HARDWARE 3D GRAPHICS ACCELERATION FOR MOBILE DEVICES

Thomas J. Olson

Texas Instruments, Inc.

ABSTRACT

Mobile phone handsets are evolving rapidly from simple voice communications terminals into portable multimedia computers. Applications of these devices include photo and video capture and display, stored or streaming media playback, web access, and 3D video games. These applications, together with attendant improvements in display quality and processing power, have created a need for hardware graphics accelerators optimized for mobile devices. This paper reviews the state of mobile 3D graphics, focusing on factors driving adoption, standards, and opportunities. It also discusses technical and practical challenges that must be met to enable a lively and successful content industry.

Index Terms— Mobile 3D Graphics, M3G, OpenGL ES

1. INTRODUCTION

Over the next few years, hardware graphics acceleration in mobile phones will go from a luxury found only in the most expensive models to an expected feature of all but the most basic products. This change is happening today, driven by the steady evolution of handsets from voice communications terminals to personal digital assistants and, ultimately, portable multimedia computers. The new applications demand high-quality rendering on high resolution displays, without large increases in power consumption or silicon area. Hardware acceleration is necessary to meet these needs.

This paper reviews the current state of mobile graphics, with a focus on general-purpose devices with relatively open architectures rather than on closed or special-purpose platforms such as mobile game consoles. The next section discusses factors driving the adoption of mobile 3D hardware. The following sections discuss the role of various types of graphics standards. Finally, the paper discusses technical and practical (market-related) problems that must be solved in order for mobile 3D graphics to achieve its potential.

2. MOTIVATION

A number of forces are converging to drive the adoption of hardware graphics acceleration in mobile phone handsets. These include changing consumer expectations, emerging applications, and changes in display size.

2.1. Changing Consumer Expectations

At the most abstract level, the driver for increasing use of graphics hardware is changing consumer expectations for what mobile phones should look like. In particular, the extensive publicity given to the Apple iPhone™, Nokia N-series, and other smartphones has had a major impact. Currently these are high-end products, but consumers will demand a similar look and feel in mid-range devices as soon as it is possible at a mid-range price.

2.2. New Applications

A second driver for the adoption of hardware graphics acceleration is the emergence of new applications that can benefit from it. These include:

Games – Mobile games are becoming a significant source of revenue for operating companies in Europe and Asia, and revenues are increasing in the Americas as well. While game developers rightly point out that graphics is not required to provide good game play, consumer expectations are strongly influenced by their experience with PC and console games. The result is that graphics and gaming are closely linked in consumers minds, and graphics capabilities are a selling point for consumers interested in games.

User Interfaces – Even consumers who are uninterested in games may be attracted by high-end 'compositing' user interfaces, which are made possible through hardware graphics acceleration. This capability has been available on the desktop for some years in Apple's Mac OS®, and is supported as an option in Microsoft's Windows Vista™. Again, the publicity given to Cover Flow™ on the iPhone has shown that a 3D-based interface can be compelling, even for applications such as music players where the need for graphics is not immediately obvious. Demand for similar-looking products is high, and compositing interfaces (or interfaces made to look like them) are likely to penetrate the market quickly.

Vector Graphic Applications – Many applications can benefit from high-quality display of 2D vector graphics, made popular through web formats such as SVG and Flash®. These include, for example, high-resolution mapping in GPS-enabled phones. Vector graphic rendering has subtleties due to its use of complex primitives (e.g. Bézier curves), specialized blend modes, and high-quality antialiasing. Dedicated 2D vector accelerators are available, but 3D accelerators can also be effective.

Multimedia – Multimedia applications such as video playback or image browsing do not require 3D acceleration, but they can benefit if it is present. 3D accelerators are adept at moving, transforming, resampling, and combining image arrays efficiently. Thus they are well suited to tasks like displaying translucent playback controls superimposed on a live video playback window, or making a playback window change size or transparency to allow display of a calendar reminder.

2.3. Display Size

A final factor driving the adoption of graphics accelerators is the steady increase in screen resolutions for mobile devices. Many

phones sold today still have main screens with resolutions derived in some way from QCIF (e.g. 176x220), but QVGA (320x240) resolution is now common, and VGA (640x480) is becoming a requirement in some markets.

As a practical matter, it is unlikely that handsets with conventional form factors will adopt resolutions much higher than VGA for their main screens, because a 2.2 inch VGA screen nears the limits of human visual acuity. However, many high-end phones are equipped with video ports allowing them to be connected to external monitors. This allows the consumer to display images taken with the handset camera on a high-quality display. This path could lead to mobile phones with HDTV 1080p (1920x1080) display capability within the next few years.

The implications of $O(n^2)$ growth in screen resolution will make hardware acceleration mandatory well before HDTV resolution is reached. Table 1 shows why. The table assumes that 30 frames-per-second (fps) rendering will be used for applications requiring smooth motion, such as games, video playback, and user interface animations, and that the phone's central processor is clocked at 200 MHz. The table gives the number of pixels per second that must be rendered at various screen sizes, and then translates that figure into the number of CPU cycles available to generate each pixel. These numbers are optimistic, for a number of reasons. First, the CPU cannot devote all of its cycles to pixel processing; it must also run the operating system and device drivers, process scene geometry, and execute the application. These loads are highly application dependent, but it is not unusual to have less than 50% of cycles available for pixel processing. Second, in many applications pixels are written to more than once per frame, a phenomenon called *overdraw*. This too is application dependent, but overdraw multiples of 2x to 3x are common in 3D games. These factors can easily cut the *effective* number of available cycles per screen pixel by a factor of four to six.

A typical mobile CPU can render a basic (e.g. nearest-neighbor, mipmapped, textured) pixel in on the order of twenty cycles. Adding features such as bilinear texture filtering, blending, and multisampling can increase that cycle count by an order of magnitude. Comparing the number of CPU cycles required to paint a pixel to the number of cycles available (Table 1) makes it clear that the days of CPU-based rendering are numbered. A 200 MHz CPU can be used for 3D rendering up to QVGA, but will be heavily stressed at that screen size; applications will be limited in the quality and complexity of the scenes they can render. At larger screen sizes, high-end rendering in software is out of the question.

Making different assumptions about frame rate or CPU speed change the picture, but only a little. Doubling the CPU clock to 400 MHz makes QVGA reasonable and HVGA possible, again with significant compromises. But even a 1 GHz general-purpose processor is marginal for rendering to a VGA screen, and higher resolutions remain out of reach.

3. THE ROLE OF STANDARDS

Application Programmer Interface (API) standards play a critical role in enabling the adoption of graphics hardware. Their most obvious contribution is that they allow application code to be portable across accelerators, or even between accelerated and non-accelerated platforms. This increases the number of platforms that can be targeted by a single application, reducing development cost per potential customer. This is critical, especially early in the adoption cycle when the number of accelerated platforms is small.

Size	pix/frame	Mpix/sec	cycles/pix
176x220	38720	1.16	172.18
QVGA	76800	2.30	86.81
HVGA	153600	4.61	43.40
VGA	307200	9.22	21.70
WVGA	408960	12.27	16.30
720p	921600	27.65	7.23
1080p	2073600	62.21	3.22

Table 1: Graphics Pixel Processing Requirements as a function of screen size. Pix/sec are in millions, assuming 30fps rendering. Cycles/pix assume a 200 MHz CPU.

A second, less obvious role of API standards is that they provide a common definition for the functionality that graphics accelerator designers should provide. Platforms that support the standard benefit not only from portable content developed for the standard, but from development tools, training, and support oriented toward the standard.

3.1. Taxonomy of Mobile Graphics Standards

The most powerful handsets available today are quite capable of running desktop API standards of recent vintage. However, mobile phone execution environments and applications are very different from the desktop, and the industry has found it worthwhile to develop new, mobile-specific graphics standards. These standards can be analyzed along several dimensions:

Low Level vs High Level – Low-level graphics APIs are designed to expose as much as possible of the hardware functionality, while still providing reasonable abstraction and portability. In these *immediate-mode APIs* the application makes explicit calls to draw primitive shapes such as points, lines, or triangles, and the hardware (at least conceptually) draws them immediately. By contrast, high-level or *retained-mode* APIs allow the application to build highly structured representations of scenes ("scene graphs"), potentially including temporal behavior. The application makes calls to construct the scene and render it, and the API takes care of breaking it down into hardware-renderable primitives.

In applications based on a low-level rendering API, the application must provide the functionality that would otherwise be provided by a high-level API, and must translate the high level data structures into primitive draw commands. This is more work for the application developer, but it has advantages in terms of flexibility and efficiency. When this work is done in the application, the code can be optimized for the application domain; for example, a ideal scene structure for a war game might be quite different from that for a flight simulator. A high-level API must provide a general-purpose scene structure that can apply to any domain, and this generality has a cost. For this reason, PC and console games are usually written to use low-level APIs.

The dominant low-level API for mobile applications is OpenGL ES [6][9], a mobile-oriented version of OpenGL [10], which is the leading open standard for 3D rendering on workstations and PCs. OpenGL ES differs from OpenGL in discarding legacy and/or redundant functionality, and in adding mobile-specific features such as fixed-point data types. On mobile versions of Microsoft platforms, OpenGL ES competes with Direct3D® Mobile [8], which is based on desktop Direct3D.

Java vs Native – Base language can have a significant impact on API design. In many mobile phone execution environments, all externally supplied applications must be written in Java™ and applications delivered as native (binary) code are forbidden. The primary motivation for this restriction is security; Java programs can be prevented from damaging the system by the runtime byte code interpreter. Security is less of an issue now that mobile phone processors have begun to incorporate hardware memory protection and modern operating system principles, but Java continues to dominate in lower cost handsets.

Mobile Java implementations have historically suffered from poor performance due to byte code interpretation, and from slow or non-existent floating point support. This has motivated designers to try to move sources of heavy computational load out of Java and into native code libraries, where the same computation can be done more quickly. In 3D games, scene graph manipulation and rendering is such a load, so mobile graphics APIs for Java often include high-level features. The "Mobile 3D Graphics" libraries, M3G (JSR-184) [2][9] and M3G2 (JSR-297) [3], are examples. There is also a low-level Java library, JSR-239 [4], which provides a direct Java binding to OpenGL ES 1.1.

Currently, there are no standard high-level APIs for native code execution environments, because programmers generally prefer to code that functionality themselves. For those who do not, it is possible to license native-code game engines from various commercial vendors. These engines provide high-level functionality which is typically specific to a class of games, and hence is more efficient than a general-purpose high-level API.

Fixed Function vs Programmable – On the desktop, graphics hardware has undergone a paradigm shift over the past decade. First-generation graphics processors were conceptualized as fixed-function hardware pipelines controlled by a large number of state registers, and were programmed by setting register values and then sending vertex data through the pipeline. In the new paradigm, the pipeline is still present, but the most important fixed-function units have been replaced by programmable processors that execute programs written in specialized graphics programming languages.

In the mobile space, the transition to programmability is under way but is not yet complete, so the APIs support a mix of functionality. First generation APIs provide fixed functionality; they include OpenGL ES 1.0 and 1.1, Direct3D Mobile, and the Java standards M3G and JSR-239. The recently released OpenGL ES 2.0 supports the programmable model, as does Java's M3G2 (currently in definition).

It should be noted that as desktop graphics accelerators have become more programmable, they have begun to be used for non-graphics applications. This trend is likely to occur in mobile devices too. Programmable graphics processors will increase the floating point performance of mobile phone CPUs by several orders of magnitude. Possible applications include geometric reasoning for augmented reality, image processing and analysis, and game physics.

4. TECHNICAL CHALLENGES

Mobile graphics is in some sense recapitulating the history of graphics acceleration on the desktop, so it might seem that technical solutions of a few years ago should be able to meet today's needs. There is some truth to this, but only some, for two reasons. First, the mobile APIs have adopted modern graphics API features (such as programmability) wherever possible, and older APIs and architectures lack these features. Beyond that, mobile platforms have constraints which are quite different from those of desktop, line-powered devices, and these demand different solutions. Two constraints that are particularly noteworthy are power and memory bandwidth.

4.1. Power

Mobile devices are by definition battery-powered, and consumers demand ever-increasing battery life. Thus the graphics solution must compete for a slice of a constantly decreasing system power budget. This is in marked contrast to desktop systems, where high-end graphics cards commonly exceed the power that can be delivered by standard expansion busses, and need auxiliary power connectors. Mobile hardware graphics accelerators use far less power to render a given scene than do general-purpose CPUs, but they also encourage designers to make heavy use of graphical features, so the graphics accelerator can still place a significant load on the battery. Thus, minimizing power dissipation is a critical challenge for mobile graphics accelerators.

Mobile graphics cores can of course benefit from standard low-power design techniques. Clock gating (disabling the clock to sections of the core that are idle) reduces dynamic (switching) power. As static (leakage) power becomes more important, more advanced techniques such as dynamic voltage scaling and power gating become important. Power gating in particular has an impact on graphics processor and driver design.

The rise of mobile graphics will create a need for more fundamental approaches to lowering the power requirements of graphics accelerators. In particular, reducing memory bandwidth will be important, since off-chip memory accesses can consume large amounts of power.

4.2. Memory Bandwidth

Mobile devices have memory bandwidth limitations that pose severe problems for 3D graphics accelerators, even beyond the power issue referred to in the previous section. Bandwidth is a concern on the desktop as well, but there it is primarily a cost issue, and all but the lowest of low-end graphics accelerators have dedicated banks of graphics memory, typically accessed through wide busses. The physical constraints of the handset form factor discourage this approach in mobile devices, even if power were not a problem. Advanced packaging techniques such as die-stacking can help, but they add to cost and interfere with stacking other components, such as system DRAM or the cellular modem. Therefore, mobile graphics cores must be architected from the start to minimize memory bandwidth.

As a result of the memory bandwidth limitation, a number of successful mobile graphics cores have adopted some form of tile-based rendering architecture. In these systems, the frame buffer is logically partitioned into disjoint regions. When the application draws a triangle, the hardware does not actually render it. Instead, it determines which regions the triangle will affect, and writes that information into a database. When the application signals that the frame is complete, the accelerator iterates over the regions. For each regionm, it reads data for all of the triangles that affect the region, and renders them into an on-chip SRAM buffer. When all triangles for the region have been rendered, its pixels are written to external memory and the next region is processed. This usually reduces memory bandwidth, since most frame buffer references are directed to on-chip memory. However, it requires each triangle to

be read once for each tile it affects, so in geometry-heavy scenes it can actually increase total memory traffic.

Mobile-specific architectures and rendering techniques remain a fruitful research area. For example, Akenine-Möller and Ström [1][11] have described interesting techniques for bandwidth-efficient antialiasing, as well as a texture compression algorithm that has been adopted as an optional extension to OpenGL ES [5]. Many interesting problems remain to be solved, and research results are likely to find rapid application.

5. PRACTICAL CHALLENGES

Although there are challenges, from a technical point of view the future of mobile 3D graphics is bright. Devices on the market today rival video gaming consoles of only a few years ago, and devices on the drawing board will bring a staggering level of performance to handheld devices. The most difficult challenges facing mobile graphics on the handset are not technical, but are related to the structure and economics of the wireless industry.

5.1. Developer Education

Programming for hardware-accelerated graphics engines is a specialized skill. Developers who have that skill generally learned it in a desktop environment, where they did not have to contend with the limitations of mobile devices. These limitations include small memories, slow processors, lack of floating point, and quirky development environments. Programmers who have worked with video game consoles are generally better prepared for this than PC developers, because video game consoles have many of the same limitations. Still, graphics programmers moving into the mobile industry need a period of retraining before they can be productive.

The situation is more difficult for mobile game programmers whose previous experience has been with software-based graphics running on general-purpose CPUs. With a software renderer, it is perfectly reasonable to draw a scene by sending one triangle at a time to the rendering API. Drawing triangles in larger groups is more efficient, but the difference is usually small. With a hardware accelerator, drawing one triangle at a time is *slower* than doing all of the rendering in software. This is because graphics hardware is organized as a very deep pipeline, which is intended to be only loosely coupled to the CPU. This presents multiple challenges for mobile graphics programmers; first, they have to learn a new and difficult way of thinking about rendering and computation, and then they have to discard or rewrite legacy software that is written in the older, more synchronous style.

5.2. The "snowflake problem"

The biggest barrier to developing a large-scale market for mobile phone applications of any kind is the diversity of execution environments. Not only does every mobile phone handset present a unique environment, but so does the *same* handset as delivered by different network operators. Developers call it the "snowflake problem"; no two handsets are alike. It is common for a large-scale mobile game developer to maintain hundreds of distinct builds of a single game. Minor differences can be handled via abstraction layers and automated build systems, but many differences can only be handled manually, and both approaches add cost.

The larger handset vendors and network operators have established internal standards intended to provide portability within their own product lines, but generally they have not been willing to give up the perceived value of differentiating their platforms for the theoretical value of creating a larger market.

The best hope for a solution to the snowflake problem lies in open standards. OpenGL ES has largely solved the problem for low-level 3D graphics, at least on devices with hardware acceleration. Application vendors still have to contend with different versions of the standard, and with different performance levels on different handsets, but the situation is now no worse than it is on the desktop. However, OpenGL ES does nothing to address non-graphics-related differences between platforms.

The Khronos™ Group, the non-profit standards body which oversees the evolution of OpenGL ES and OpenGL, has launched an initiative called OpenKODE™ [7] to define an execution environment for rich media applications that standardizes access to operating system services, graphics, and multimedia. If it is widely adopted, even as a 'portability sandbox' within a differentiated environment, it promises to enable for the first time an economically viable market for multimedia and graphics applications on mobile devices.

6. REFERENCES

[1] T. Akenine-Moller and J. Strom, "Graphics for the Masses – A Hardware Rasterization Architecture for Mobile Phones," *ACM Transactions on Graphics, v.22 no.3 (Proc. ACM SIGGRAPH 2003)*, ACM Press, New York, pp. 801-808, July 2003.

[2] Java Community Process, "JSR-000184 Mobile 3D Graphics API for J2ME", http://jcp.org/aboutJava/ communityprocess/final/ jsr184/

[3] Java Community Process, "JSR 297: Mobile 3D Graphics API 2.0", http://jcp.org/en/jsr/detail?id=297

[4] Java Community Process, "JSR-000239 Java Bindings for OpenGL ES API", http://jcp.org/aboutJava/communityprocess/ final/jsr184/

[5] The Khronos Group, "OES_compressed_ETC1_RGB8_ texture", 2006, http://www.khronos.org/registry/gles/extensions/ OES/OES_compressed_ETC1_RGB8_texture.txt

[6] The Khronos Group, "OpenGL ES Overview", 2007, http://www.khronos.org/opengles/

[7] The Khronos Group, "OpenKODE Overview", 2007, http://www.khronos.org/openkode/

[8] Microsoft, "Direct3D Mobile", 2007, http://msdn2.microsoft. com/en-us/library/aa921056.aspx

[9] K. Pulli, T. Aarnio, K. Roimela, and J. Vaarala, "Designing Graphics Programming Interfaces for Mobile Devices", *IEEE Computer Graphics and Applications* v. 25, no. 6, 2005, pp 66-75.

[10] M. Segal and K. Akeley, "The Design of the OpenGL Graphics Interface", Technical Report, Silicon Graphics Inc, 1994.

[11] J. Ström and T. Akenine-Möller, "iPACKMAN: High-Quality, Low-Complexity Texture Compression for Mobile Phones", *Proc ACM SIGGRAPH / EUROGRAPHICS Conf. on Graphics Hardware*, ACM Press, New York, pp. 63-70, July 2005.