



Seaweed
Systems

AVIATION QUALITY COTS SOFTWARE: REALITY OR FOLLY

Martin Beeby, Seaweed Systems, Woodinville, WA

Abstract

Many times the choice of COTS can mean additional risk for systems developers, with not insignificant efforts being required to assure the quality necessary for it to be workable as part of a certification submission.

There is a new wave of COTS software and hardware vendors that understand the rigorous requirements of system certification and they are providing the avionics developer with true COTS solutions. These are not commercial solutions “shoe-horned” into avionics applications, but open standard based products that have been developed specifically to meet the requirements of avionics certification.

Using the Seaweed Systems OpenGL subset API as an example, this paper describes in detail, the techniques used to take an open standard graphics API (OpenGL) and turn it into an avionics COTS product that can offer significant savings for systems developers. The paper will describe how a from-scratch implementation has all the quality, safety, and performance attributes required to meet the stringent certification requirements for airborne systems.

COTS products specifically developed to meet the certification requirements of avionics systems significantly reduce the risk and offer a huge cost advantage over in house development. This paper will show cost models that clearly illustrate the risk and development advantages of using COTS targeted for avionics certification.

The paper will provide key questions to ask COTS vendors to determine whether they are offering a solution that increases, or decreases, the risk for systems developers. By making the right choice of COTS products the

application developer can realize very significant benefits in risk, cost, and time to market.

Introduction

Today’s developers of avionics software face cost and time-to-market hurdles that the commercial system developer might find familiar from the past. These hurdles include the time and effort involved in the design, development, documentation and testing of a custom solution. Avionics developers have a kind of “envy” for open standards-based solutions available in other technical arenas (e.g. on workstations and PCs) but are unavailable to them. 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 due to the safety criticality of aircraft cockpit displays.

A number of companies who produce avionics displays are tending toward greater use of OpenGL or an OpenGL look-alike API. 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. A certifiable OpenGL driver is a prerequisite for the successful deployment of avionics applications developed using these tools.

COTS graphics software for avionics displays can offer significant benefits to the system developer if the COTS software supplier can meet the certification standards required by the FAA. (The guidelines used by the FAA, and other aviation certification authorities all around the world, are known as RTCA/DO-178B).

What is OpenGL?

The OpenGL graphics system is a high performance, window system independent, software interface to graphics hardware. OpenGL provides a very rich API to the applications developer for generating high quality color 2D and 3D graphics efficiently. OpenGL is 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 of 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. You do not need to look very far to see evidence that OpenGL has realized this goal.

The authors of the OpenGL API prefer for its users to think of the API as providing a state machine (see figure 2) which happens to render 3D graphics. The state machine can further be thought of as providing the following features: vertex, pixel, and primitive operations (the primitives being points, lines, 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. 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 can 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).

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 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.

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 providing opportunities for lower risk, lower cost and providing improvements in time to market. The benefits of using COTS can be even greater when the software is based on an open standard. This is due to the standard’s already-defined syntax and semantics, its potentially widespread deployment, and its available development labor pool.

The OpenGL API is a great example of where 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, or rendering, them to an image on a display. OpenGL is extensively supported by COTS graphics accelerator hardware that allows the developer to implement very complex and powerful display applications with high refresh rates.

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

a) OpenGL is ubiquitous, which gives OpenGL two advantages over a proprietary API:

1) Ubiquity in platforms,

Because OpenGL runs on a variety of platforms, it means that application writers can develop on a host platform (e.g. Windows) and then port the application to the target platform with minimal effort. It is even possible to use the same graphics accelerator on the host platform and on the target platform, making the host that much closer to the target environment. This is a significant advantage since the developer is able to better judge target performance at development time.

2) Ubiquity of labor pool.

Because OpenGL runs on many platforms, there are a lot of skilled OpenGL application writers available.

b) Graphics acceleration,

Hardware exists which accelerates OpenGL, thus enabling high performance. There is a non-trivial amount of hardware that accelerates the entirety of the OpenGL API. This is important, as any rendering done without hardware assistance in a certified environment would be slow, and because of the complexity of the software rendering stack, troublesome for the FAA to certify for use. Further, these products are continuously evolving to guarantee developers access to the latest and most powerful functionality and performance that will help them “future-proof” their applications.

c) 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. 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, 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 is very similar to OpenGL.

Even though OpenGL is very well suited to embedded avionics display applications, it was never designed with safety, footprint 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 of software 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 (see table 1 for explanation of criticality levels). Aircraft displays that provide critical parameters to the pilot (ones that the pilot relies upon for in-flight decisions), 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 “RTCA/DO-178B” (usually referred to simply as “DO-178B”). These

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.

Table.1

Definition of criticality level.

guidelines do not mandate processes or documentation formats, but provides objectives, activities, and evidence that must be satisfied by the software developer. Fundamentally, the FAA needs to know 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 does what it is required to do. That is to say that under all required conditions the behavior of the software can be characterized. 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 that, through requirements based testing, the software has met its requirements. The verification objectives also require the software to be robust, and that requirement based testing executes every statement, decision, or modified condition/decision (Depending on software level) in the source code. Any code not reachable during testing is dead code and must be removed from the software.

OpenGL was never designed with safety of implementation in mind. Indeed, commercially available OpenGL implementations which were designed primarily for flexibility, are uncertifiable to higher levels of DO-178B without significant modifications for the following reasons:

- a) High reliance on dynamic memory usage.
- b) High reliance on operating system facilities, making behavior dependent of operating system behavior.
- c) Code constructs that are not readily verifiable.
- d) Dead code within the driver.

To overcome the drawbacks of existing implementations it would be necessary to develop from scratch a certifiable OpenGL implementation. Typically software developed to meet DO-178B takes around 1 person month to develop 125 lines of code with all the supporting processes and documentation. So an implementation of, for example, 300k lines of code would take about 2400 person months, or 200 person years!

In order to provide a useful rendering layer, while still fulfilling the rigor of DO-178B, the sensible route is to subset the OpenGL API to only include the portions of the OpenGL API which the application needs. Avionics applications that want to use OpenGL as their rendering layer typically have two display needs:

- a) simple 2D graphics applications show various forms of dials and gauges in digital form.
- b) more complex 3D graphics for, orbitalADIs, enhanced and synthetic vision, situational awareness and a multitude of weapons systems

targeting applications: such uses could include 3D digital maps and TAWS for example.

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, the amount of testing is lessened, and the documentation evidence that shows everything was done according to DO-178B guidelines is more manageable.

Just to be clear, the subsetting work not only benefits those who are creating a certifiable OpenGL implementation, it benefits the customer 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).

The Cost of DO-178B implementations

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 and reliability. 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.

¹Modified Condition/Decision Coverage — 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 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.

Objective	Software Level			
	A	B	C	D
Test procedures are correct	I	S	S	
Test results are correct and discrepancies explained	I	S	S	S
Test coverage of high level requirements is achieved	I	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. **S** : The objective should be satisfied.
Blank : Satisfaction of the objective is at the applicant's discretion.

Table.2

Verification requirements vs software level..

When estimating a from scratch DO-178B software development there are a lot of rigorous development processes, volumes of supporting documentation to be generated and considerable verification activities required to assure product quality and reliability. In addition to the development costs, there is also a risk cost and dilution of application development to be assessed. The key cost drivers can be identified as:

-  Experience of 178B compliant development processes
-  Experience of OpenGL implementations
-  Experience of graphics processor and CPU architectures
-  Experience of embedded operating systems
-  Size and Complexity of the code
-  Level of verification/reliability required
-  Level of documentation required

Cost Models can provide a useful guide as to what it takes to write software to meet RTCA/DO-178B standards. Two models that are popular are COCOMO and an experience metric approach.

COCOMO

Originally developed by Dr. Barry Boehm in 1981, the Constructive Cost Model (COCOMO) [1] has been developed and enhanced over the years to reflect modern development practices and experience [2].

The COCOMO model is widely used for estimating software development efforts and provides a good rough order of magnitude of software costs. The model provides a number of parameters to tune so that the output is relevant to the type of software being developed and the experience of the development team. Several models are freely available:

sunset.usc.edu/research/COCOMOII/index.html

www.jsc.nasa.gov/bu2/COCOMO.html

For our estimate the following assertions were used in COCOMO II:

1. 2D certifiable subset of OpenGL API is approximately 12,500 loc
2. 3D certifiable subset of OpenGL API is approximately 25,000 loc
3. DO-178B demands:
 - a. Greater attention to software design for safety
 - b. Significant documentation overhead
 - c. Comprehensive verification
4. Applications typically demand:
 - a. High performance
 - b. Short development time

Plugging all that into COCOMO II gave the results that have been summarized in Table 3.

	2D Subset			3D Subset		
	Best	Median	Worst	Best	Median	Worst
Size (loc)	12500	12500	12500	25000	25000	25000
Efficiency (loc/mm)	186.05	129.03	111.11	177.78	125.00	111.11
Total effort (hours)	10750	15500	18000	25500	32000	36000
Cost Rate (\$/hour)	75	75	75	75	75	75
Total Cost	\$806,250.00	\$1,162,500.00	\$1,350,000.00	\$1,687,000.00	\$2,400,000.00	\$2,700,000.00

Table.3

COCOMO II estimate results for 2D and 3D subsets.

Experience Metric Cost Model

The Experience cost model is very simplistic and is based on historic development team productivity metrics being extrapolated for the new development. This model does require historical productivity metrics for comparable processes and developments, but does not need to be sophisticated.

Our experience shows that a from-scratch DO-178B level A development achieves a productivity of 125 lines of code per man month. This productivity is in line with what COCOMO II predicts and gives the results in table 4.

What do the Cost models mean?

Both cost models illustrate how big and costly such a “from-scratch” development can be. If this is to be an additional component of an application development, it is certainly going to be a significant distraction from the application development itself. A from-scratch development of this size also carries some risk. COTS solutions can offer significant cost and risk reduction by amortizing the development costs over a number of customers and reducing risk through fixed pricing.

Despite that, many organizations often do not believe the cost estimates and opt for in house development over COTS based solutions. Common reasons for not believing the estimate include:

- the measurements they have don't reflect total cost of development

- the organizations do not have a repeatable process
- they never make the same thing twice
- development can be spilt over several cost centers and the total cost is never measured
- this is sensitive data, and under measuring can be advantageous

It is also worth noting that neither of the cost models presented includes costs for maintenance and support, which can prove to be a further significant burden on application development resources for many years to come.

Minimize Size and Complexity

We have illustrated that subsetting is the starting point for bringing Open Standards based solutions to the avionics developer. There are other activities necessary to achieve certifiable OpenGL for avionics.

We should not forget that the hardware element of the solution can have complete coverage of all OpenGL rendering functionality. While not eliminating the need for software, there is 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, combined with much simpler software

to drive that hardware.

By minimizing the software required to drive the display, we minimize the time, effort, and risk of certifying the software.

By designing the software from scratch with DO-178B constraints in mind and taking full benefit of subsetting, it is possible to vastly reduce the size of a useable OpenGL driver. Table 5 illustrates how the Seaweed Systems' Certifiable driver represents a small fraction of the size of a full OpenGL driver.

Table 5 also illustrates the benefit of from-scratch development, with comment density being increased from 26% to 54%. This leads to significantly more readable, verifiable, and maintainable code.

In addition to simply reducing the code size, reducing the code complexity also reduces the development cost by making verification much less onerous. Code complexity is directly related to the effort involved in achieving the level of structural coverage required by DO-178B. McCabe cyclometric complexity is a measure of the number of linearly independent paths through a program. When MC/DC coverage is being sought, the level of effort required to achieve 100% coverage is exponentially related to the code complexity. Typically, a DO-178B development will aim for McCabe Cyclometric complexity of 10 or less. Full OpenGL drivers originally designed for non-embedded environments are uncertifiable because of average complexities approaching 10 and peak complexities in the 100s. By developing a from-scratch implementation with this constraint in mind, Seaweed Systems has achieved very significant reduction in average and peak complexity. This has resulted in a driver that is readily certifiable up to the highest levels of structural coverage.

As a byproduct of reducing complexity, we have seen that less paths through the code means fewer decisions and therefore more efficient and deterministic code.

	2D Subset	3D Subset
Size (loc)	12500	12500
Efficiency (MD/LC)	32.00	32.00
Efficiency (loc/mm)	125.00	125.00
Total effort (hours)	16000	32000
Cost Rate (\$/hour)	75	75
Total Cost	\$1,200,000.00	\$2,000,000.00

Table.4

Experience metric estimate results for 2D and 3D subsets.

	Full OpenGL Driver	Certifiable Driver	Fraction
Files:	499	61	12.2%
Functions:	5181	287	5.5%
Lines:	415564	20402	4.9%
Break Lines:	54565	2168	n/a
Lines Code:	303947	12744	4.2%
Lines Comment:	79117	6956	n/a
% Content:	26.0%	54.0%	207.7%

Table.5

Code size comparison.

COTS meets the DO-178B challenge

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 by

- Architecting the implementation to be static in its memory usage.
- Minimizing the dependence on operating system facilities, to allow deterministic behavior and portability across embedded operating systems.
- Enforcing coding standards that lead to simple, deterministic, and verifiable code.
- Using requirements based testing to verify the implementation in order to ensure that no dead code exists within the implementation.
- Addressing the Level A DO-178B objectives from the beginning of the software development.
- Carrying out the DO-178B activities, and producing the evidence required during development.

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 how the software component has been validated as part of that 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: a system that uses a multi-tasking operating system, that has previously been included in successful certification submissions, does not work if the application has the potential for deadlock between tasks. While this example is an over simplification, it serves to illustrate that usage of 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 systems developers.

Many COTS vendors developing products for the avionics market understand the constraints and rigors required in the domain of avionics software. There are also COTS vendors who do not understand the market constraints and offer solutions that can increase risk in application development. In order to identify good COTS suppliers from the not so good, we offer the following sets of questions:


General questions


How long have they been in business?

Are they stable or growing in size?


 Shrinking could indicate instability/risk


Do they have a good reputation?

 Good customer reference list

 Reputation for delivering on commitments


Do they possess integrity and act professionally?


 Do they make you feel you are important to them?


 Do they listen to your suggestions and offer advice?


Do they possess unique skills/products that you find difficult to obtain or justify holding internally?


Domain specific questions


 Are they focused on your industry?


 What % of their revenue comes from Aerospace?


 What % of their workforce focuses on Aerospace?


 Could they do without Aerospace business?


 Do they have references specific to your business?

 Do they understand your business, including support?

 What is their support record like?

 Do they match your quality needs?

 Do they deliver products under recognized quality system or to a relevant standard?

 Do they possess knowledge of and participate in relevant industry standards?

Is their solution COTS or custom development?

Is the product developed from scratch or reverse-engineered?

Is their technology current with your demands?

Is their product direction in line with yours?

Do they have the relevant experience?

Summary

What Seaweed Systems has done with OpenGL is an example of what COTS vendors are doing to really meet the needs of the avionics application developers. There are other COTS software that are certifiable, which either support an OpenGL renderer or require the presence of such a renderer. A well-designed piece of COTS software results in the greatest benefits to systems developers when a suite of similarly well-designed COTS products is used in combination. Products like eGENUITY Technologies' Qualifiable Code Generator (QCG) can be used to auto-generate certifiable, embedded code for display applications. QCG requires a rendering layer, like Seaweed Systems' Certifiable OpenGL implementation. OpenGL implementations rely on, and applications can benefit greatly from, facilities provided by embedded operating systems such as Wind River's VxWorks OS, Green Hills' INTEGRITY OS or LynuxWorks' LynxOS OS, all of which are certifiable. Hardware vendors like Radstone and Dy4 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 require use of technologies, where those technologies are uncertifiable. These vendors do not provide viable solutions for systems developers in this arena.

It is only with products from vendors, like Seaweed Systems, 3Dlabs, Wind River, Green Hills, LynuxWorks, eGENUITY, Radstone and DY4, who truly understand and support the certification requirements for avionics systems, that the system developer can leverage COTS solutions to their maximum benefit. A clear set of COTS hardware and software products is emerging that truly support avionics developers and this is good news for everyone.

[1] Barry W.Boehm, 1981, Software Engineering Economics, Prentice Hall.

[2] Barry W.Boehm et al, 2000, Software Cost Estimation with COCOMO II, Prentice Hall



Seaweed Systems Inc
Woodinville, WA 98077
+1.425.895.1721
www.seaweed.com