Editor: Lawrence Rosenblum and Simon Julier

Making Augmented Reality Practical on Mobile Phones, Part 1

Daniel Wagner and Dieter Schmalstieg, *Graz University of Technology*

In the past few years, mobile phones have become an increasingly attractive platform for augmented reality (AR; see Figure 1 for some examples). According to Gartner, 1.22 billion mobile phones were sold in 2008.[1] Some forecasts estimate that this number will rise to 1.8 billion units in 2012, of which 800 million are expected to be smartphones.[2] Although not all these devices are open for custom software development, the trend toward open software environments for mobile phones seems inevitable.

In 2003, we started an AR framework for mobile phones (for a glimpse of other research on AR for mobile phones, see the sidebar). We intended its first generation as primarily a proof of feasibility. The second generation was an attempt to port a fully featured PC-based AR framework, Studierstube 4 (http://studierstube.org), to a phone platform. You can port existing applications and make them run on mobile phones. However, as we had to painfully experience ourselves, this approach typically produces slow, bloated, and unstable software. Optimally using phones' scarce resources requires different algorithms and architectural decisions than for PCs, leading to a complete reengineering of an existing solution.

So, for the third generation, Studierstube ES, we largely abandoned compatibility requirements and added new elements to the design, such as an asymmetric client-server technique, that are specific to mobile devices.[3] In this first installment of our two-part tale of Studierstube ES and what we've learned along the way, we describe the mobile phone platform's restrictions and how our software architecture allows fast development of mobile phone AR applications.

Software Architecture

A major difference between PCs and mobile devices is the amount and bandwidth of available memory. Mobile devices still can handle only a few megabytes per application. Memory access is comparatively slow, and even the transfer of megabytes of application code over the air is costly in some cases. Consequently, software frameworks must be at least an order-of-magnitude slimmer than is common on PCs. This constrains many design decisions—for example, heavy use of templated C++ code results in a prohibitive increase in code size. For instance, Studierstube 4 has a modest 8 Mbytes of binary for a minimally useful set, and other AR frameworks are reportedly 5–10 times larger.

In contrast, the binary of Studierstube ES for Windows Mobile 5 is about 1 Mbyte, with all standard features enabled as shown in Figure 2. A positive side effect of such a lean approach is that the framework also runs very fast on a PC, which we use mainly for prototyping mobile applications.

Studierstube ES's main duties are hardware and API abstraction as well as providing high-level services to applications built on top of it. At its lower levels (Studierstube Core), it unifies native APIs such as for window management, user input, and file or camera access, for all supported platforms. The complete software stack can be compiled to use either fixed- or floating-point mathematics, to optimally support every possible device.

Studierstube ES applications typically generate graphical output by manipulating scene-graph objects, running on top of OpenGL ES (1.x and 2.x), OpenGL, or Direct3D Mobile. Although these APIs are quite different, they target the same hardware and can therefore be abstracted with a very thin layer.

Tracking pose with six degrees of freedom in real time is a fundamental requirement of any AR system. Studierstube Tracker is a high-performance tracking library for mobile phones. We developed it from scratch to optimally support tracking with limited resources. It supports multiple types of artificial fiducials and, recently, natural-feature tracking. There are no heap allocations at runtime, and Tracker stores data in cache-friendly data structures. So, the average smart phone can analyze a typical image in approximately 10 milliseconds, which leaves enough memory for graphics-intensive applications.

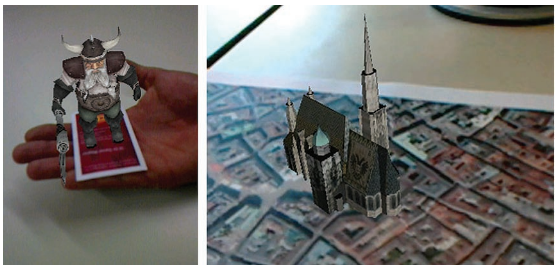

Figure 1. Examples of mobile-phone-based augmented reality (AR): (a) a virtual avatar on a business card and (b) a virtual building on a satellite picture using natural-feature tracking. Both applications run in real time (20 frames per second overall) on a mobile phone.

To facilitate persistent location-based applications and multiuser collaboration, we designed a client-server system for multiuser applications. The server consists of a proprietary XML database combined with a finite-state machine. We designed it to run in real time with hundreds of clients. The server manages communication among clients and stores transient and persistent client data. This architecture both encourages data-driven designs and helps clients recover from network loss or crashes.

Component-based software design is a well-acknowledged way to create large software packages. Partitioning large software projects into small modules (dynamic-link libraries) with clear APIs can effectively reduce complexity. Yet, an often-forgotten disadvantage is that doing this usually increases the overall binary size, because a module always contains a component's full implementation. On the other hand, static linking of libraries can strip implementation code that the application doesn't need. So, it's worth carefully considering whether to use multiple modules, which makes software updates easier, or a single monolithic approach, which could lead to smaller applications.

Development and Debugging Strategies

Developers typically try to minimize effort by reusing previously written code. Instead of developing from scratch, they combine existing solutions to form new ones. Although this approach efficiently reduces development time, it leads to nonoptimal prototypes because most often the reused components were developed for different purposes. So, many research prototypes don't achieve high performance and compensate for this by using faster hardware.

On mobile phones, this approach isn't feasible because even the fastest devices are still comparatively slow. Because the available battery capacity (which increases approximately 10 percent per year) mostly drives development, no huge improvements are likely. In addition to raw performance, OS limitations, such as memory management constraints, make straight code porting impossible or at least extremely inefficient. A well-written application runs about 5-10 times

Related Work on Augmented Reality for Mobile Phones

Since we started our handheld augmented-reality framework in 2003, several other research projects have targeted similar applications. A group at Nokia research has developed a SURF (Speeded Up Robust Features) implementation that runs in real time on mobile phones.[1] Michael Zöllner and his colleagues developed a tracking system that combines randomized trees for detection with KLT (the Kanade-Lucas-Tomasi feature tracker) for tracking.[2] The system runs in real time on a PC and has been recently ported the iPhone, but not running in real time.

References

1. T.H.D. Nguyen et al., "SURFTrac: Efficient Tracking and Continuous Object Recognition Using Local Feature Descriptors," to be published in *Proc. Conf. Computer Vision and Pattern Recognition* (CVPR 09), IEEE CS Press, 2009.
2. M. Zoellner et al., "Reality Filtering: A Visual Time Machine in AR," *Proc. 9th Int'l Symp. Virtual Reality, Archaeology, and Intelligent Cultural Heritage* (VAST 08), Eurographics Assoc., 2008, pp. 71–77.

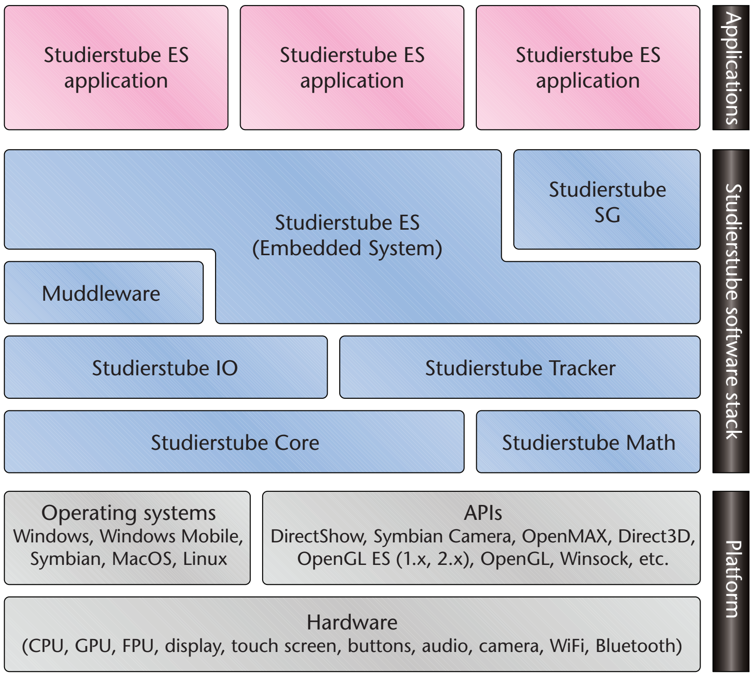
Studierstube ES application
Studierstube ES application
Studierstube ES application
Applications
Studierstube ES (Embedded System)
Studierstube SG
Muddleware
Studierstube IO
Studierstube Tracker
Studierstube Core
Studierstube Math
Studierstube software stack
Operating systems
Windows, Windows Mobile, Symbian, MacOS, Linux
APIs
DirectShow, Symbian Camera, OpenMAX, Direct3D, OpenGL ES (1.x, 2.x), OpenGL, Winsock, etc.
Hardware
(CPU, GPU, FPU, display, touch screen, buttons, audio, camera, WiFi, Bluetooth)
Platform

Figure 2. The software stack of the Studierstube handheld AR framework. The lower levels (Core, Math, IO, Tracker, and Muddleware) provide the basic functionality that an AR system requires. Studierstube ES (Embedded System) and SG (Scene-Graph) combine these services in a high-level layer for the applications running on top of it.

more slowly on a fast mobile phone than on an average PC.

So, a good practice is to rely only on software specifically developed for mobile phones. Because these libraries are usually developed by experts on the platform, they deliver high performance and robustness. Unfortunately, this forces developers to abandon existing standards and compatibility with legacy software. Often, development from scratch becomes necessary.

When no existing solution is available, developers must invest much effort into developing new solutions or porting existing ones. For example, the current version of Studierstube Tracker is about 100 times faster and uses only a fraction of memory compared to the first port of this software to mobile phones. On the down side, we had to invest about two person-years of work.

Even though mobile phone CPUs usually don't have parallel-execution units, you can use multithreading or interleaving to accelerate many operations.

Whereas mobile devices' basic working principles are similar to those of PCs, APIs differ a lot in the details. The Windows Mobile APIs are mostly similar to their PC-based counterparts, but the Symbian APIs differ strongly. Abstracting APIs makes software more portable among different target devices. It also lets you develop software mostly on the PC, running only final tests on the target platform.

All major mobile phone development kits come with emulators. However, these emulators aren't very close to real hardware. Performance is usually much lower than on real devices, and many restrictions (such as introduced by ARM CPUs or the OS) often aren't enforced. Specifically, emulators don't include camera support, which renders them mostly useless for developing AR applications.

A better solution is to test phone AR applications as native PC applications first, using the underlying hardware abstraction of Studierstube ES. Obviously, this approach doesn't consider certain limiting characteristics of mobile phones. However, we can solve most algorithmic problems in a convenient desktop environment.

Explicit Parallel Execution

Even though mobile phone CPUs usually don't have parallel-execution units, you can use multithreading or interleaving of actions to accelerate many operations, such as camera access or network communication. This is because those two techniques are I/O rather than CPU bound.

An AR application generally consists of five major tasks:

- camera reading (image retrieval),
- pose tracking,
- network communication (for multiuser applications),
- application overhead, and
- rendering.

At first glance, these tasks form a chain of actions in which each step depends on the previous step's outcome. Pose tracking requires a camera image, the application can't work without tracking and networking, and rendering relies on tracking and application results. However, you can relax this chain so that many actions can execute in parallel—even if no CPU support is available for multithreading, such as hyperthreading or multicore.

Camera reading is often performed by a dedicated CPU unit or directly by the camera chip. This is because the image must be scaled down from the native camera resolution, which is highly computationally expensive. Camera reading therefore requires only minimal overhead, unless the software must perform custom pixel format conversion. At the OS level, management of camera reading is often implemented in a separate thread. At the cost of increased latency—using the available image, rather than waiting for a new one—we can reduce camera-reading overhead to almost zero.

On a system with hardware that supports 3D rendering, pose tracking is often the most CPU-intensive task. It naturally depends on the latest camera image and is required for application logic and rendering.

Network communication can run asynchronously or in a separate thread. Because the amount of data to receive and process is usually small on a mobile phone, most of the time is spent waiting for replies. By restricting the networking to work asynchronously, you can efficiently interleave it with all other tasks. So, Studierstube ES applications usually expect network replies one frame after Studierstube ES has sent out requests.

Our experiences indicate that typical application logic's overhead is minimal compared to the framework's housekeeping, such as tracking and video rendering. We've therefore found it sufficient to let application logic execute at two points in the pipeline: directly before and directly after scene-graph rendering. Whereas the former lets applications modify the scene-graph, the latter targets custom 2D and 3D rendering.

Many of today's high-end phones have dedicated 3D acceleration for gaming, so we expect that a 3D accelerator will soon be an integral part of a standard phone CPU. Such 3D acceleration will reduce the rendering of optimized 3D content (fixed-point, vertex buffers) to mostly a background task.

To sum up, pose tracking, especially from natural features, presents the most CPU-intensive task. The other tasks either can run mostly in parallel (camera reading, network communication, and rendering) or require little overhead (application logic).

In part 2, we'll show how to maximize performance by careful memory use and efficient fixed-point mathematics. We'll also dissect the timings of typical AR applications on mobile phones and predict the future of mobile phone AR.

References

1. "Mobile Phone Sales Up Six Pct in 2008, 4th Qtr Weak," 3 Mar. 2009, Physorg.com; www.physorg.com/news155308101.html.
2. "Mobile Phones," Imagination Technology, 2009; www.imgtec.com/markets/mobile-multimedia.asp.
3. D. Schmalstieg and D. Wagner, "Experiences with Handheld Augmented Reality," *Proc. 6th IEEE/ACM Int'l Symp. Mixed and Augmented Reality* (ISMAR 07), IEEE CS Press, 2007, pp. 1–13.

Daniel Wagner *is a postdoctoral researcher at the Graz University of Technology and deputy director of the Christian Doppler Laboratory for Handheld Augmented Reality. Contact him at wagner@icg.tugraz.at.*

Dieter Schmalstieg *is a professor of virtual reality and computer graphics at the Graz University of Technology, where he directs the Studierstube research project on augmented reality. Contact him at schmalstieg@icg.tugraz.at.*