OpenGl ES 1.1 software implementation on mobile phones

Hwanyong Lee¹, ², Nakhoon Baek¹, ³a), and James K. Hahn⁴
¹ School of Computer Sci. and Eng., Kyungpook Nat’l Univ., Daegu, Korea
² Huone Inc., Daegu, Korea
³ Mobile Graphics Inc., Daegu, Korea
⁴ Dept. of Computer Sci., The George Washington Univ., United States
a) Corresponding author. E-mail: oceancru@gmail.com

Abstract: OpenGL ES 1.1 is a de facto standard for the 3D graphics API on embedded systems and handheld devices including mobile phones. We present design process and implementation results of our software OpenGL ES 1.1 product. Since the standard document only specifies the API functions and their external actions, the implementer should design all the details of the internal 3D graphics pipeline and exhaustively optimize them. To clearly express the internal pipeline and to explicitly represent related state variables, we introduce some enhancements to UML activity diagrams. Based on these enhanced diagrams, we accomplished the initial draft design and iterative optimizations of the internal pipeline. During the implementation stage, starting from our previous wrapper implementation, we used an iterative block-by-block implementation scheme with immediate verifications. Finally, we achieved a full software implementation of OpenGL ES 1.1, which passes all the official conformance test suites and also satisfies all the requirements in the standard specification. This product is now ready for commercial services.

Keywords: OpenGL ES, 3D graphics library, de facto standard
Classification: Electron devices, circuits, and systems

References

1 Introduction

OpenGL ES [1] is a de facto 3D graphics API (application programming interface) standard from Khronos Group [2]. It is an improved version of OpenGL, dedicated to Embedded Systems (ES), with newly introduced features such as the fixed-point numeric type. Many commercial products including Apple iPhone, Sony PlayStation3, Google Android, and others are adopting it as their standard 3D graphics library.

We have consistent and strong needs for the software implementation of OpenGL ES on handheld devices, due to their cost-effectiveness and low power consumption, even though full hardware solutions are available for a few high-end mobile phones. Since nowadays embedded system CPU’s become more powerful, software implementations show acceptable performances [3].

Currently, two different versions of OpenGL ES are available: ES 1.1 [1] for the fixed function pipeline and ES 2.0 [4] for the programmable hardware. OpenGL ES 1.1 supports both of the common profile and the common-lite profile, which are mainly designed for floating-point and fixed-point processing hardware, respectively. In the case of OpenGL ES 2.0, only the common profile is supported. At this time, OpenGL ES 1.1 is more suitable for mid-tier handheld devices, mainly due to their limited computing powers.

We aimed at providing a software 3D graphics library for mid-tier mobile phones. In this paper, we present design and implementation process of our OpenGL ES 1.1 full software implementation. The Khronos group, who maintains OpenGL ES and its related standards, also provides their conformance test suites (CTS) [5]. A strictly implemented OpenGL ES system should pass all these conformance tests and this is one of the most important factors to show its conformity to the standard. Our implementation passed all the conformance tests and satisfied all the requirements in the standard specification. At least to our knowledge, this paper is the first literature communication dealing with a software implementation of OpenGL ES.

2 Design and implementation process

OpenGL ES 1.1 consists of more than 200 API functions and more than 300 state variables. At the very first stage of our design process, we went through the difficulty of efficiently representing not only the mainstream 3D graphics pipeline but also its internal data objects and processing algorithms. OpenGL ES basically works as a state machine, while we also need to present
various low-level graphics algorithms for its implementation. We tried lots of presentation schemes, only to find some draw-backs for each of them. Instead, we tried to introduce a set of new features into the traditional UML activity diagram. As shown in Figure 1 (a), the 3D graphics pipeline described in the standard specification is first expressed in the usual UML diagrams [6]. Then, the logical flows and important mathematical expressions (in green background) are attached to enhance those UML diagrams, while related state variables are listed in the separately arranged area (in red color), as shown in Figure 1 (b). Their correspondences and data flows are also explicitly denoted (in blue color). Using this enhanced representation scheme, we can successfully express all the reference relationships between the state variables during the overall optimization of underlying graphics algorithms. Several graphics teams including ours currently use this notational scheme as a graphics algorithm optimization tool.

Figure 1 (b) is the final optimized diagram for the normal vector transformation sequences in the per-vertex operation. Notice that the order of processing is rearranged for more efficiency. Through exhaustively optimizing these diagrams, we achieved lots of technical improvements in the various stages of the 3D graphics pipeline, especially for light-and-material processing, rasterization and texture mapping stages.

Equipped with the optimized activity diagrams of the whole 3D graphics pipeline, we started their implementation to meet another difficulty in the reliability acquisition. In our case, we have an OpenGL ES wrapper implementation [7, 8], and it already passed the OpenGL ES CTS, to guarantee its strictly correct graphics outputs. As shown in Figure 2 (a), this wrapper
implementation can be regarded as an OpenGL ES implementation based on the desktop OpenGL renderer. From the reliability point of view, we found that it is more cost-effective to take a block-by-block substitution method starting from this wrapper implementation.

We implement each step of the 3D graphics pipeline as a small software block, and then its corresponding part in the existing wrapper implementation is substituted with it. A series of the OpenGL ES 1.1 conformance test is immediately followed. Thus, during the overall implementation period, we always have an executable and CTS-verified version. Use of the underlying OpenGL hardware is steadily substituted by our full software implementation in a step-by-step manner, as shown in Figure 2 (b). Based on this implementation strategy, we secured the conformability of each block. Additionally, due to its block-by-block substitution, we can adaptively support various hardware features. For example, some mobile phones are equipped with multimedia processors to provide hardware matrix multiplication features, in which case our full software implementation would be substituted by those hardware features, to get better performance. We finally succeeded to achieve the full software implementation of OpenGL ES 1.1, as shown in Figure 2 (c).

Our implementation process can be regarded as a kind of iterative and incremental development [9]. The conformance test suite plays the role of test programs, and the early wrapper implementation acts as the initial implementation. Figure 3 shows selected test and benchmark programs used for checking the performance of our implementation.
3 Implementation results

As presented in Section 2, we always have an executable version of our implementation which passed OpenGL ES 1.1 CTS, during all the implementation period. The OpenGL ES 1.1 CTS consists of many OpenGL ES application programs. Their execution results are investigated with respect to the correct answers derived from the standard specification, to finally conform whether the newly implemented OpenGL ES system goes strictly by the standard specification.

Our implementation supports both of OpenGL ES 1.1 common and common-lite profiles, and passed both of the corresponding conformance test suites. Additional OpenGL ES benchmark programs [10] are also successfully executed, as shown in Figure 3.

On a typical mid-tier system with Intel PXA 270 520 MHz processor, our test programs and benchmark programs show at least 6 to at most 45 frames per second, depending on the complexity of the graphics output. This frame rate is sufficient for most of the user experiences on the mid-tier mobile phones.
4 Conclusion

We presented the design and implementation process of our OpenGL ES 1.1 product. Our implementation fulfilled all the OpenGL ES conformance test suites and the whole requirement in the standard specification.

At the design optimization stage, we need to handle the 3D graphics pipeline with a mass of state variables. We adopted an enhanced notation scheme of UML-based activity diagrams with explicit presentation of related state variables. Through performing local and overall optimizations over these diagrams, we successfully used this notation scheme as a graphics pipeline optimization tool. During its implementation stage, we introduced an iterative block-by-block implementation process with immediate verification of conformance tests, and achieved a highly-reliable and hardware-adaptable system. This strategy would be also effective for the implementation of other graphics API's.

Our implementation is commercially available and would contribute to the wide spreading of 3D graphics facilities on various handheld devices in nowadays consumer markets. We are preparing other graphics solutions including OpenGL ES 2.0 and OpenGL SC (safety-critical profile).

Acknowledgments

Most of this work was initiated during a stay of the second author at the Department of Computer Science, George Washington University. The second author thanks this institution for its hospitality.

This work was partially supported by the National Institutes of Health with a grant R01-DC007125-01B1 to develop computer-based tools for medialization laryngoplasty.

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 2009-0088544).