
OpenGL-ES Safety Critical Overview

Chris Hall, Seaweed Systems



SIGGRAPH2006

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 a variety of data type sizes, eg 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 an 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

