Synthetic content approach for benchmarking mobile 3D graphics

Kari J. Kangas* Nokia
Mika Qvist† Nokia
Kari Pulli‡ Nokia

Abstract

This work-in-progress paper describes a synthetic content approach for measuring OpenGL ES 3D graphics performance of mobile devices. Our approach relies on a synthetic content tool that can create different kinds of OpenGL ES graphics content according to a large number of input parameters. The input parameters are obtained by analyzing real OpenGL ES content with an OpenGL ES tracer. The synthetic content is validated by comparing the performance of the real and synthetic contents in the same platform. Although we do not yet have all the required elements needed by our synthetic benchmark approach, initial studies have produced promising results which demonstrate that the synthetic content can match quite well the real content from the performance point of view.

1 Introduction

Being able to benchmark the OpenGL ES performance as early as possible in the design cycle of a mobile device is important. For example, the benchmark results can be used to guide performance optimization. Benchmarks can be used to understand the performance of a particular mobile device and how the performance changes with different content; this is especially important for understanding the performance of an OpenGL ES solution obtained from an outside supplier. We can also provide good estimates of the available graphics performance of an upcoming mobile device to content developers so that they can start designing content before they have seen the actual device.

However, benchmarking a device while it is still under development can be challenging. The most obvious benchmarking approach would be to use publicly available OpenGL ES games and other applications. Alternatively, we could use dedicated benchmark software such as FutureMark's SPMark04 or 3DMarkMobile06. These approaches, however, have their shortcomings. Third-party benchmark applications usually require completely integrated operating system with GUI support and such environments may not be available until quite late in the device development cycle. Software binary breaks may make it impossible to run existing binaries in new devices. The source code of benchmark applications may not be available, or if it is, it typically requires continuous, non-trivial porting efforts. Device manufacturers do not usually grant access to software development environments to outside companies until the product has been publicly launched, and some devices do not even support third party native applications. Finally, the OpenGL ES content in mobile devices ranges from simple 3D UI elements to Java M3G games to full-blown native OpenGL ES games. Therefore, using only few benchmark applications does not cover the expected range of OpenGL ES use cases very well.

In order to avoid these problems, we have adopted a synthetic benchmark content approach for measuring the OpenGL ES performance of mobile devices. Our approach relies on a special synthetic content tool that can create varying OpenGL ES graphics content according to a large number of input parameters. For measuring the OpenGL ES performance, the tool is executed on a target platform with selected input parameters. When executed, the tool draws the same synthetic content repeatedly and measures the steady-state frames-per-second (FPS) performance of the particular platform we are benchmarking.

The main input parameters for our synthetic content tool are triangle count, triangle size, and overdraw factor (how many times each pixel is written to, on the average). Triangles can be textured and we can control, for example, the texture type and texture filtering mode. The synthetic content tool has been written mostly using platform-independent ANSI C, with a special emphasis on making the software as easy to port as possible to different platforms. The OpenGL ES content used in benchmarks is created completely during run-time to keep the binary size small and to eliminate the need for artistic work.

The main problem with synthetic benchmark content is the question whether or not the synthetic content is similar enough (from the performance point of view) to the real content to be usable for benchmarking. The solution we present in this paper is that we measure features such as triangle count, average triangle size, and overdraw factor from real content, create synthetic benchmark content that has matching features, and then compare the performance of the real content and the synthetic content on the same platform. If the performance matches well enough, we conclude that we have successfully synthesized the real content.

Characterizing real-world 3D graphics workload for benchmarking purposes is described for example by Dunwoody and Linton [1990], Mitra and Chiueh [1999], and more recently by Antochi et al. [2004]. In our ongoing work, we are not focusing so much on analyzing the workload features, but more on the question how the analyzed workload features can be mapped into synthetic benchmark content so that the original workload and the synthetic workload are similar enough from the performance point of view. Our work resembles also the work done in render-time estimation (see for example [Funkhouser and Sequin 1993]) in a sense that we approximate real workload with a simplified synthetic workload (or model).

Although we do not yet have all the required elements needed by our synthetic benchmark approach, initial studies so far have produced promising results.

2 Creating synthetic benchmark content

The process of creating synthetic benchmark content is illustrated in Figure 1. We begin the synthetic benchmark content creation by running the OpenGL ES application we wish to synthesize on top of an OpenGL ES tracer. The tracer is a special version of OpenGL

*e-mail: kari.j.kangas@nokia.com
†e-mail: mika.qvist@nokia.com
‡e-mail: kari.pulli@nokia.com

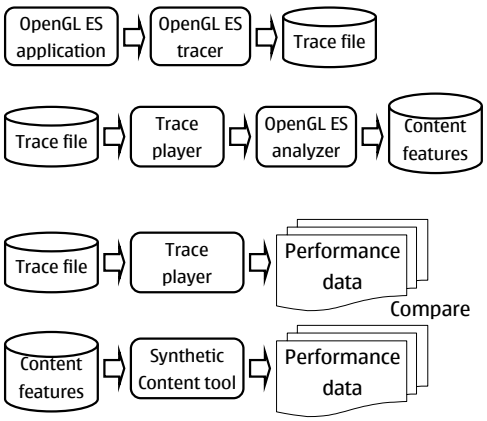

Figure 1: Synthetic benchmark content creation process.

ES library that captures the sequence of OpenGL ES calls, with the corresponding input parameters, into a trace file and then forwards the calls and parameters to the actual OpenGL ES library. To keep the size of the trace reasonable, we can instruct the tracer to capture only the OpenGL ES calls comprising the specific frames we are interested in subsequent analysis. This also allows us to minimize the delay caused by the trace capture as we can use a memory buffer of limited size for trace storage instead of constantly writing the trace into a memory card or similar mass storage.

The purpose of the tracer is to separate the OpenGL ES graphics calls from the OpenGL ES application. We can use the OpenGL ES trace player to replay the sequence of OpenGL ES calls, which recreates the OpenGL ES graphics in a controlled environment. For example, we can measure the performance of traced OpenGL ES content potentially on any OpenGL ES platform.

To get the content features such as triangle count, average triangle size, and overdraw factor for each frame in the traced OpenGL ES content, we feed the trace to an OpenGL ES analyzer. The OpenGL ES analyzer is a special version of the OpenGL ES library that keeps track of various content features used during rendering. For example, the analyzer records the number of incoming triangles, the average size of triangles (in fragments), and the total number of fragments written to the frame buffer for each OpenGL ES frame.

It is possible run the OpenGL ES application directly on top of the OpenGL analyzer to get the content features immediately as the application is run. However, the processing needed for the content analysis may interfere with the rendering so that some applications might, for example, skip frames to keep the rendering in sync with the audio playback. Alternatively, the analyzer can be implemented so that it analyzes the OpenGL ES calls directly without doing any actual rendering.

After the content features are extracted from the trace, we create synthetic benchmark content that has the matching content features. We can either make a matching synthetic frame for each original content frame, or we can make a single synthetic frame that represents a large combination of content frames. We can run the synthetic content tool on top of our OpenGL ES analyzer to make sure the content features are similar in both the original and synthetic content. Synthetic content tool will not skip any frames even if the rendering is very slow as the tool fully controls its own execution.

In order to verify and validate the synthetic benchmark content, both the synthetic the actual OpenGL ES trace are rendered on the same OpenGL ES platform. If the performance does not match within desired limits, we can refine the models in which the synthetic content tool creates OpenGL ES content from the input parameters. We can also modify the analyzer, if new types of input parameters are needed. Once we have successfully synthesized the content in one OpenGL ES platform, we can verify that the performance matches also on different platforms.

Eventually, the performance of the synthetic benchmark content should match the performance of the original OpenGL ES content on all tested platforms. When this happens, we can use it for benchmarking purposes instead of the original OpenGL ES content. We can even extrapolate from existing content so that we can test system responsiveness on content that has not been created yet.

3 Discussion and Future Work

We do not yet have a working OpenGL ES tracer, trace player, or analyzer. Instead, we have these tools for (desktop) OpenGL, and we have used them to analyze the content features of existing OpenGL games such as Quake and used the resulting features to create synthetic OpenGL ES benchmark content. The synthetic content tool works on top of both OpenGL (Win32) and OpenGL ES (Win32, Symbian, and PocketPC).

To provide some evidence that our current synthetic content tool can produce useful benchmark content, we have analyzed by hand the content features of one moderately complex OpenGL ES application (Nokia E3 2005 demo, see http://web.ngage.com/e3/video/videos.html?ID=22). We created matching synthetic benchmark content and compared the performance. The FPS performance of the real application was around 20 FPS whereas the FPS performance of the synthetic content was 24 FPS. This quick comparison should be considered only as an early indication that the performance of our synthetic benchmark content is roughly in the same ballpark as the real OpenGL ES with the matching content features.

During our work, we have found out that synthetic content tool is a very handy tool for experimenting how a particular OpenGL ES platform performs with different kinds of content. Our tool has a large set of easily configurable input parameters so creating a wide variety of synthetic content is easy. We have also developed tools for rapid presentation and comparison of the benchmark results. Both the easy content configurability and the ability to rapidly present the results have proved to be very valuable for understanding and especially communicating the 3D performance issues. As an example, if someone asks what happens to the performance once we change the texture filtering mode from nearest to bilinear and keep everything else as is, we can quickly measure it and present the results in an easily understandable format.

We have also investigated the possibility of using the trace as benchmark content. The main problem with trace is that the benchmark content (trace) cannot be modified very easily. For example we cannot easily modify the triangle count per frame and see how that affects the performance. However, once we analyze the content, we can change some parameters to create speculative benchmark content estimating possible future OpenGL ES applications, which we can use to see how the performance of a certain platform scales up.

Our future work comprises of developing the OpenGL ES tracer, trace player, and analyzer. After these tools are in place, we can properly analyze how well our synthetic benchmark content matches the source OpenGL ES content using more applications and different platforms. We are also studying different mechanisms of how the analyzed content features are used best to create syn-

thetic content with matching content features and similar performance characteristics.

We are also extending our synthetic content tool so that it can support other types of workloads besides OpenGL ES. For example, the synthetic content tool can generate synthetic CPU and memory load, play audio, and do game physics engine calculations while rendering OpenGL ES graphics. The synthetic content tool is designed so that diverse workloads can be mixed and matched easily within the same benchmark frame. Having this kind of mixture of workloads allows us to analyze the total system performance for complete and more realistic applications and use cases instead of concentrating on unrealistic graphics-only applications. As an example, we can easily test whether a particular phone model is capable of simultaneously transmitting and receiving wireless data at certain rate, playing mp3 with certain bit rate, calculating collision detection for three objects, and rendering graphics similar to Quake at 20 frame-per-second. By using a current analyzer, we can also easily measure how much power the phone consumes while doing all this. We feel that this flexibility allows us to use the synthetic content tool in scenarios that are not supported directly by other synthetic benchmark systems that concentrate only on one specific performance area such as 3D graphics performance.

We have also investigated the possibility of analyzing the high level structure of real-world OpenGL ES frames from the OpenGL ES trace. This structure could highlight for example how the application first draws the background and the background objects, followed by the foreground objects, followed by a special effects layer. We think that this structural information will allow us to better understand the important features of real OpenGL ES content. We also think that we could use the structural information to improve the quality of our synthetic content, for example by compositing the synthetic benchmark content frame from a set of synthetic content objects which are structured within a frame in a similar way as their counterparts in the OpenGL ES trace.

Finally, we are also extending our synthetic benchmark tool to support OpenVG benchmarking.

4 Conclusion

We have presented an approach and some preliminary results for estimating 3D performance of mobile devices while they are still under development and even when there is little existing 3D content available. Such a system is valuable both for engineers optimizing the design as well as for content creators preparing applications for the device. In both cases, performance estimates for applications could be made available even before hardware to run them exists. The crux of our future work lies in transferring our analysis tools from OpenGL to OpenGL ES, extending the analysis of content features and validating them with more experiments, and including other workloads such as the use of CPU for application logic and audio.

References

ANTOCHI, I., JUURLINK, B., VASSILIADIS, S., AND LIUHA, P. 2004. GraalBench: A 3D graphics benchmark suite for mobile phones. In *LCTES '04: Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, ACM Press, New York, NY, USA, 1–9.

DUNWOODY, J. C., AND LINTON, M. A. 1990. Tracing interactive 3d graphics programs. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics*, ACM Press, New York, NY, USA, 155–163.

FUNKHOUSER, T. A., AND SEQUIN, C. H. 1993. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 247–254.

MITRA, T., AND CHIUEH, T. 1999. Dynamic 3d graphics workload characterization and the architectural implications. In *MICRO 32: Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture*, IEEE Computer Society, Washington, DC, USA, 62–71.