

## COTS OpenGL Software Eases Certifiable Avionics Code

The OpenGL graphics API is an avionics developer's dream, but developing DO-178B certifiable code requires choosing the right COTS software.

---

Martin Beeby, Director, Software Development,  
Seaweed Systems

---

Avionics developers have a kind of envy for the standards-based solutions that are available for workstations and PCs, but are unavailable for airborne applications. This envy is particularly noticeable in the avionics display domain where the benefits of open standards-based solutions are high, but are countered by the intensive certification requirements required to meet the safety criticality of aircraft cockpit displays.

Fortunately, a number of companies who produce avionics displays are tending toward greater use of OpenGL or an OpenGL look-alike application programming interface (API). Using COTS graphics software for avionics displays can offer significant benefits to the system developer if the COTS software supplier can meet the certification challenges required by the FAA, known as RTCA/DO-178B. Even better, display development tools are now available which generate OpenGL calling C code. Such tools allow system developers to implement highly complex displays in very short time scales, all the while keeping a close eye on the certification requirements they'll eventually face.

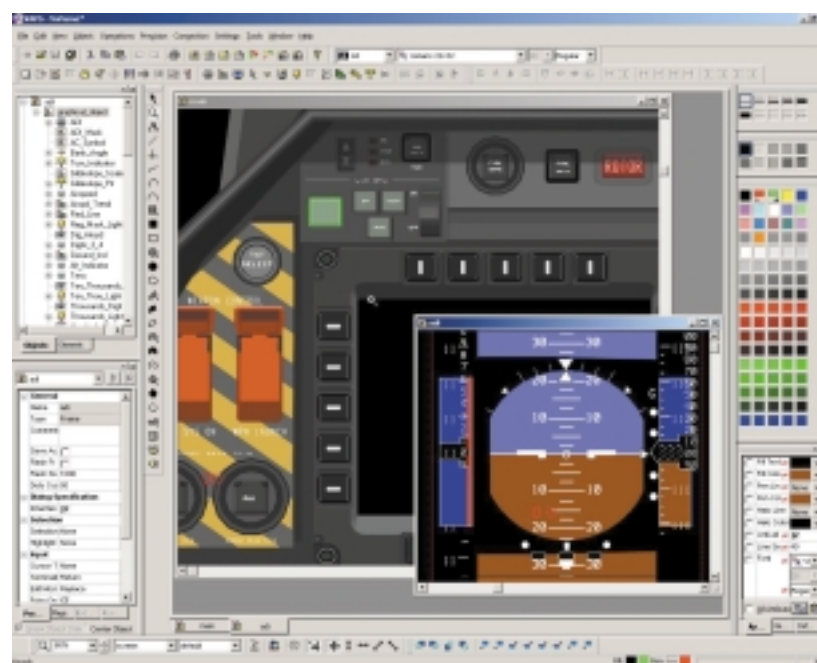


Figure 1

By coupling certifiable OpenGL with an embedded code-generator graphics package that's also designed to be certified, avionics GUIs can be easily developed. Shown here is the Qualifiable Code Generator (QCG) by eNGENUITY (formerly Virtual Prototypes). (Courtesy: eNGENUITY Technologies.)

### OpenGL for Avionics

As is typical for COTS software users, the benefits of using COTS software in avionics applications are significant for the systems developer. The benefits include, among other things, provid-

ing opportunities for lowering risk and providing improvements in time-to-market. The benefits in using COTS are potentially even greater when the software is based on an open standard, due to such a standard's already-defined syntax

Class	Environment
Level A	Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft.
Level B	Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a hazardous/severe-major failure condition for the aircraft.
Level C	Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a major failure condition for the aircraft.
Level D	Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft.
Level E	Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function with no effect on aircraft operational capability or pilot workload. There are no certification requirements for this level of software.

Table 1

Definition of DO-178B software criticality levels that range from Level A, the most critical, to Level E, the lowest level.

and semantics, its possibly wide deployment, and its available labor pool.

The OpenGL API, which is a widely deployed graphics API, is a great example of how COTS can lead to significant benefits for the avionics system developer. OpenGL is an API standard that provides 2D and 3D graphics functionality by taking primitives (points, lines, polygons, bitmaps, or images) defined by the user and converting—called “rendering”—them into an image on a display. OpenGL is extensively supported by graphics accelerator hardware that allows the developer to implement very complex and powerful display applications with high refresh rates (see sidebar: OpenGL Overview).

Many avionics display developers are motivated to utilize OpenGL in their applications for following reasons:

**OpenGL is ubiquitous** – This gives OpenGL two advantages over a proprietary API. The first is its ubiquity in platforms. Because OpenGL runs on a variety of platforms, it means that application writers can develop on a host platform (for example, Microsoft Windows) and then port the application to the target platform with minimal effort. It’s even possible to use the same graphics hardware accelerator on the host platform and on the target platform, making the host that much closer to the target environment.

Secondly, OpenGL has a ubiquitous labor pool. That is, because OpenGL runs on many platforms, there are a lot

of skilled OpenGL application writers available. It should be noted that the ubiquity of OpenGL really has no effect on its certification other than to stimulate the success of this COTS standard, possibly making it more compelling for avionics applications.

**Graphics acceleration** - Hardware exists that accelerates OpenGL, thus enabling high performance. And, there is a non-trivial amount of hardware that accelerates the entirety of the OpenGL API. This is important, because any rendering done without hardware assistance in a certified environment would be slow. Furthermore, a software-only rendering implementation would be very complex, making the software rendering stack troublesome for the FAA to certify for use.

**OpenGL is leveraged** - There are a number of tools that exist that help the designer of graphic avionics applications generate the displays, or “formats” that are displayed (Figure 1). The tools typically run on workstation platforms and the engineer tweaks the format until it looks “just so” and it behaves “just so” (that is, so that it responds to the inputs that an avionics display needs to respond to in an appropriate manner). Once the engineer is happy, a button is pressed and C code is emitted which implements the format. The emitted code has to call some sort of graphics library, and more often than not, the graphics library that is called is either OpenGL or something

that’s very similar to OpenGL. Here, it is desirable to select development tools that either output certifiable code, or at least have certification in mind as an eventual outcome.

But even though OpenGL is very well suited to embedded avionics display applications, it was never designed with safety or high availability in mind. For OpenGL to really fulfill its potential in the avionics display domain there are certification issues that need to be addressed.

### Certification to RTCA/DO-178B

The FAA’s certification is required for any software used on or connected to an aircraft. Without this certification, the software cannot be used. The FAA characterizes software as belonging to one of five levels of criticality: “A” through “E”. Level A is the most critical and applies to systems, which if they fail, would probably lead to a catastrophic failure of the aircraft. Level E is at the other end of the spectrum and refers to systems, which if they fail, will probably go unnoticed (Table 1).

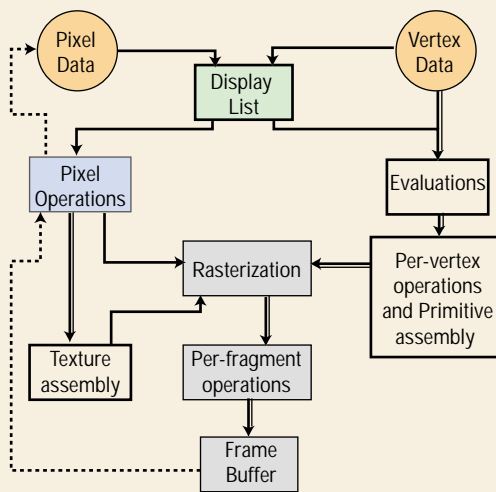
Aircraft displays that provide critical parameters to the pilot, such as Altitude, Attitude, and Speed are required to be certified to the highest level of criticality, level A. As the criticality of the software is increased, so is the rigor in the development and verification requirements in DO-178B, as illustrated in Table 2.

For software, the FAA mandates adherence to guidelines known as

## OpenGL Overview

The OpenGL graphics system is a high performance, window system independent, software interface to graphics hardware. OpenGL provides a very full-featured API to the applications developer for efficiently generating high quality color 2D and 3D graphics. OpenGL is

The state machine can further be thought of as providing the following features: vertex, pixel, and primitive operations (the primitives being points, line, and polygons), display lists, texturing, lighting, modeling, projection, and viewport transforms, perspective, clipping, a variety of color models, display lists, and Z, accumulation, scissor, and alpha buffers.



Simplified OpenGL State Machine, showing the order of operations to produce an image in the frame buffer.

These features are offered in an orthogonal manner; that is, the use or non-use of a feature does not depend upon the use or non-use of another feature. This orthogonality has the advantage that users combine features in ways undreamed of by the designers of OpenGL—which means that OpenGL is used in unanticipated problem domains. This orthogonality has the further advantage that it makes subsetting the API achievable and meaningful (by meaningful, we mean that the retained portions of the API retain all the semantics held in the un-subsetted API).

typically used in conjunction with graphics acceleration hardware to provide performance suitable to real-time applications.

OpenGL was developed by Silicon Graphics Inc. (SGI) based on a decade of experience in computer graphics hardware and software design. Control for the OpenGL standard has been turned over to the OpenGL Architecture Review Board (ARB), which consists of leaders in the computer industry. Its founding members included DEC, IBM, Intel, Microsoft and Silicon Graphics. The collective vision underlying OpenGL is that through a common, fast software interface for programming graphics, 3D graphics can rapidly become a mainstream computer technology.

The authors of the OpenGL API prefer for its users to think of the API as providing a state machine, which happens to render 3D graphics (see Figure).

The authors of the OpenGL API also wanted the API to be portable among a variety of environments. To this end, they deliberately kept any mention of “window system” out of the OpenGL API. By avoiding reference to, or a dependence on, the window system within the API, they skirt neatly around portability issues with respect to window systems. This is one of the reasons why the OpenGL API can exist in Microsoft Windows, the X Window System, MacOS, BeOS and OS/2.

Besides a studied refusal to refer to the host window system, OpenGL also segregates a few system-dependent functions regarding buffer swap and management of the data “bundle” which carries the state of the OpenGL state machine. The same structuring which allows for environment portability at the workstation-class also allows for environment portability for embedded, “non-standard” systems.

“RTCA/DO-178B” (usually referred to simply as “DO-178B”). These guidelines do not mandate processes or documentation formats, but instead provide objectives, activities, and evidence that must be satisfied by the software developer. Fundamentally, the FAA needs to know that the software implementation is characterizable and bounded in the time domain (that is, how long do things take to do), in the space domain (how much memory do things use) and what is the software required to do.

These requirements must be met under all required conditions and permutations of the software’s behavior. And, the objectives and activities in DO-178B require the software developer to clearly define what is required of the software. In addition, the software developer is required to produce evidence through requirements-based testing that the software has met its requirements. The verification objectives also require the software to be robust, and that requirements-based testing executes every statement, decision, or modified condition/decision (depending on software level) in the source code.

Modified condition/decision coverage is quite complex, and means that all of the software’s loops, branches, entries and exits have been tried and verified. More specifically, what’s tested is that every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken all possible outcomes at least once, every decision in the program has taken all possible outcomes at least once, and that each condition in a decision has been shown to independently affect that decision’s outcome. A condition is shown to independently affect a decision’s outcome by varying that condition while holding fixed all other possible conditions. Any code not reachable during testing is dead code and must be removed from the software.

The trouble is that OpenGL was never designed with safety of implementation in mind. Indeed, commercially available OpenGL implementations, of some 200 k lines of code and 300 API functions, are probably uncertifiable to

high levels of DO-178B without significant modifications for the following reasons:

- High reliance on dynamic memory usage
- High reliance on operating system facilities making behavior dependent of operating system behavior
- Code constructs that are not readily verifiable
- Dead code within the driver

To overcome the drawbacks of existing implementations would necessitate developing from scratch a certifiable OpenGL implementation. Typically software developed to meet DO-178B takes around one person-month to develop 125 lines of code with all the supporting processes and documentation. So an implementation of 200 k lines of code would take about 1600 person-months, or 133 person-years!

### Subsetting and Hardware Make Code Manageable

In order to provide a useful rendering layer, while still fulfilling the certification requirements of DO-178B, the preferred route is to subset the OpenGL API and only include the portions of the OpenGL API that the application needs. Avionics applications that want to use OpenGL as their rendering layer typically have two display needs. The first need is for simple 2D graphics, such as showing various forms of dials and gauges in digital form. The second need is for more complex 3D graphics for out-the-cockpit terrain painting views. Examples here include drawing a kind of highway-in-the-sky and aiding in collision avoidance either between aircraft, or between aircraft and the ground.

Small 2D and 3D subsets of the OpenGL API can address the majority of avionics display needs. By subsetting the API, the implementation effort is lighter, the set of requirements is smaller, and the amount of testing is lessened. At the same time the documentation evidence that shows everything was done according to DO-178B guidelines is more manageable.

Objective	Software level			
	A	B	C	D
Test procedures are correct	I	S	S	
Test results are correct and discrepancies explained	I	S	S	
Test coverage of high level requirements is achieved	I	S	S	S
Test coverage of low-level requirements is achieved	I	S	S	
Test coverage of software structure (modified condition/decision coverage) is achieved	I			
Test coverage of software structure (decision coverage) is achieved	I	I		
Test coverage of software structure (statement coverage) is achieved	I	I	S	

Legend: I : The objective should be satisfied with Independence (requires the activity to be performed by someone independent from the developer.)  
 S : The objective should be satisfied. Blank : Satisfaction of the objective is at the applicant's discretion.

Table 2

DO-178B Verification Requirements vs Software Level. For Level A (the most critical), all objectives require an independent verification. For Level E (the least critical and not shown in the table), satisfying the objectives is not required by the FAA and is completely at the discretion of the software developer.

Just to be clear, the subsetting work does not only benefit those who are creating a certifiable OpenGL implementation. The subsetting benefits the customer as well because the cost is cheaper (due to less work), the route to certification more certain (due to lower complexity), and the time scales shortened (due to less coding and testing).

When you get down to it, the amount of effort put into an OpenGL implementation for the highest safety level consists of about 5% driver development and 95% DO-178B activities and evidence generation. It is not that the driver development is not important; it most certainly is because avionics customers absolutely require performance. But what is even more paramount than performance is correctness. The most favorable path to correctness is comprehensive planning, combined with complete and demonstrable analysis, and a

perfectly documented and ultimately flawless execution of the driver development.

To further minimize the software effort, the hardware element of the solution has to have complete coverage of all OpenGL rendering functionality. This requirement is not onerous because graphics hardware available for use in the avionics market—for example, the Permedia 3 and Glint graphics accelerator product lines from 3Dlabs—do implement the complete OpenGL engine in hardware. Such complete hardware rendering assistance means that rather than requiring complex software to effect rendering, complex hardware can be substituted along with much simpler software to drive the hardware. By minimizing the software required to drive the display, the time, effort, and risk of certifying the software is minimized.

Key Attribute of Subset
Architect the implementation to be static in its memory usage.
Minimize the dependence on operating system facilities to allow deterministic behavior and portability across embedded operating systems.
Enforce coding standards that lead to simple, deterministic and verifiable code.
Use requirements-based testing to verify implementation to ensure no dead code exists within the implementation.
Address the DO-178B objectives from the beginning of the software development (as opposed to leaving certification as an "afterthought").
Carry out the DO-178B activities and produce the evidence required during development.

**Table 3**

Attributes of an OpenGL subset designed for DO-178B certification. (Courtesy: Seaweed Systems)

## COTS Meets the DO-178B Challenge

Once the software product has been developed, it is then necessary to package the software and all the required certification evidence and data for customers. It should be noted that the FAA only gives certification to complete systems. The FAA examines the use of each software component with respect to a system, and examines how the software component has been validated as part of the system. Even though the software component may have previously been included in systems that have been certified, it does not necessarily follow that the software component will be certified in the new system.

The use of the software component within the system is a key element of the overall system safety. For example, it does not help the certification effort if a system that uses a previously certified multi-tasking operating system has the potential for deadlock between tasks. While this example is an over-simplification, it serves to illustrate that usage of a software component is more important than prior certification history. For this reason, it is not possible for COTS vendors to get stand-alone certification for any software components that they supply. Even though it is not possible to certify a COTS software component in isolation, it is possible to package that component in a form that facilitates certification by a systems developer.

Subsetting COTS OpenGL imple-

mentations offers the easiest path to software certification. Among the COTS vendors providing OpenGL implementations designed for certification, Seaweed Systems has developed from scratch implementations of 2D and 3D subsets of the OpenGL API. These implementations have been targeted to meet the highest certification requirements for avionics display systems through the attributes shown in Table 3.

What Seaweed Systems has done with OpenGL is not unique. There is other COTS software that is certifiable, which either supports an OpenGL renderer or requires the presence of such a renderer. But besides the OpenGL code itself, a well-designed piece of COTS software results in the greatest benefits to systems developers when a suite of COTS products is used in combination. Products like eNGENUITY Technologies' Qualifiable Code Generator (QCG) can be used to auto-generate certifiable, embedded code for the display applications. QCG requires a rendering layer like Seaweed Systems' Certifiable OpenGL implementation.

OpenGL implementations also rely on, and applications can benefit greatly from, facilities provided by embedded operating systems such as Green Hills' INTEGRITY OS, and Wind River's VxWorks OS, both of which are certifiable. Hardware vendors like Radstone and Dy 4 are able to support the hardware requirements for avionics systems and utilize graphics devices from 3Dlabs.

For hardware, particularly for the graphics accelerator devices, the hardware vendor also needs to be ready, willing and able to assist the customer through the entire life cycle of the product and provide information critical to the application developer's certification submission. Some hardware vendors who are not dedicated to the avionics market are very protective of their technology and will not divulge certain information about their architecture or processes. Some even use technologies that are uncertifiable. These vendors do not provide viable solutions for systems developers designing and deploying avionics displays. ■■

Seaweed Systems  
Woodinville, WA.  
(425) 895-1727.  
[www.seaweed.com].

3Dlabs  
Sunnyvale, CA.  
(408) 530 4700.  
[www.3dlabs.com].

Dy 4 Systems  
Kanata, Ontario, Canada.  
(613) 599-9199.  
[www.dy4.com].

eNGENUITY Technologies  
Montreal, Quebec, Canada.  
(514) 341-3874.  
[www.engenuitytech.com].

Green Hills Software  
Santa Barbara, CA.  
(805) 965-6044.  
[www.ghs.com].

Radstone Technology  
Montvale, NJ.  
(800) 368-2738.  
[www.radstone.com].

Silicon Graphics  
Mountain View, CA.  
(650) 960-1980.  
[www.sgi.com].

Wind River Systems  
Alameda, CA.  
(510) 748-4100.  
[www.windriver.com].