White Paper

6 Ways Software Rendering Boosts Embedded System Graphics

OpenGL® SC running on a proven, safety-critical compliant software rendering platform creates unique possibilities for embedded system graphics. Designers running into barriers when looking at implementing graphics with challenges like DO-178B/C certification, radiation hardening, texture engines and overlays, FPGA implementations, distributed displays, and SWaP have a new option in IGL®.

Inside is a look at the advantages OpenGL provides, one key item embedded developers will likely want to avoid in a graphics implementation, and how IGL brings a proven solution for software rasterization into the realm of safety-critical systems. Then, the six “boosts” for critical situations where IGL outperforms alternatives are explored.

September 2012

ENSCO Avionics, Inc.

Introduction

Graphic displays are becoming part of more and more everyday devices, from smartphones to cars to appliances and home automation devices. This is driving the expectations of users, who are looking for clean, fast, simple-to-use graphics when choosing a device. Well-executed graphics interfaces contribute to faster learning and ease-of-use of devices, and the efforts of designers to achieve a superior user experience have a large payoff.

In safety-critical fields, such as avionics, defense, medical, industrial, and others, the importance of graphics goes well beyond user experience. Decisions which can impact significant outcomes are based on a real-time presentation of graphical information, which must be accurate and trustworthy at all times. Hard failures are unacceptable, but soft failures such as slowing down or skipping frames entirely can also lead to dire consequences. Many systems require certification to a safety-critical standard such as DO-178B/C, which attests to the trustworthiness of the software.

Some safety-critical environments present even bigger challenges for designers of embedded systems. The constraints of performance, and system and life cycle cost come into play. In space-borne applications, radiation tolerance is mandatory, while in defense applications size, weight, and power (SWaP) are optimized. These requirements make the use of many commercial-grade hardware graphics processor units (GPUs) less attractive, or even infeasible in some cases.

IGL is a safety-critical, DO-178B certifiable implementation of an OpenGL® SC 1.0 software rasterizer. By using OpenGL SC and software rendering, designers of embedded systems gain advantages in determinism, reuse, configuration longevity, certification, and system size.
This white paper explores how software rendering, such as implemented in IGL, can solve problems in safety-critical embedded systems. After a short overview of OpenGL SC, the key limitation of gaming-class graphics in safety-critical applications is discussed. An architectural overview of IGL follows, and then six ways software rendering boosts embedded system graphics are illustrated.

OpenGL Brings Reusability and Realism

OpenGL is an industry-standard graphics API, maintained by the Khronos Group (www.khronos.org), which first gained popularity in visualization workstations as a way to render complex 2-D and 3-D scenes with many objects and detailed visual effects, and yet keep control over graphics primitives. OpenGL applications create nearly the same visual effects on any platform and operating system that support the standard.

OpenGL applications are a stream of commands that act on a data set. Commands can draw an object, create an effect, or change a configuration within the graphics pipeline. Objects are lines or polygons. Effects in the current version, OpenGL 4.3, include texture mapping, alpha blending, fog, anti-aliasing, and more.

The powerful graphics effects, cross-platform support, and code reusability achievable with OpenGL attracted two groups of computer technologists: game developers, and hardware GPU developers. Game developers could create source for OpenGL, which could immediately be deployed on any supporting platform, and could be sustained as hardware platforms evolved quickly. Hardware GPU developers — companies including AMD™, NVIDIA®, and S3 Graphics —

concentrated on developing engines that could efficiently execute OpenGL graphics pipelines with more objects and more effects, and continue to develop increasingly faster GPUs.

As OpenGL continues growing in capability, more and more developers jump on the standard. Gamers press for increasing visual realism, and the OpenGL standard continues to evolve by adding advanced effects. Smaller GPU cores, suitable for lower power system-on-chip processors, have found their way into smartphones and tablets, and OpenGL has also become a standard in mobile designs. The vision of multiplatform, realistic graphics is compelling.

Critical Objects Must Be In Sight

Adding more realism in OpenGL solved issues for many market segments, but created a big problem for the safety-critical community. In order to be trustworthy under all conditions, safety-critical applications must abide by a strict sense of determinism. Real-time isn’t just a benefit in these applications; it is an absolute requirement.

The superscalar, multithreaded technology used within hardware GPU architectures creates a high performance potential and the ability to manage many objects and effects, but does little to guarantee the delivery of one or more objects through the graphics pipeline within a fixed time window. This is manifested in a phenomenon most gamers have observed firsthand: *lag*. Of course lag can result from waiting on data from a distributed application, but more often lag indicates the GPU graphics pipeline is temporarily overwhelmed with everything being asked of it.

When a game session begins, the pipeline is well under control. There is usually some type of map with scenery, and complex symbols representing characters or vehicles begin moving around. But as the game advances and action ensues, things get complicated: events like explosions, splashes, weather, smoke, and changing scene detail create a mix of more and more objects the GPU has to render. If the gamer has effects settings relatively high to achieve more realism, those advanced effects are being applied to a rapidly increasing number of objects in play.

At some point, the GPU buckles under the sheer weight of all the effects in the pipeline, and one of several things happens to the rendering. Portions of the scene, inevitably objects of interest like opponents, can appear or disappear unexpectedly or warp in location. Even worse, frame rates slow to a crawl, or a complete stop. When the display resumes at the expected frame rate, frames are missing, with whatever action happened in them gone.

While lag is an irritating loss of control for gamers, lag can be hazardous and is always unacceptable for safety-critical applications. There are several ways to mitigate lag: buy a faster hardware GPU, turn down the effects settings, or avoid using some effects altogether. In short, the first two options aren’t guaranteed to eliminate lag from occurring, and are no help while lag is in progress.

The safety-critical community considered this when creating OpenGL SC (the SC stands for safety-critical), which is based on OpenGL 1.3. By restricting OpenGL SC operations to a mostly effects-free profile, safety-critical applications benefit from smaller code size and less complexity. Features such as a texture matrix, compressed textures, multisampling, dithering, and fog are eliminated, and general texturing and alpha blending are restricted to a subset of capability. Features added to OpenGL SC are display lists, draw pixel and bitmap capability, and line anti-aliasing.

With a simpler code base and limited effects to manage, a software rasterizer running OpenGL SC becomes an intriguing approach to creating graphics in a safety-critical embedded system.

How IGL Works

IGL is a pre-compiled, C-callable library, and is platform independent with support for a variety operating systems and processor architectures. Its API conforms to OpenGL SC 1.0, with the addition of several configuration and common utility functions.

The simple design of IGL features three main subsystems, as shown in Figure 1. Applications request IGL services by feeding OpenGL commands into a state machine that maintains information about the configuration and graphics data. The rendering engine then takes over, performing the necessary transformational computations on the data set. Finally, formatted data is passed to a frame buffer for display, with a variety of possible configurations.

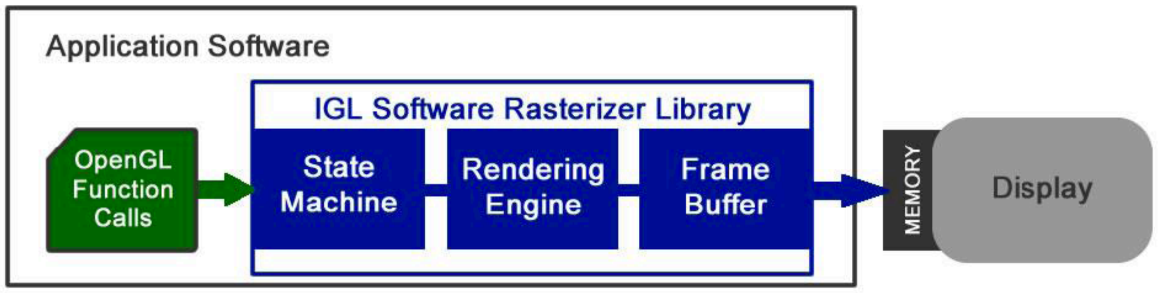


Figure 1 — IGL Software Rasterizer Architecture

OpenGL is fundamentally a state machine that accepts commands and manages variables. IGL brings all the expected capabilities of OpenGL SC to the embedded graphics application programmer, and offers additional flexibility. The IGL State Machine can accept commands directly from an application residing on the same processing core, or remotely from a different processing core generating graphics requests via an optional IGL Platform Driver.

The IGL Rendering Engine implements the library of OpenGL SC functions, computing the required transforms on the graphics pipeline. It applies the defined states and commands to a matrix stack, which encompasses texture, display list, color depth, and frame buffer data, along with local OpenGL data structures. Objects are rendered into a scene, and passed to the Frame Buffer.

The IGL Frame Buffer is a physical block of memory, which can be in local memory space, or PCI or PCI Express memory space. IGL supports display resolutions up to 2047 x 2047 pixels. All IGL needs to know is where the Frame Buffer is located, not the nuances of displaying it. Various third-party hardware implementations are available to take the Frame Buffer memory and display it on HDMI, DisplayPort, DVI, and legacy VGA displays.

Conformance to OpenGL SC and simplicity in design are just the beginning of the IGL story. What is exciting about the idea of a software rasterizer is the ways it addresses problems embedded system designers face with graphics — problems difficult and expensive to solve using other approaches.

6 Boosts for Critical Situations

If the embedded system setting is benign, with needs for high performance graphics or very large resolutions, there are many great hardware solutions that can fill the need. The use of OpenGL brings the realism and cross-platform advantages already discussed.

Where software rasterization starts to outperform alternatives is when the settings become critical, with constraints making the size and cost of specialized hardware a larger factor, and performance is measured more by latency on objects of interest. Here are six ways IGL and software rendering can boost unique solutions, solving challenges common in safety-critical environments.

Boost 1 – Safety-Critical Certification

Many safety-critical applications have to pass a stringent set of criteria, such as DO-178B/C and DO-254 for avionics. Complex solutions are difficult to verify, and simple solutions are preferred by certifying agencies. IGL complies with DO-178B, has been certified in avionics applications, and is a Reusable Software Component with artifacts and traceability required by the standard. There are several advantages to using IGL in a situation calling for certification, but perhaps the largest is having one OpenGL SC platform across devices and over the life of the project.

IGL enables stability, as an API library never goes obsolete. OpenGL SC application code does not have to change to support multiple platforms or different hardware architectures. There are no hardware-specific drivers which must be certified, or re-certified, as hardware changes. This is a big difference compared to a hardware-based strategy, which can involve lifetime buys or re-certification cycles as hardware becomes obsolete, and may have a software impact if drivers supporting newer hardware are different.

The tasks of certification to DO-178B/C can be substantial, and using proven solutions can save a project from increased costs and schedule risks. The optional IGL Certification Kit service delivers a suite of 25 compliance documents that analyze and verify configuration traceability and code integrity. Because every line of code in IGL is known, understood, and proven in safety-critical applications, designers can rely on it as a stable platform, and focus on the details of creating and certifying an OpenGL SC application.

Boost 2 – Radiation Hardening

In benign settings, background radiation doesn’t cause a concern for electronics, but in space and nuclear contexts it is a quite different situation. Space-borne applications, or those designed for some medical and industrial environments, must be able to endure significant radiation, sometimes well beyond levels humans should be exposed to.

Most semiconductor processes, especially small geometry silicon typical of high performance CPUs and GPUs, just don't withstand concentrated doses of radiation. Soft errors can be injected, which corrupt data, or hard failures can occur.

Specialized, radiation-hardened processors or field programmable gate arrays (FPGAs) have the radiation tolerance necessary to perform error-free in these environments. Many of the latest "rad-hard" processors have been developed with processor cores for advanced software applications. Display interfaces are typically based on simpler mixed signal semiconductors from mature process nodes, with larger geometries not as prone to radiation upset.

Once the rad-hard processor is established in a system, IGL functions can run as any other application software would. This means rad-hard processors can now run OpenGL SC as part of their application suite without requiring any processor-specific features, or having to augment the CPU with non-hardened GPU hardware.

Boost 3 – Texturing Engine and Overlays

Since IGL is based on OpenGL SC, programmers familiar with OpenGL programming can be productive immediately. Additionally, a unique feature of a software rasterizer is its ability to coexist with a hardware GPU in a system. Combining these two ideas, partitioning the application so each rendering engine leverages its strengths, creates an immense amount of flexibility in designing embedded system graphics.

One scenario is to use a hardware GPU to create the outline of the scene, using polygons, and use IGL to map textures onto polygons to create effects. By using IGL to create effects, the hardware GPU is offloaded and determinism is preserved for critical objects.

Another way to use IGL is to have it completely manage the symbols of interest — such as tracked targets — on a background managed by a hardware GPU. This can be done with an overlay or video-blending strategy depending on the display implementation.

The objective in all these approaches is to ensure that symbols of interest in the scene are positioned exactly where they are supposed to be, in real-time, at all times, while the background doesn't change as quickly.

Boost 4 – FPGA Acceleration

While IGL is platform independent and does not require specific processor features to run, there is an opportunity to take advantage of hardware to gain performance. The IGL rasterization engine is essentially a number cruncher, performing repeated operations on a data set to transform the graphics pipeline into a scene. Most OpenGL SC applications are implemented with calls to 15 to 20 API functions, meaning the payoff to accelerating an often-used function can be large.

Running IGL in an FPGA with a processor core and configurable logic can provide that acceleration. By analyzing the OpenGL SC functions in use and the underlying math involved, a multiply-accumulate block in the FPGA can be dedicated to create an IGL function co-processor. This can be a very efficient optimization since there is a minimal set of hardware applied, as opposed to tapping a digital signal processor (DSP) or vector processing unit in the same role.

An optional utility for IGL, the Performance Monitoring Graphics Library (PMGL), provides counts and timing of function calls and measurements of frame rates. This can aid in both optimizing OpenGL SC programming and exploring the possibilities for FPGA acceleration.

Boost 5 – Distributed Displays

In a simple embedded system, graphics are displayed locally from data generated on a processor core running the application. In larger systems with multiple processor cores, sometimes the display is not attached to the same core as the graphics rendering engine. A single graphics display might need to merge data from several distributed sources.

With IGL, managing distributed displays becomes easier as the same software can be deployed on nodes as necessary. Graphics nodes do not have to be local to processing nodes in these configurations. Using the optional IGL Platform Driver, commands can be sent using a PCI-like protocol over a bus or a network. This allows another processor core to perform computations and request operations on a display. It also allows a display to be replicated on a remote core, staying in sync. While network and bus latency can be a concern as with any application, this provides flexibility at a lower cost than adding specialized hardware.

Noteworthy here is the idea of dedicating a processor core, either a standalone CPU or a core in a multicore processor, to an application running IGL managing the critical objects in a scene as a graphics offload engine to a hardware GPU or another instance of IGL. This can help not only with graphics performance, but with certification of a system, as graphics operations are partitioned.

Boost 6 – Reducing SWaP

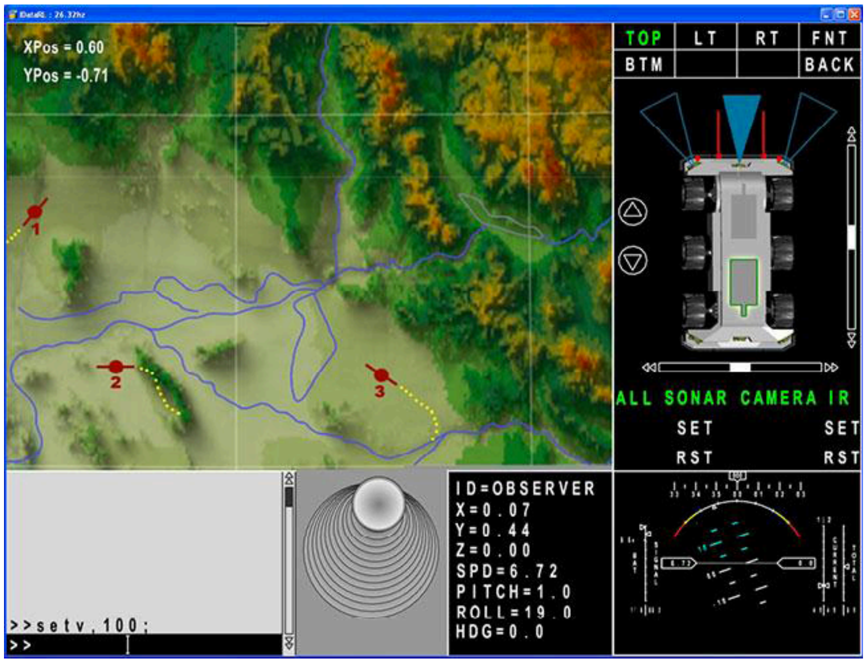
Size, weight, and power (SWaP) are prime considerations in more and more applications. Put mildly, many of today’s high performance GPUs are physically large, power-hungry, and require active cooling. Eliminating a GPU, or more than one, can be a huge SWaP savings in a system, and reduce maintenance and life cycle costs as well.

Most high-performance microprocessors have enough computing power to run OpenGL SC software rendering. IGL has a compact footprint — less than 1MB of memory supports general operation — and the Frame Buffer size depends on the display resolution. Since IGL is a library, it renders graphics when it is called, efficiently, without consuming resources in the background.

If a project is up against a power supply budget, or doesn’t have enough, or the right kind, of cooling available, or just needs to stay within as small a space as possible, IGL gives designers an alternative for embedded systems graphics implementation.

Putting IGL to Work In Your System

With the advantages and openness of rendering OpenGL SC graphics in software, its conformance to safety-critical standards, and flexibility in configuration and use, IGL creates opportunities to solve problems in designing embedded systems graphics solutions.



```
>>setv,100;
>>
```

```
ID=OBSERVER
X=0.07
Y=0.44
Z=0.00
SPD=6.72
PITCH=1.0
ROLL=19.0
HDG=0.0
```

Figure 2 — IGL Software Rasterizer Image Quality

IGL is platform independent. It runs on Power Architecture®, Intel® Architecture, and ARM® architecture processors, and targets popular operating systems for safety-critical environments such as DEOS™, INTEGRITY®, Linux®, PikeOS™, and VxWorks®, and an evaluation version is available for Windows®. Depending on the configuration, it is delivered in a shared (.so or .dll) or linkable (.a or .lib) library. Applications calling IGL can be compiled using a C compiler and standard C libraries; safety-critical applications for certification require using a certified C compiler.

More information, including a user manual and evaluation tools, is available online at

www.ensco.com/avionics-products/igl

To learn more about IGL licensing, and services including evaluation, porting, and certification, contact Ray Niacaris at +1-909-593-2055, or sales@idatavs.com.

ENSCO Avionics, Inc. Headquarters	General	ensco.com/avionics
3110 Fairview Park Drive, Suite 300	Technical Support	idatavs.supportportal.com
Falls Church, VA 22042-4501 USA		+1-877-825-4890
		support@idatavs.com