모바일 3차원 그래픽 아키텍쳐를 위한 시뮬레이션 프레임웍

이원종° 박정수 한탁돈
연세대학교 컴퓨터과학과 미디어시스템연구실
airtight°@yonsei.ac.kr
jspark@yonsei.ac.kr
hantack@kurene.yonsei.ac.kr

KISS Korea Computer Congress 2006

A Simulation Framework for Mobile 3D Graphics Architecture

Won-Jong Lee° Jeong-Soo Park Tack-Don Han
Media System Laboratory, Department of Computer Science, Yonsei University

요 약

In this paper we describe a simulation and development framework for designing mobile 3D graphics architectures. We are developing a simple and flexible simulation and verification environment (SVE) that uses glTrace's ability to intercept and redirect an OpenGL|ES streams. In combination with glTrace to trace OpenGL|ES commands, the SVE simulates the behavior of mobile 3D graphics pipeline during playback of traces, and then produces the second geometry trace that can be used as a test vector for the Verilog/HDL RT-level model. By comparing the frame-by-frame results, we can conduct architectural verification. To demonstrate the functionality of the SVE, we show the implementation of the verified mobile 3D architecture on a FPGA board. For this, we also present an application development environment (ADE) includes a mobile graphics API and a device driver interface (DDI). The proposed two software environments, the SVE and the ADE could be used for developing and testing mobile applications, architectural study and speculative hardware designs.

1. Introduction

Mobile devices such as hand-held phone, smart phone, digital multimedia broadcasting (DMB) terminal, PDA, and portable gaming console are being used all over the world. The market has accepted mobile devices as multi-functional convergence devices that will take the place of many traditional, portable, consumer electronic devices, such as cameras and music players. Increased interest in mobile graphics can be seen in the activities of ongoing standard APIs, such as OpenGL|ES (for embedded systems) [1] and M3G (Java Specification Request 184) [2]. Recently, various hardware manufacturers are releasing graphics processing units (GPUs) for mobile devices. With the advent of system-on-chip (SOC) design paradigm for embedded system and advanced ASIC technologies, many chipsets or IP cores are under development [3][4][5][6][7].

Designing mobile 3D graphics hardware requires an efficient software environment. Software simulation environment could support easy modification and fast testing the architectures before designing the cycle accurate RT-models. Moreover, the simulation environment could be used for functional level verification after designing the RT-models. However, the simulation of graphics architecture presents some unique challenges compared to general-purpose computer system [8]. During the time it takes to build a complex simulation infrastructure, the simulated architecture can easily become obsolete due to the rapid advances in GPU architectures. The architecture of GPUs is largely secret; vendors in the highly competitive PC graphics arena are reluctant to release architectural details. Moreover, the architecture of GPUs is moving towards the general purpose for supporting programmability. These features are inherited to the mobile 3D hardware and affect standard mobile graphics APIs such as OpenGL|ES. Consequently, a flexible simulation framework to support various architectural studies is indispensable for designing the mobile 3D hardware.

In this paper, we present a simulation and verification environment (SVE) for designing mobile 3D graphics hardware. The SVE consists of three components - benchmark applications, a publicly available tracing tool (*glTrace* [9]), functional simulator (we call this *simDavid*) models generic OpenGL|ES pipeline. In combination with glTrace to trace OpenGL|ES commands, the simDavid simulates the behavior of mobile 3D graphics pipeline during playback of OpenGL|ES call traces, and then generates the other geometry traces. The resulting traces can be fed into the Verilog/HDL RT-level model - *emDavid* [10]. Finally, the simDavid can simulate a rendered scene frame by frame that can be used for the verification of the architecture by comparing the results of RT-level model. To demonstrate the functionality of the SVE, we show the implementation of the verified mobile 3D architecture on a FPGA board. For this, we also present an application development environment (ADE) that includes a mobile graphics API compatible with OpenGL|ES(ver.1.1) and a device driver interface (DDI). The proposed two software environments, the SVE and the ADE could be used for developing and testing mobile applications, architectural study and hardware design.

The rest of this paper is organized in five sections. Section 2 describes the simulation and verification environment. Section 3 explains our implementation of related API and device driver. Section 4 shows the demonstration of our software environment on FPGA boards. Section 5 concludes the paper and discusses future work.

2. Simulation and Verification Environment

The overall simulation environment and data flow is shown in Fig. 1. The simulator is driven using a trace of OpenGL|ES graphics commands instrumented with additional information that describes the behavior of those commands and the contents of memory. This trace is fed into our functional simulation model -simDavid that simulates the flow of data and computation through each stage in the decoupled architecture of OpenGL|ES pipeline. During execution, we generate the geometry traces and feed them into the Verilog/HDL simulator for the cycle-timer simulation of rasterizer operations. Finally, the simDavid outputs the rendered frame images that are used for the verification of a given architecture by comparing with the results from the Verilog/HDL model. In this section, we describe the structure of our simulation and verification environment.

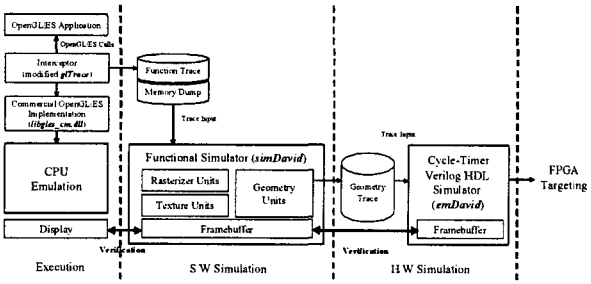

Fig. 1 Overview of the proposed simulation & verification environment

3.1 Capturing the traces of OpenGL|ES commands

Driving our simulation environment is a trace of graphics commands. This is similar to the OpenGL tracing approach described by Sheaffer [8]. We utilize the glTrace [9] to capture the original graphics commands trace. The glTrace is a tool for manipulating streams of traditional OpenGL commands and generates traces and occurrence statistics. The use of glTrace simplifies the task of capturing the behavior of real-world applications, since the glTrace can be applied non-invasively to applications for which source code is not available and can store an application's OpenGL stream to disk for playback and reproducible analysis later. Besides, the glTrace is itself a DLL file to replace the standard OpenGL library (opengl32.dll). As the application makes calls to the OpenGL API, the glTrace leaves the called function log with its parameter lists and then re-call the actual original OpenGL library for execution. Thus, we can use any OpenGL application as a benchmark without any requirement on the availability of source code.

We modified the glTrace so that OpenGL|ES call is supported. As a result, this modified glTrace can replace a commercial OpenGL|ES library implementation (libgles_cm.dll). We used Hybrid Graphics' OpenGL|ES implementation[15] that supports Pocket PC, Nokia Series, BREW, and Windows: x86 compatible PC. The OpenGL|ES applications can be executed with this commercial library on the desktop PC. Therefore, modified glTrace can successfully capture the commands during the OpenGL|ES application's execution.

3.2 Functional simulator description

After gathering OpenGL|ES commands traces and memory dumps, the behavioral level simulation of OpenGL|ES pipeline can be performed in the functional simulator, simDavid. The internal block diagram of simDavid is shown in Fig. 2. The simDavid simulator consists

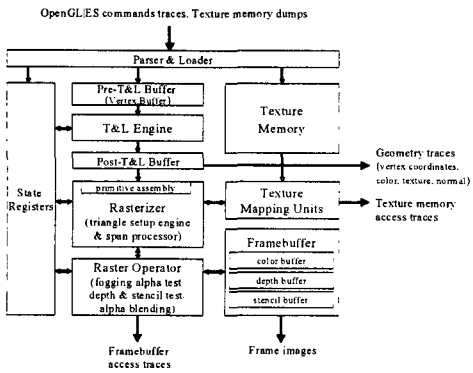

Fig. 2 The internal block diagram of simDavid

of parser/loader, T&L engine, rasterizer, texturing unit, raster operator, and graphics memory. Simulation proceeds in the following way:

* ***Parsing and loading:*** Initially, the OpenGL|ES command traces are parsed line-by-line and classified into function names and parameter lists. The geometry information (vertex coordinates, texture coordinates, and normal) in vertex array is fed into the Pre-T&L buffer. Any uniform and state values such as matrices, lighting parameters, viewport, and texture parameters are stored in state registers. The texture is loaded to the texture memory.

* ***Geometry processing:*** Whenever the Pre-T&L buffer is full or any global states for primitives are changed, simDavid starts the geometry pipeline: transformation, lighting, clip test, texture coordinate generation, and projection. In each stage, needed states values (matrices, lighting parameters) are fetched from state registers. Computed geometry values are stored to the Post-T&L buffer and traced primitive-by-primitive.

* ***Rasterization:*** After geometry processing, each vertex is indexed with regard to the current OpenGL|ES primitive in "primitive assembly". Next, the general rasterization processing (triangle setup and edge work, and span processing) is performed.

* ***Texture mapping & raster operation:*** After rasterization, per-fragment operations are performed. This includes texture mapping, fogging, alpha test, depth and stencil test, alpha blending, and frame buffer operation in a fully pipelined manner. Texture memory and framebuffer accesses are recorded and traced for performance evaluation in the cache system. Finally, rendered frame images are generated.

In summary, the simDavid simulates a fixed-function mobile GPU's behavior. Every functional unit either advances in its computation, possibly producing an output for the next stage and incurring the cost of that computation's operations, or stalls as it waits on a buffer. By analyzing the activities of the various stages, we can study the performance and bottlenecks of the system. The simulation advances relatively quickly by architectural simulation standards. A simulated frame for a medium-complex (QVGA resolution) OpenGL|ES application requires approximately 3-5 seconds (except image and trace generation) on a 2.8GHz Intel Pentium IV with 512Mbytes RAM. This enables rapid exploration of design parameters.

3.3 Verifying the RTL model

The simDavid is used in the architectural verification of emDavid [10]. The emDavid is a rasterizer architecture that supports OpenGL|ES ver 1.0 and designed for mobile 3D graphics that support VGA (640x480) or SVGA (800x600) screen size, 32bit color, depth buffer, Gouraud shading, texture mapping and raster operation. The performance specification calls for 24-30 frames per second with a maximum of 100K vertices per frames and maximum of 4M triangles per second, while running at 100MHz.

The emDavid rasterizer at the RT-level (RTL) is modeled using Verilog/HDL. The simDavid simulator provides the geometry traces as test vectors for emDavid and shows the correct values for comparison with the emDavid simulation results. The test vectors are translated by the Verilog programming language interface (PLI) routine for the Verilog/HDL model. The Verilog/HDL simulator uses the test vectors, executes the simulation, and produces a pixel data dump. To find error bounds, this pixel dump is compared with the results of simDavid. Using the verification scheme. we can conduct architectural studies involving visual simulation and use of a few fixed-point formats for computation in data paths.

4. Application Development Environment

To demonstrate the functionality of the simulation and verification environment, we show the implementation of the verified emDavid on a FPGA board. For this, we also present an application development environment (ADE) that includes a mobile graphics API compatible

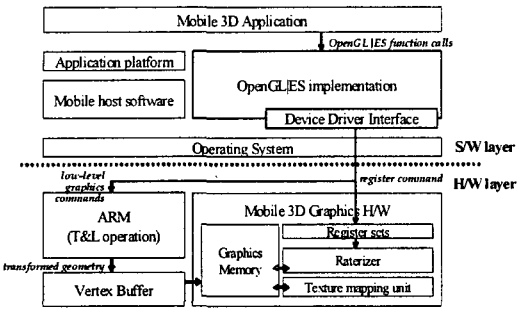

Fig. 3. Application development environment includes mobile graphics API and device driver interface on a FPGA

with OpenGL|ES ver. 1.1 and a device driver interface (DDI). A well-defined graphics API optimized for mobile devices is needed to visualize and accelerate the graphics application on mobile 3D hardware. The OpenGL|ES is a low-level, lightweight API for advanced embedded graphics using well-defined subset profiles of OpenGL. This standard 3D graphics API for embedded systems makes it possible to offer many advanced 3D graphics and games across all major mobile and embedded platforms. Since OpenGL ES is based on OpenGL, no new technologies are needed. This ensures synergy with, and a migration path to, full OpenGL -the most widely adopted cross-platform graphics API. Currently, most of the mobile 3D hardware vendors try to support the full specification of OpenGL|ES.

To support the compatibility of running mobile applications, an OpenGL|ES ver. 1.1 compatible mobile graphics library, is implemented by modifying and repacking the Mesa 3D graphics library [11]. Although the Mesa has a few advantages, it has some limitations for use as a mobile graphics library. Data structures (contexts, states, etc.) that are defined in Mesa are too big and complex. The size of compiled footprint is about 800Kbytes. We simplified the overall structure and optimized the functionalities to support embedded systems. The application development environment is shown in Fig.3. It fills the gap between mobile application and hardware blocks on the application platform of mobile devices.

Device driver interface (DDI) drives the mobile graphics hardware by controlling the register set. Both device driver and graphics hardware controller are monitoring register sets. They are synchronized using handshaking protocol. After geometry processing in ARM, the set of transformed vertices, ready to be rasterized, are stored to 4Kbytes vertex buffer (VB). The VB contains the hardware setting information, vertex coordinates, texture coordinates, color, depth, and others. After vertex buffer is generated, DDI sets control registers that result in copying VB in system memory to graphic memory. Mobile graphics hardware reads current VB line by line, and starts rendering triangle by triangle. If all VBs corresponding to current frame are rendered, DDI sets registers for flushing framebuffer. Then, DDI sets the control registers to let graphics hardware clear and initialize current results and buffer for next frame.

5. Implementation and Demonstration on a FPGA Board

The implementation and demonstration of emDavid on two target FPGA boards using the ADE of Fig. 3 is described in this section. Target FPGA board is a dedicated board for designing digital multimedia broadcasting (DMB) terminal and consists of an ARM 10 processor, two FPGA chips (Xilinx Virtex-2) and peripheral blocks. Its display supports screen resolution of 640x480 or 800x600 and pSOS [17] is used as the operating system. The mobile graphics API, DDI and emDavid of Fig. 3 are ported to target board. We developed a few

(a) Texture mapping (b) Environment mapping

Fig. 4. Screenshots of real-time rendering on target FPGA board ported our ADE

simple OpenGL|ES applications for testing. Fig. 4 show the screenshots of real-time rendering of these applications on FPGA board. On the average, we obtain a rendering performance of 5-10 frame/second at low clock speed (10-20MHz).

6. Conclusion

A simple and flexible simulation and verification environment for developing mobile 3D graphics hardware has been described in this paper. In combination with a publicly available tool-glTrace to trace OpenGL|ES commands, simDavid simulates the behavior of graphics hard-ware during playback of OpenGL|ES call traces, and then generates the other traces-memory access, geometry information, etc. The resulting traces are fed into the VHDL/Verilog RT-level model for performance evaluation. Finally, simDavid can make a rendered scene frame-by-frame that can be used for the verification of RT-level model. For mobile visualization, we also implemented a mobile graphics API compatible with OpenGL|ES ver. 1.1 by repacking the Mesa 3D graphics library and a device driver interface for the graphics renderer-emDavid implemented on FPGA boards.

The proposed application development environment comprising a simulator, an API and a device driver can be used for other applications, architectural study, and hardware design.

References

[1] Khronous Group, OpenGL ES, http://www.khronos.org
[2] Java Community Process, JSR-184, http://www.jcp.org/en/jsr/detail?id=184
[3] GoForce 3D 4500, nVidia, http://www.nvidia.com/page/goforce_3d_4500.html
[4] Imageon 3D, ATI, http://www.ati.com/products/handheld/products.html
[5] G40, Bitboys, http://www.bitboys.com/products.php
[6] Mali, Falanx, http://www.falanx.com/product.html
[7] PowerVR MBX, Imagination Technology, http://www.powervr.com/Products/Graphics/MBX/Index.asp
[8] Sheaffer, J.W., Luebke, D., and Skadron, K.: 'A flexible simulation framework for graphics architectures', Proceedings of SIGGRAPH/ Eurographics Workshop on Graphics Hardware, Aug. 2004, pp. 85-94
[9] glTrace2, Hawk Software, http://www.hawksoft.com/gltrace
[10] Jeong, C.H., Park, W.C., Jeong, J.C., Woo, H.J., Lee, K.W., Lee, W.J., Kim, I.S., Lee, S.G., Kim, J.H., Han, T.D., and Lee, M.K.: 'EmDavid: an embedded 3D graphic accelerator for mobile devices', Proceedings of IEEE Cool Chips, April 2003, pp. 72
[11] Mesa 3D graphics library, http://www.mesa3d.org
[12] Ewins, J.P., Watten, P.L., White, M., McNeill, M.D.J., and Lister, P.F.: 'Codesign of graphics hardware accelerators', Proceedings of SIGGRAPH/ Eurographics Workshop on Graphics Hardware, 1997, pp. 103-110
[13] Lee, I.H., Kim, J.Y., Im, Y.H., Choi, Y., Shin, H., Han, C., Kim, D., Park, H., Seo, Y.I, Chung, K., Yu, C.H., Chun, K., and Kim, L.S.: 'A hardware-like high-level language based environment for 3D graphics architecture exploration', Proceedings of IEEE International Symposium on Circuits and Systems, May 2003, pp.512-515
[14] Crisu, D., Cotofana, S.D., Vassiliadis, S., Liuha, P.: 'GRAAL - a development framework for embedded graphics accelerators," Proceedings of Design, Automation and Test in Europe, Feb. 2004, pp. 1366 - 1367
[15] Rasteroid OpenGL|ES implementation, Hybrid Graphics, http://www.hybrid.fi/main/download/tools.php
[16] DINERO IV cache simulator, http://www.cs.wisc.edu/~markhill/DineroIV
[17] pSOS real-time operating system, Wind River, http://www.windriver.com