Compositing and the Reuse of Software in Safety Critical Graphics Applications

Introduction

Avionics and automotive hardware is ever-evolving, but often the accompanying display software doesn't need to change. CoreAVI has a long history of collaboration with the Future Airborne Capability Environment (FACE) Consortium to lead the way in the creation of truly portable display software. Open standards including OpenGL®, FACE™, and Vulkan® allow a clear separation between display software and the hardware on which it runs. This separation benefits software developers as it allows them to start development work without needing to know what GPU they will be writing code for.

CoreAVI has introduced a way to use the GPU to composite the visual output of different pieces of display software without requiring them to be rewritten each time a new symbol is added to the display. This method of compositing maintains the transparency of each pixel and sends them to the compositing application for proper blending, which enables full screen overlays to be run as separate applications as well as the splitting up the screen's geometry. Such composition allows all but the compositing application to remain unchanged as new technologies are added or parts of the display rearranged. This white paper details how existing safety critical DO-178C or ISO 26262 application software source code can effectively be rehosted on advancing hardware.

What is the EGL Compositor?

The FACE Consortium published the EGL_EXT_compositor extension with the Khronos Group on February 3, 2017 to add compositing capability to EGL. This extension minimizes application effort and allows for the composition of multiple windows within a multi-partition EGL system. The extension allows for the following capabilities:

- Creation of a primary EGLContext and window for each display
- Creation of additional windows using non-displayable surfaces
- Composition of all non-displayable windows to a single display through the primary EGLContext.

There is one primary context created for the display while other EGLContexts are referred to as secondary contexts. The EGL_EXT_compositor is beneficial as it allows for off-screen asynchronous updates and keeps information secure by preventing any non-primary contexts and surfaces from rendering to the display. It allows (but does not specify the need for) management and control of GPU allocation to specific contexts including how much GPU memory a specific application can use. Each application draws into its own off-screen window, providing a level of security that ensures one application will not be overwritten with other applications' video data, or provide back-channel access though a shared framebuffer. The compositing application controls visibility of applications and prevents one application from drawing over another application unless the system designer allows it.

This extension ensures that OpenGL graphics cannot be rendered to the display without express instructions from the compositing application, nor can they interfere with any other rendering contexts.

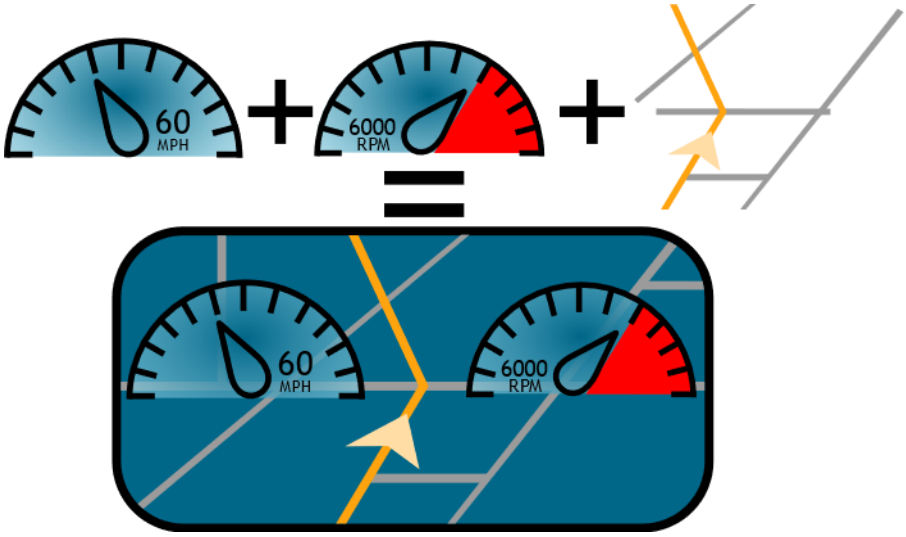

Figure 1: The Compositor allows multiple applications to be combined into one display

The above figure illustrates an example of compositing through a car dashboard. In this example the symbology of the dashboard is separated into three applications. There is an application to create the speedometer symbology, an application to create the tachometer symbology, and an application to create the GPS symbology. The result of each individual application is combined into a final display that shows all the three applications. Each of these applications could be written using different graphics APIs and safety certified separately.

How can existing software work with a Compositor?

For existing legacy applications that use safety critical open standards such as OpenGL SC 1, OpenGL SC 2, and EGL, there would be minimal application changes required to move to a compositor setup using the EGL_EXT_compositor extension. For an application that renders offscreen, it can simply create an EGL context in the same manner as it would without the compositor. Existing SC1 and SC2 applications are likely using a driver specific API to create their native window handles. If using the compositor, an EGL compositor specific native window handle would be required. This is beneficial as the application is driver independent. The off-screen rendering application can then continue to function thinking that it is the only graphics application that is running on the system.

Conversely, we can demonstrate an example where a compositor application is added to the system. It will initialize all the offscreen buffers to ensure the offscreen applications can initialize properly. It will also be responsible for choosing how to render each of the offscreen buffers to the final display. The compositor application has final authority over what is physically displayed on the screen and how it is displayed on the screen. It can overlay applications with alpha blending or dedicate certain regions of the screen to a single application.

Modern applications that are leveraging the new graphics capabilities of Vulkan SC 1.0 can take advantage of compositor technology through the use of the KHR_Display extension. This extension works similarly to the EGL_EXT_compositor extension and will allow a Vulkan application to either draw to an offscreen buffer or set up offscreen buffers into which other Vulkan SC 1.0 applications can render. Taking advantage of this extension to open a display will allow the Vulkan SC 1.0 application to be used in a compositor setup where there are other Vulkan SC 1 applications or even OpenGL SC 1 and OpenGL SC 2 applications.

By adding an additional application to the system and making small modifications to the initialization process, any application can take advantage of the features offered by the compositor. Through these changes, an application will become more driver independent and be able to interoperate with legacy and modern applications.

How can an application take advantage of new technology?

One way to illustrate how an old application can take advantage of new technology is in the case of a map application. Suppose we have a map application that was written several years ago as an OpenGL SC 1 application, was subsequently safety certified, and the developers proactively used the EGL compositor extension for surface creation. Now suppose that we have been tasked with updating this map application to have an overlay written against modern graphics standards using modern technology. This modern overlay is an existing technology that has been developed in the new open graphics standard Vulkan SC 1, which is not backwards compatible with OpenGL SC 1. Therefore, the new overlay software cannot directly be added to this existing certified map application. Without using compositing, this scenario has two options for success. The overlay application can be ported to OpenGL SC 1, or the map application can be ported to Vulkan SC 1.

The idea of porting the application presents significant downsides. First, porting the more modern overlay to OpenGL SC 1 means the potential loss of modern features and performance improvements gained by using the more modern Vulkan SC 1 API. This overlay may depend on features such as compute shaders, instanced rendering, and the programmable graphics pipeline, which are not available in OpenGL SC 1. Without some of these features, it could be impossible to port some of the application features from Vulkan SC 1 to OpenGL SC 1. Likewise, porting the map application from OpenGL SC 1 to Vulkan SC 1 could be a momentous task. To achieve high performance in OpenGL SC 1, the map application likely took advantage of legacy features such as display lists. These features do not exist in Vulkan SC 1 and are no longer the recommended method to render graphics. Overall, for this case of porting an application, we face substantial challenges.

Through the use of compositor technology, we can sidestep these issues by simply allowing the legacy application to run as an OpenGL SC 1 application and having the modern overlay run as a Vulkan SC 1 application. We can then have a third compositor application that combines the output of the two applications to a final image that is then shown on the display. This provides two significant advantages. First and foremost, none of the applications need to be rewritten to use a different graphics API. This means that the existing application can be reused as much as possible while introducing new functionality. The second advantage is that all the existing infrastructure, such as tests, documentation, and artifacts for the legacy application, can be reused since the legacy application remains unchanged.

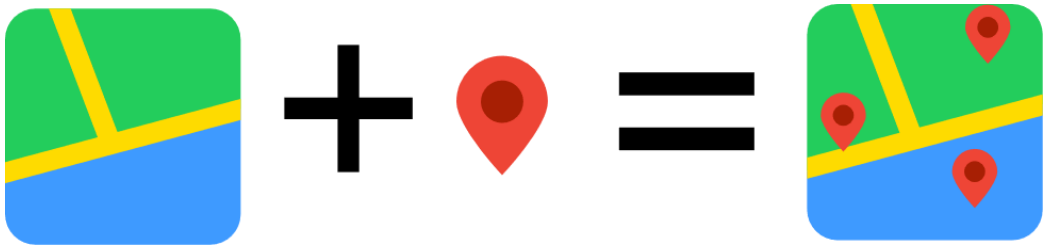

Figure 2: Legacy map application with the addition of modern widgets gives a whole new application

The compositor provides a clear benefit by allowing new symbology to leverage modern technology on legacy applications, with minimal or no additional cost. The compositor takes advantage of open standards such as EGL's EGL_EXT_compositor or Vulkan's KHR_Display, allowing applications to be written without knowing the final target hardware.

What do we gain with compositing?

Although the above example is basic, it is clear that adopting compositor technology provides three primary benefits. First, since the compositor is an open standard, an application can be written without expecting to target certain graphics hardware or software. This means application development can be done early and tested robustly. The next major advantage is the ability to combine symbology written against different graphics APIs into a single final application. The third advantage is the ability to reuse old applications, and the artifacts associated with them, in new systems without having to change the application and invalidate the artifacts. Overall, the correct use of the compositor system can significantly reduce development time through the reuse of existing safety critical applications.

Contact Sales@CoreAVI.com for more information how the EGL_EXT_compositor extension can benefit your safety critical applications.

Author

Lucas Fryzek

Field Application Engineer

Lucas Fryzek has been a software developer with CoreAVI for 4 years and has recently transitioned over to the role of Field Application Engineer (FAE). Lucas has extensive hands-on experience in developing embedded software systems deployable in DO-178C safety critical environments and has worked in tandem with many of CoreAVI's largest leading customers to deliver safety critical solutions, including those specializing in the division of graphics workloads across multiple CPU cores. Lucas' experience with OpenGL SC and Vulkan APIs allow him to provide expert technical support and guidance both to CoreAVI's customers and internal team.