ICAT 2004

Real-time Translator from OpenGL to OpenGL ES for Mobile Devices

Zhigeng Pan[1], Bing Tang[1], Jian Yang[2], Shushen Sun[1]
[1]State Key Lab of CAD&CG, Zhejiang University, Hangzhou, China
[2]Centrality Communications Co., LtD, Shanghai, China
{ zgpan, btang, jyang, sss}@cad.zju.edu.cn

Abstract

With the rapid development of mobile devices, advanced 2D/3D graphics capabilities are required for new applications such as video, games. OpenGL ES has been designed to implement a subset of the routines of the well-known OpenGL industrial standard. Many developers dream of reusing the applications already written in OpenGL for mobile devices with `minor` or no modification. However, not all the OpenGL functions are supported by OpenGL ES API. The challenge is to make it possible by using clever substitutions. In this paper, we introduce the work of OpenGL translator, which can convert the wide range of existing OpenGL applications in PC world to OpenGL ES in real time. The workloads collected in our experiences also can help to analyze the major bottleneck of the graphics pipeline, and provide valuable information for mobile devices graphics chip design.

Key words: Mobile graphics; Computer graphics; Workload analysis; OpenGL ES

1. Introduction

In recent years, with the rapid development of mobile phones, the number of users of interactive 3D graphics applications is expected to increase drastically in the future. Several companies have developed low-power 3D graphics accelerators to support 3D graphics in real time.[1] OpenGL ES has been designed to implement a subset of the routines of the well-known OpenGL industrial standard.

Since OpenGL ES is just a subset of OpenGL, not all the functions of OpenGL are supported by OpenGL ES. The applications already written for PC platform can not directly run on the mobile devices without any conversion. One solution is to convert and re-compile the sources codes, which will wastes a lot of human resources for this kind of hand-made translation. Furthermore, an original third-part source code could even not be available for many reasons.

The primary goal of this paper is to provide a translator to convert the OpenGL commands to OpenGL ES or emulate most of them in real time. This approach will allow the developers to reuse the applications already written for PC directly on the mobile phone with binary files only. Users will be able to run most applications created for OpenGL on OpenGL ES easily if many applications run at reasonable speed.

The second goal of this paper is to study the dynamic polygonal 3D graphics workloads characterization. Some statistics such as the average and maximum numbers of triangles, the total required texture memory are collected for evaluating 3D graphics accelerator cards. We hope such statistics could help to analyze the major bottleneck of the graphics pipeline and be used to guide the development of low-power, mobile 3D graphics architecture.

2. Related work

In the past ten years, there have been plenty of trace tools developed to help people to study the graphic pipeline. These traces can gather the command stream traces from real OpenGL applications, allowing the entire graphics portion of the application to be replayed at a later time. GLTrace from Hawksoft is a well-known OpenGL wrapper[2]. It is an OpenGL programming utility that intercepts and logs OpenGL calls made by a running application. GLSim & GLTrace is another famous tracing package for recording and playing back OpenGL traces.[3] D3D9Wrapper is a similar tool for Direct3D calls tracer and debugger.[4]

When OpenGL ES 1.0 specification was released by Khronos group released in 2003, many developers were attracted by the newborn standard. DOGLESS was developed as a generic real-time translation engine, which focuses on translating most OpenGL functions into OpenGL|ES.[5] This translator has been tested on some suite application including the famous porting of Quake called “TomazQuake” and Quake2. But the translator does not capture completely reproducible OpenGL calls, and display list has not been supported. DirectX port is a project to emulate the DirectX API calls thru OpenGL commands and other platform specific commands in order to run DirectX application running on other platform than windows.[6]

There have been some researches made to 3D graphics workload characterization.[7] But most works are focused on PC platform and high-end applications.[8, 9] Mitra and

Chiuch presented the dynamics, polygonal 3D graphics workloads characterization in their research.[10] Most related to our work is the study of Iosif etc.[11] They proposed a set of benchmarks that represents the type of 3D applications that might be run on low-power, mobile systems. However, all their tests are based on OpenGL rather than OpenGL ES.

3. Methodology

In this section, we will describe the whole emulator used to translate an OpenGL application to OpenGL ES at runtime and benchmark sets to study the workloads, including the tracing/recording and software simulation environment.

3.1. Tracing Environment

In order to convert the OpenGL commands to OpenGL ES at runtime and obtain a set of repeatable workloads, we have to trace the existing applications and record all OpenGL calls. A tracing environment should be set up firstly.

Our tracer is based on GLTrace from Hawksoft. It wraps all the standard OpenGL functions. It intercepts and logs OpenGL calls made by a running application, and then calls the OpenGL function invoked by the application. No source code is required, since it redirects calls from OpenGL dynamical library to a specified provider DLL, with API function trace to output debug string or a text file.

Since the trace recorded by GLTrace is a text trace, which is rather slow, we modified it by adding a binary logging mode that significantly improves the performance. GLTrace does not log completely reproducible OpenGL calls such as textures. We modified the GLTrace library so that the OpenGL calls used by the targeted applications are completely reproducible. And we also added the number of vertexes and frames counter in GLTrace. In order to get a set of same and repeatable workloads for an application, a trace player is developed to play back the recorded OpenGL traces.

The work statistics were collected using the OpenGL simulator based on Mesa, which is a public-domain 3D graphics rendering library implementing OpenGL API.[12]

3.2. The translator

After the tracing environment was set up, it is the time to translate the commands from OpenGL to OpenGL ES. But a lot of problems can come from the fact that OpenGL ES does not support all the functions of OpenGL. The challenge is to try to make it possible using clever substitutions.

First, the difference between OpenGL and OpenGL ES should be found out. It can be found in the OpenGL ES specification, which can be freely downloaded from the website.

Second, a virtual OpenGL ES processing pipeline should be built: buffer the OpenGL streams and translate buffered streams (send by the applications) into the commands of the appropriate OpenGL ES API. Many futures of the DOGLESS have been referenced in our translator. If the commands are supported by OpenGL ES, just let it alone. If not, use the equivalent codes to emulate most of them at runtime. We listed the main OpenGL features that are not available in OpenGL ES:

- glBegin/glEnd paradigm;
- Some geometric primitives: Quads, Quad strips, Polygons;
- Complex data types;
- Display list.

For example, the Begin/End paradigm is not supported, and the primitives: QUADS, QUAD_STRIP, and POLYGON are also not supported. We trace and intercept the command streams. If the glBegin command was detected, vertex array will be enabled to substitute the commands. At the same time, the primitive mode was saved to force it draw with triangles.

At last, for the memory of mobile devices was limited, simplification of the application database should be made. Furthermore, such devices currently handle scenes with a low resolution of 320×240 pixels. It does not need so high-resolution textures. Textures were automatically subsampled to reduce the required texture memory. We are planning to develop a tool to simplify the scenes.

3.3. *Benchmark sets*

In a near future, the most popular 3D applications running on mobile platforms would be interactive 3D games.[13] Two game environments were introduced in our benchmark sets. According to the investigation of Iosif,[11] three sets of benchmarks are used: a test of the Viewperf 6.1.2 benchmark, two demos of popular interactive 3D game --Quake3 and Tux Racer, and three VRML 1.0 models. The Viewperf benchmarks are designed for high-resolution output devices, and most of the benchmarks have more than 20,000 triangles per frame. The polygon count of these benchmarks is too high for mobile devices. Some benchmarks are CAD/CAM applications. It is unlikely that such application will be offered on mobile platforms. So we did not select it.

Benchmark sets include:

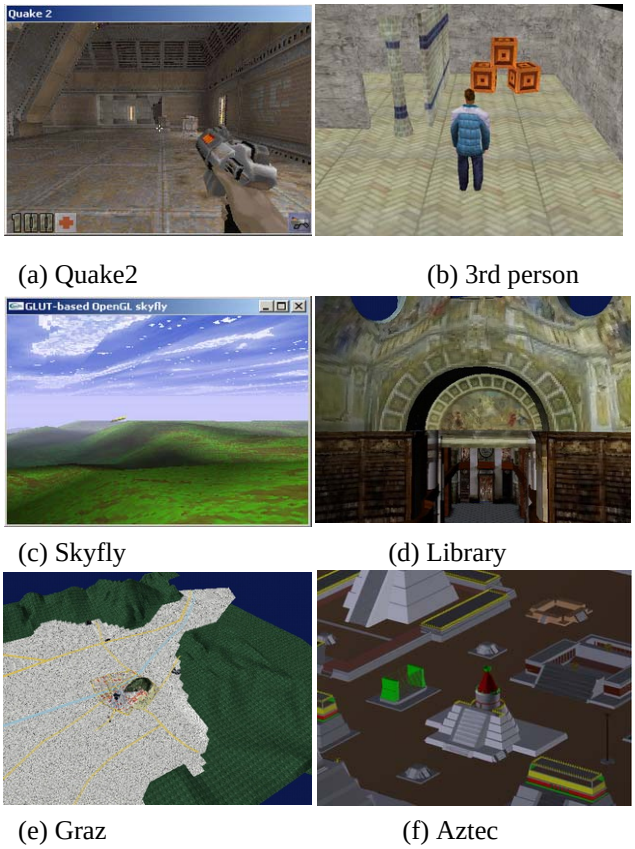
- Quake2[14]. It is a first person shooter game.
- 3rd person. It is a freely available 3D graphics application with game like environment.[15]
- Skyfly. This is GLUT demo selected as our benchmark.

- Library, Graz, and Aztec. These three VRML scenes were chosen based on their diversity and complexity. Library is a virtual model of Austrian National Library and consists of 10292 polygons with detailed textures, Graz is a model of Graz University of Technology, Austria and consists of 8859 polygons,[16] and Aztec is a complex model of Aztec city created by architect Ignacio Marquina.[18]

4. Results and Analysis

4.1. Results of translator

The translator has been tested on the suit applications provided. All the tests have been succeeded in converting the OpenGL commands to OpenGL ES. According to the currently limited resolution display of mobile devices, the output resolution was set as 320× 240 pixels. Textures are subsampled to test visual quality on systems with limited texture memory, and 25 percent of texture resolution is used. The video quality is really near to the original OpenGL version (Fig. 1.).



(a) Quake2 (b) 3rd person

(c) Skyfly (d) Library

(e) Graz (f) Aztec

Fig. 1. Tests of translation

Table 1 shows some statistics of the efficiency during translation. The characteristics and statistics presented in the table are:

- Frames. The total number of frames in each test.
- Total calls. The total number of OpenGL calls in each test.
- Translated calls. The number of OpenGL calls was translated into OpenGL ES in each test.
- Frame rate. The average frames were got per seconds in each test.

Table 1. Statistics of the translator

Applications	Frames	Calls	Translated calls	Translation rate	Frame rate
Quake2	292	5,114,187	1,528,425	29.9%	4
Skyfly	320	2,2869,55	1,042,492	45.6%	15
3rd Person	312	6,264,511	1,7646,72	28.2%	8

We observed that there was a sharp drop of frame rate in these tests when all the trace logs were turned on. It could not reflect actual efficiency of the translator, so the frame rates listed in Table 1 were got with logs toggled off. The Quake 2 demo can get average 4 fps performance in our simulator. These statistics indicate that the translator should be competent for the overall converting work.

4.2. Workload analysis

Table 2 and 3 present some statistics of the workloads. The characteristics and statistics presented in these tables are:

- Image resolution. Since the typical resolution of currently mobile devices is 320×240 pixels, we selected the resolution of 320×240.
- Frames. The total number of frames in each test.
- Texture memory. An indication of the amounts of texture memory required.
- Avg. Triangles. The average number of triangles sent to the rasterizer per frame.
- Max. Triangles per frame. The maximum number of triangles was sent for one frame

Table 2. Characteristics of the benchmarks

Benchmarks	Frames	Image resolution	Texture memory (MB)
Quake2	292	320×240	1.6
Skyfly	320	320×240	0.2
3rd Person	312	320×240	1.2
Library	301	320×240	1.8
Graz	300	320×240	2.2
Aztec	299	320×240	0

As texture mapping becomes pervasive in 3D applications, particularly games, an efficient way is vital important to a mobile device with limited texture memory. Therefore, it is required to list that the amount of texture memory (as presented in Table 2). If the textures are subsampled efficiently, only 25 percent of the texture memory is needed to run these applications

correspondingly. Notice that Quake 2 did not require so much texture memory as we expected. As profiting from the high reuse of texture data, the game can significantly reduce memory bandwidth requirement. However the average amount of texture data that needs to be fetched per frame would be much larger than other applications.

Table 3. Statistics of the benchmarks

Benchmarks	Frames	Avg. Triangles	Max. Triangles
Quake2	292	3,751	6,247
Skyfly	320	1,278	1,597
3rd Person	312	3,982	4,896
Library	301	4,786	14, 024
Graz	300	5,901	9,982
Aztec	299	10, 241	31, 287

Some information can be obtained from Table 3. It can be observed from the column labeled "Max. Triangles per frame" that the scenes generated by Aztec more complex than the others, and it contains more than 30,000 triangles. If we take the average of triangles of Quake 2 as example, the required bus bandwidth is approximately to 0.3 MB per frame (Assuming that each triangle is represented individually and each vertex can potentially have xyz coordinates, each 4 bytes, rgb for color and alpha for transparency, each 1 byte, and texture coordinates uvw, each 4 bytes.) Compared with the other benchmarks, Quake2, skyfly and 3rd person have a more smooth change of triangles. This result is due to a smooth viewpoint change and clipping stage. The full 3D graphics potential would be used in these cases.

5. Conclusion and Future work

As the fast development of mobile devices, low-power 3D graphics accelerators will be available everywhere in near future. And OpenGL ES mobile graphics API has received growing industry-wide acceptance. More and more 3D applications will adopt this newborn OpenGL industrial standard. This paper aimed to reuse the applications already written in OpenGL on mobile platforms. A real-time translator was proposed to convert most of the codes. And some statistics of workloads were collected to analyze the major bottleneck of the graphics pipeline and guide the development of mobile 3D graphics architecture.

As future work, more features will be added into the translator to support more 3D applications on mobile devices. And we intend to extend the number of components for workload analysis.

5. Acknowledge

The research is co-supported by TRAPOYT program of MOE, P.R.C. and the 973 project (grant NO: 2002CB312100). Thanks to Phil Frisbie, Fabio Andreoli and Daniele Paccaloni for the codes on which the translator is based.

References

1. Neil Trevett, "Building the Industry Infrastructure for embedded 3D". *CES 2004 OpenGL ES products & Conformance Testing* (2004).
2. Hawk Software, GLTrace Programming Utility. Available at htpp://www.hawksoft.com/gltrace.
3. Stanford University, GLSim & GLTrace. Available athttp://www.cs.virginia.edu/~gfx/Courses/2002/BigData/glsim.html,
4. SourceForge, D3D9Wrapper project. Available at http://sourceforge.net/.
5. SourceForge, DOGLESS project: OpenGL to ES realtime translator with extra profiling and debugging capabilities. Available at http://sourceforge.net/projects/dogless.
6. RealtechVR. Dxglwrap project. Available at http://sourceforge.net/projects/dxglwrap/
7. J.C. Dunwoody and M.A. Linton, "Tracing Interactive 3D Graphics Programs", Proc. of ACM Symposium on Interactive 3D Graphics (1990).
8. D. Kirk, "Unsolved Problems and Opportunities for High quality, High-performance 3-D Graphics on a PC Platform". ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware (1998).
9. I. Antochi, B.H.H. Juurlink, A. G. M. Cilio, P. Liuha, "Trading Efficiency for Energy in a Texture Cache Architecture", Proc. of the 2002 Euromicro Conference on Massively-Parallel Computing Systems, Ischia, Italy, 189-196 (2002).
10. Tulika Mitra and Tzi-cker Chiueh, "Dynamic 3D Graphics Workload Characterization and the Architectural Implications", 32nd ACM/IEEE Int. Symp. on Microarchitecture (MICRO), Haifa, Israel, 62-71 (1999).
11. Iosif Antochi, Ben Juurlink, and A. Cilio, "A Low-Cost, Power-Efficient Texture Cache Architecture", Proc. of the 12th Annual Workshop on Circuits, Systems, and Signal Processing (ProRISC2001), Veldhoven, The Netherlands, 29-30 (2003).
12. The Mesa Project, The Mesa 3D Graphics Library, Available at http://www.mesa3d.org.
13. Neil Trevett, "The State of the Mobile 3D Industry - continuing to grow rapidly". Game Developers Conference 2004, San Jose, California (2004).
14. Id Software Inc., Quake 2 & Quake 3. Available at http://www.idsoftware.com.
15. Ronny André Reierstad, 3rd person project. Available at www.morowland.com.
16. IICM, Graz University of Technology, Austria. Sample VRML models. http://www2.iicm.edu/vrml.