported in hardware.

Why not just OpenGL ES?

- Originally this was the plan
  - Same goal, different reasons
  - First cut at OpenGL ES subset came from Seaweed
- But, there was divergence in requirements
  - Anti-aliasing,“2D” operations
  - Display lists
  - More legacy SC applications

Extra Material – won't be talked about

OpenGL SC - Vertex Specification and Primitive Assembly

- Vertex Specification
  - Choice of Vertex Arrays, or Begin/End
  - Data Formats
    - Color Specification
    - Surface Normals
    - Texture Coordinates
    - Positional Data
  - Remove QUADS and QUAD_STRIP
    - Implementation as triangles anyway

OpenGL SC - Rasterization

- Anti-aliasing
  - Critical functionality
  - Multi-sample AA no good for 2D lines and points
  - Support edge AA, not multi-sample
- Points and Line Segments
  - Many different modes which could be emulated with texturing, but…
  - So - retain the full set of this functionality

OpenGL SC - Rasterization (2)

- Polygons
  - Remove QUAD, QUAD_STRIP and POLYGON
  - Remove Point and line PolygonMode
    - Useful for debugging
    - Not the best way to render gridlines over terrain
- Pixel Rectangles
  - OpenGL provides support for a very large number of pixel rectangle formats and data types
  - OpenGL provides support for data format conversion
  - OpenGL provides support for color manipulation
  - This advanced functionality is not typically supported in hardware, and we don't want to write the complex software to implement it.

OpenGL SC - Rasterization (3)

- Texturing
  - OpenGL provides 1D, 2D, 3D textures, etc.
  - 1D texturing can be emulated using 2D textures
  - 2D texturing is an essential feature
  - 3D texturing has specialized uses, is often poorly supported in hardware, and requires substantial amounts of texture memory
  - Include only 2D texturing
    - Include Multi-texture – 2 texture units
    - Color-index textures are useful for drawing map data
- Texture management
  - Applications must have ability to manage texture memory
  - Create and Delete texture objects

OpenGL SC - Rasterization (4)

- Fog
  - OpenGL provides several fog modes
  - Most common mentioned use is to hide pop-up
  - Only need to support one mode – LINEAR

OpenGL SC - Per-Fragment Actions

- Scissor
  - Essential
- Stencil, Depth Buffering
  - Essential functionality
  - Do we need all modes?
    - Stencil – yes
    - Depth – no
- Alpha Test
  - Important to keep, but cut down on combinations

OpenGL SC - Per-Fragment Actions (2)

- Logic Ops
  - Not important in this space
- Blending
  - Important, but need to cut down on combinations
- Masks (color, depth)
  - Important
- Dithering
  - Not relevant

OpenGL SC - Display Lists

- Provide a mechanism to store large amount of data in the “server” to improve performance
  - For implementations which completely isolate applications from anything which touches the hardware
- Provide a mechanism to enable optimization of inefficient (but safe) application usage of OpenGL ES SC
  - Optimization occurs once, and can run more slowly – easier to certify?

Any Questions?