OpenGL SC Implementation on the OpenGL Hardware

Nakhoon BAEK†1, Member and Hwanyong LEE††, Nonmember

SUMMARY  The need for the OpenGL-family of the 3D rendering API’s are highly increasing, especially for graphical human-machine interfaces on various systems. In the case of safety-critical market for avionics, military, medical and automotive applications, OpenGL SC, the safety critical profile of the OpenGL standard plays the major role for graphical interfaces. In this paper, we present an efficient way of implementing OpenGL SC 3D graphics API for the environments with hardware-supported OpenGL 1.1 and its multi-texture extension facility, which is widely available on recent embedded systems. Our approach achieved the OpenGL SC features at the low development cost on the embedded systems and also on general personal computers. Our final result shows its compliance with the OpenGL SC standard specification. From the efficiency point of view, we measured its execution times for various application programs, to show a remarkable speed-up.

key words: OpenGL SC, OpenGL safety-critical, 3D graphics library, de facto standard, cost-effective implementation

1. Introduction

OpenGL SC (Safety Critical profile) [1] is derived from OpenGL, the mostly widely used 3D graphics API, to meet the unique requirements of the safety-critical systems. The need for this 3D graphics API is rapidly increasing with the growth of safety-critical market. The current 1.0.1 version of OpenGL SC targets various application areas including but not limited to the followings [2]:

avionics applications: The Federal Aviation Administration (FAA) mandated DO-178B certification process for software airplane cockpits demands 100% reliable graphics drivers for instrumentation, navigation and controls. Currently, OpenGL SC is the only 3D graphics API compliant with the DO-178B standard.

automotive applications: Integrated dashboard applications will need OpenGL SC safety-critical reliability.

military applications: Primarily avionics. But increasingly embedded training and visualization on handheld devices.

medical applications: Real-time display of medical data requires 100% reliability especially for surgery.

industrial applications: Equipment for power plant instrumentation, transportation monitoring, networking, surveillance, and others will be updated with the graphics API that meets safety-critical certification.

Thus, at this time, we naturally need a cost-effective way of OpenGL SC implementation especially based on COTS (commercial off-the-shelf) items. In this paper, we focus on the large amount of OpenGL 1.x pipelines available on nowadays embedded devices, and aimed at the OpenGL SC emulation over them.

We have few cases of OpenGL SC implementations. Some of them provide fully-dedicated OpenGL SC semiconductor chips or exclusively new device drivers on existing OpenGL chips [3], [4]. These solutions require the full development costs for silicon chips and device drivers. Although a few software solutions are also available, their execution speeds are not so promising [5].

Implementation of a graphics library over another existing graphics pipeline shows advantages such as cost-effectiveness and high portability. In the case of OpenGL ES standard [6], [7], we have an example of OpenGL ES 1.1 implementation over OpenGL ES 2.0, where their ES 2.0 pipeline was modified to fully support ES 1.1 emulations [8]. An OpenGL ES emulation over the desktop Open GL is also available [9].

To the best of our knowledge, we failed to find any previous literature communication dealing with the implementation of OpenGL SC over OpenGL pipeline. We demonstrate we can emulate the whole features of the OpenGL SC standard over the OpenGL 1.1 pipeline with ARB_multitexture extension, which may be one of the lowest hardware profiles for embedded 3D graphics systems. Additionally, our OpenGL SC emulation can be used for the PC-based OpenGL SC application development environment.

2. Design and Implementation

The OpenGL SC specification provides a total of 101 API functions, based on the OpenGL 1.3 specification [10]. These API functions can be classified into the core API functions and several extensions: a core extension of OES_single_precision, a mandatory extension of EXT_painted_textures and an optional extension of EXT_shared_texture_palette. The last two texture-related extensions are critical to most avionics 2D mapping applications. They separate color table from the texture data to allow rapid color table change and permit palettes to be shared between multiple textures. Our implementation supports all the available extensions of the OpenGL SC. Figure 1 shows the overall block diagram of the OpenGL SC rendering pipeline to support all the features.
Although most of OpenGL SC core API functions can be supported by OpenGL 1.1 core, some texturing functions to realize multiple texture units are only supported by OpenGL 1.3 core. These texturing functions are not included in OpenGL 1.1 or 1.2 core, while these functions are available in the ARB_multitexture extension of OpenGL, even with those lower version cores. Thus, we targeted OpenGL 1.1 core systems equipped with the ARB_multitexture extension, and successfully provided the OpenGL SC emulations on them. Notice that OpenGL SC 1.0.1 specification is based on the OpenGL 1.3 core, while we found that we can accomplish all the OpenGL SC features on OpenGL 1.1 core with the multi-texture extension.

Although the same function names are defined in both of OpenGL SC and its ancestor OpenGL specifications, they do not provide the same functionality. Customized to the safety-critical devices, the features and acceptable parameter values for the OpenGL SC functions became different from the original OpenGL specifications. Thus, to fulfill OpenGL SC requirements, we do perform strict error checking and proper numerical conversions prior to the OpenGL hardware execution. These extra works were performed in a case-by-case manner, as similar to those of the same category implementation of OpenGL ES 1.1 over the desktop OpenGL [9].

Our implementation strategies and their details for the whole OpenGL SC features can be summarized as follows:

**Core functions from OpenGL 1.1:** These functions are expected to be provided by the underlying OpenGL hardware pipeline. However, as already mentioned, they are not the exactly same functions to that specified in OpenGL 1.1 specification. Most of them have changes in their acceptable parameter values and need extra and more specific error handling. Some of them additionally require numerical conversions before calling the underlying OpenGL functions. Thus, they should be implemented in a case-by-case manner.

**Core functions from OpenGL 1.3:** OpenGL SC was originally based on the OpenGL 1.3 specification. Excluding OpenGL 1.1 core functions, the remaining pure OpenGL 1.3 core functions are all related to the multi-texture, which naturally requires more than two hardware texture units. After carefully analyzing the detailed requirements of those functions, we concluded that only the ARB_multitexture extension is required rather than the whole OpenGL 1.3 core features. Thus, we tried to implement OpenGL SC on the OpenGL 1.1 core with multi-texture extensions and finally succeeded.

**OES_single_precision extension:** This single precision extension is a mandatory extension, which must be supported by any OpenGL SC implementation. Fortunately, these functions are single precision floating point type variations of the original double precision floating point-based API functions in the original OpenGL specification. Thus, from the viewpoint of OpenGL SC implementers, these functions can be accomplished to convert the user-provided single precision floating point values into double precision floating point values, and then, calling the underlying OpenGL functions with extra error checks.

**EXT_paletted_texture extension:** This extension is also one of the mandatory extensions in the OpenGL SC specification. It is highly required to support the legacy avionics applications, while currently available OpenGL-related devices do not support it any more. With this extension, we can define an index-based texture and its corresponding color table (or color palette). Currently, most graphics devices use the direct color system and do not support this indexed color features. To fully support this extension, we introduced a new texture processing pipeline, with our own software implementations.

**EXT_shared_texture_palette extension:** It is an optional extension to the OpenGL SC standard, and can be applied only when the above EXT_paletted_texture extension is already supported. Thus, it met the same problem. Our newly designed texture processing pipeline also supports this optional extension.

For the EXT_paletted_texture and EXT_shared_texture_palette extensions, we strongly needed to rearrange the overall texture pipeline. One of the most suitable ways to support the palette texture extensions would be fully utilizing a hardware device with paletted texture features. However, we failed to find any commercial off-the-shelf graphics cards supporting hardware-accelerated paletted textures. The EXT_paletted_texture and EXT_shared_texture_palette extensions are naturally dropped off from the desktop graphics hardware, while OpenGL SC still requires it, mainly due to the backward compatibility. In our implementation, we added a remarkable amount of software codes to support these features. For each texture unit, we need a dedicated color table for the paletted texture extension. An extra color table in the global context is additionally required for the shared texture palette extension.

When defining a texture with the internal format of RGBA, each pixel in the texture is stored in a 4-byte quadruple color value. These quadruple color values are directly
used as the texture source colors, as specified in the typical texture processing pipelines. In the case of palette textures, pixels are expressed as 1-byte color index values. Later, our texture processing pipeline uses these indices to pick up the actual quadruple color values from the color tables in the texture unit or the global context, according to the user-specified flag values. Conceptually, these color restoration procedures would be repeatedly performed whenever we need those textures. In actual implementation, we naturally introduce a texture cache for each texture unit. Thus, the system may repeatedly use the corresponding cached texture, instead of the original paletted texture. Either when the user provides a new texture or when updates the corresponding color table, the cached texture would be discarded, and we perform a new restoration process.

3. Implementation Results

Our first stage implementation was done on a Linux-based system with the hardware-accelerated OpenGL device driver. All the features of OpenGL SC are implemented over the OpenGL 1.1 and multi-texture API functions and some software emulation codes are added. Most of the optimization and debugging works are performed on this Linux-based implementation. The OpenGL SC conformance test suite from Khronos group was used to verify the correctness of our implementation, and it passed all of their tests.

In the second stage, we emigrated to typical low-powered embedded systems, which are equipped with OpenGL 1.2-based graphics chips with the multi-texture extension. We verified the execution of all the OpenGL SC test applications on these systems, as shown in Fig. 2.

Table 1 shows the overall cost of our OpenGL SC emulation library. We first executed original OpenGL sample programs and then converted them into OpenGL SC programs, and then compared their performance. Figure 3 represents screen shots from those sample programs. Note that the OpenGL SC programs cannot use OpenGL-specific GL_QUAD or GL_POLYGON primitives, and additionally perform extra software emulations for the paletted textures. In spite of these handicaps, our implementation shows less than 2% delays.

4. Conclusion

We showed that all the core and extension features of OpenGL SC are achieved as an emulation layer over OpenGL 1.1 hardware equipped with multi-texture extension. The results demonstrate the effectiveness of our approach. Our implementation is able to run various OpenGL SC applications and conformance tests correctly, with less than 2% time delays. Our next step is implementation of OpenGL SC over lower-powered chips such as multimedia processors or DSP chips.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (Grant 2011-0014886).

References

[1] B. Stockwell, OpenGL SC: Safety-Critical Profile Specification, version 1.0.1 (difference specification), Khronos Group, 2009.
[2] Khronos Group, http://www.khronos.org/ogles/sc/
[3] Alt Software Inc., http://altsoftware.com/
[4] Presagis USA Inc., Seawind-178, http://www.presagis.com/
[5] Quantum 3D Inc., IGL 178, http://www.quantum3d.com/
[6] A. Munshi and J. Leech, OpenGL ES common/common-lite profile specification, version 1.1.12, Full Specification, Khronos Group, 2008.
[7] A. Munshi and J. Leech, OpenGL ES Common Profile Specification, version 2.0.24 (full specification), Khronos Group, 2009.
[8] S. Hill, M. Robart, and E. Tanguy, “Implementing OpenGL ES 1.1 over OpenGL ES 2.0,” Digest of Technical Papers, IEEE International Conference on Consumer Electronics, pp.1–2, 2008.
[9] N. Baek and H. Lee, “OpenGL ES 1.1 implementation based on OpenGL,” Multimedia Tools and Applications, (on-line published).
[10] J. Leech, The OpenGL Graphics System: A Specification, version 1.3, OpenGL ARB, 2001.