Chapter 1

Android, the New Kid on the Block

As a kid of the early nineties, I naturally grew up with my trusty Nintendo Game Boy. I spent countless hours helping Mario rescue the princess, getting the highest score in Tetris, and racing my friends in RC Pro-Am via link cable. I took this awesome piece of hardware with me everywhere and every time I could. My passion for games made me want to create my own worlds and share them with my friends. I started programming on the PC but soon found out that I couldn't transfer my little masterpieces to the Game Boy. I continued being an enthusiastic programmer, but over time my interest in actually playing video games faded. Also, my Game Boy broke . . .

Fast forward to 2010. Smartphones are becoming the new mobile gaming platforms of the era, competing with classic dedicated handheld systems such as the Nintendo DS or the Playstation Portable. That caught my interest again, and I started investigating which mobile platforms would be suitable for my development needs. Apple's iOS seemed like a good candidate to start coding games for. However, I quickly realized that the system was not open, that I'd be able to share my work with others only if Apple allowed it, and that I'd need a Mac to develop for the iOS. And then I found Android.

I immediately fell in love with Android. Its development environment works on all the major platforms, no strings attached. It has a vibrant developer community happy to help you with any problem you encounter as well as comprehensive documentation. I can share my games with anyone without having to pay a fee to do so, and if I want to monetize my work, I can easily publish my latest and greatest innovation to a global market with millions of users in a matter of minutes.

The only thing I was left with was actually figuring out how to write games for Android and how to transfer my PC game development knowledge to this new system. In the following chapters, I want to share my experience with you and get you started with Android game development. This is of course a rather selfish plan: I want to have more games to play on the go!

Let's start by getting to know our new friend: Android.

M. Zechner, *Beginning Android Games*

A Brief History of Android

Android was first publicly noticed in 2005 when Google acquired a small startup called Android, Inc. This fueled speculation that Google wanted to enter the mobile space. In 2008, the release of version 1.0 of Android put an end to all speculation, and Android became the new challenger on the mobile market. Since then, it's been battling it out with already established platforms such as iOS (then called iPhone OS) and BlackBerry, and its chances of winning look rather good.

Because Android is open source, handset manufacturers have a low barrier of entry when using the new platform. They can produce devices for all price segments, modifying Android itself to accommodate the processing power of a specific device. Android is therefore not limited to high-end devices but can also be deployed to low-budget devices, thus reaching a wider audience.

A crucial ingredient for Android's success was the formation of the Open Handset Alliance (OHA) in late 2007. The OHA includes companies such as HTC, Qualcomm, Motorola, and NVIDIA, which collaborate to develop open standards for mobile devices. Although Android's core is developed mainly by Google, all the OHA members contribute to its source in one form or another.

Android itself is a mobile operating system and platform based on the Linux kernel version 2.6 and is freely available for commercial and noncommercial use. Many members of the OHA build custom versions of Android for their devices with modified user interfaces (UIs)—for example, HTC's HTC Sense and Motorola's MOTOBLUR. The open source nature of Android also enables hobbyists to create and distribute their own versions of Android. These are usually called *mods*, *firmwares*, or *ROMs*. The most prominent ROM at the time of this writing was developed by a fellow known as Cyanogen and is aimed at bringing the latest and greatest improvements to all sorts of Android devices.

Since its release in 2008, Android has received seven version updates, all code-named after desserts (with the exception of Android 1.1, which is irrelevant nowadays). Each version has added new functionality to the Android platform that has relevance in one way or another for game developers. Version 1.5 (Cupcake) added support for including native libraries in Android applications, which were previously restricted to being written in pure Java. Native code can be very beneficial in situations where performance is of upmost concern. Version 1.6 (Donut) introduced support for different screen resolutions. We will revisit this fact a couple of times in this book because it has some impact on how we approach writing games for Android. With version 2.0 (Éclair) came support for multi-touch screens, and version 2.2 (Froyo) added just-in-time (JIT) compilation to the Dalvik virtual machine (VM), which powers all the Java applications on Android. The JIT speeds up the execution of Android applications considerably—depending on the scenario, up to a factor of five. At the time of this writing, the latest version is 2.3, called Gingerbread. It adds a new concurrent garbage collector to the Dalvik VM. If you haven't noticed yet: Android applications are written in Java.

A special version of Android, targeted at tablets, is also being released in 2011. It is called Honeycomb and represents version 3.0 of Android. Honeycomb is not meant to

run on phones at this point. However, some features of Honeycomb will be ported to the main line of Android. At the time of this writing, Android 3.0 is not available to the public, and no devices on the market are running it. Android 2.3 can be installed on many devices using custom ROMs. The only handset using Gingerbread is the Nexus S, a developer phone sold by Google directly.

Fragmentation

The great flexibility of Android comes at a price: companies that opt to develop their own user interfaces have to play catch-up with the fast pace at which new versions of Android are released. This can lead to handsets not older than a few months becoming outdated really fast as carriers and handset manufacturers refuse to create updates that incorporate the improvements of new Android versions. The big bogeyman called *fragmentation* is a result of this process.

Fragmentation has many faces. For the end user, it means being unable to install and use certain applications and features because of being stuck on an old Android version. For developers, it means that some care has to be taken when creating applications that should work on all versions of Android. While applications written for earlier versions of Android will usually run fine on newer versions, the reverse is not true. Some features added in newer Android versions are of course not available on older versions, such as multi-touch support. Developers are thus forced to create separate code paths for different versions of Android.

But fear not. Although this sounds terrifying, it turns out that the measures that have to be taken are minimal. Most often, you can even completely forget about the whole issue and pretend there's only a single version of Android. As game developers, we're less concerned with differences in APIs and more concerned about hardware capabilities. This is a different form of fragmentation, which is also a problem for platforms such as iOS, albeit not as pronounced. Throughout this book, I will cover the relevant fragmentation issues that might get in your way while you develop your next game for Android.

The Role of Google

Although Android is officially the brainchild of the Open Handset Alliance, Google is the clear leader when it comes to implementing Android itself as well as providing the necessary ecosystem for Android to grow.

The Android Open Source Project

Google's efforts are summarized under the name *Android Open Source Project*. Most of the code is licensed under Apache License 2, a very open and nonrestrictive license compared to other open source licenses such as the GNU General Public License (GPL). Everyone is free to use this source code to build their own systems. However, systems that are claimed to be Android compatible first have to pass the Android Compatibility

Program, a process ensuring baseline compatibility with third-party applications written by developers like us. Compatible systems are allowed to participate in the Android ecosystem, which also includes the Android Market.

The Android Market

The *Android Market* was opened to the public in October 2008 by Google. It's an online software store that enables users to find and install third-party applications. The market is generally accessible only through the market application on a device. This situation will change in the near future, according to Google, which promises the deployment of a desktop-based online store accessible via the browser.

The market allows third-party developers to publish their applications either for free or as paid applications. Paid applications are available for purchase in only about 30 countries. Selling applications as a developer is possible in a slightly smaller number. Table 1–1 shows you the countries in which apps can be bought and sold.

Table 1–1. *Purchase and Selling Options per Country.*

Country	User Can Purchase Apps	Developer Can Sell Apps
Australia	Yes	Yes
Austria	Yes	Yes
Belgium	Yes	Yes
Brazil	Yes	Yes
Canada	Yes	Yes
Czech Republic	Yes	**No**
Denmark	Yes	Yes
Finland	Yes	Yes
France	Yes	Yes
Germany	Yes	Yes
Hong Kong	Yes	Yes
Hungary	Yes	Yes
India	Yes	Yes
Ireland	Yes	Yes

Country	User Can Purchase Apps	Developer Can Sell Apps
Israel	Yes	Yes
Italy	Yes	Yes
Japan	Yes	Yes
Mexico	Yes	Yes
Netherlands	Yes	Yes
New Zealand	Yes	Yes
Norway	Yes	Yes
Pakistan	Yes	**No**
Poland	Yes	**No**
Portugal	Yes	Yes
Russia	Yes	Yes
Singapore	Yes	Yes
South Korea	Yes	Yes
Spain	Yes	Yes
Sweden	Yes	Yes
Switzerland	Yes	Yes
Taiwan	Yes	Yes
United Kingdom	Yes	Yes
United States	Yes	Yes

Users get access to the market after setting up a Google account. Applications can be bought only via credit card at the moment. Buyers can decide to return an application within 15 minutes from the time of purchasing it and will receive a full refund. Previously, the refund time window was 24 hours. The recent change to 15 minutes has not been well received by end users.

Developers need to register an Android Developer account with Google for a one-time fee of $25 in order to be able to publish applications on the market. After successful

registration, a developer can immediately start to publish a new application in a matter of minutes.

The Android Market has no approval process but relies on a permission system. A user is presented with a set of permissions needed by an application before the installation of the program. These permissions handle access to phone services, networking access, access to the Secure Digital (SD) card, and so on. Only after a user has approved these permissions is the application installed. The system relies on the user doing the right thing. On the PC, especially on Windows systems, this concept didn't work out too well. On Android, it seems to have worked so far; only a few of applications have been pulled from the market because of malicious behavior.

To sell applications, a developer has to additionally register a Google Checkout Merchant Account, which is free of charge. All financial business is handled through this account.

Challenges, Device Seeding, and Google I/O

In an ongoing effort to draw more developers to the Android platform, Google started to hold challenges. The first challenge, called the Android Developer Challenge (ADC) was launched in 2008, offering relatively high cash prices for the winning projects. The ADC was carried out in the subsequent year and was again a huge success in terms of developer participation. There was no ADC in 2010, which can probably be attributed to Android now having a considerable developer base and thus not needing any further actions to get new developers on board.

Google also started a device-seeding program in early 2010. Each developer who had one or more applications on the market with more than 5,000 downloads and an average user rating of 3.5 stars or above received a brand new Motorola Droid, Motorola Milestone, or Nexus One phone. This was a very well-received action within the developer community, although it was initially met with disbelief. Many considered the e-mail notifications that came out of the blue to be an elaborate hoax. Fortunately, the promotion turned out to be a reality, and thousands of devices were sent to developers across the planet—a great move by Google to keep its third-party developers happy and make them stick with the platform and to potentially attract new developers.

Google also provides the special Android Dev Phone (ADP) for developers. The first ADP was a version of the T-Mobile G1 (also known as HTC Dream). The next iteration, called ADP 2, was a variation of the HTC Magic. Google also released its own phone in the form of the Nexus One, available to end users. Although initially not released as an ADP, it was considered by many as the successor to the ADP 2. Google eventually stopped selling the Nexus One to end users, and it is now available for shipment only to partners and developers. At the end of 2010, the latest ADP was released; this Samsung device running Android 2.3 (Gingerbread) is called the Nexus S. ADPs can be bought via the Android Market, which requires you to have a developer account. The Nexus S can be bought via a separate Google site at `www.google.com/phone`.

The annual Google I/O conference is an event every Android developer looks forward to each year. At Google I/O, the latest and greatest Google technologies and projects are revealed, among which Android has gained a special place in recent years. Google I/O usually features multiple sessions on Android-related topics, which are also available as videos on YouTube's Google Developers channel.

Android's Features and Architecture

Android is not just another Linux distribution for mobile devices. While you develop for Android, you're not all that likely to meet the Linux kernel itself. The developer-facing side of Android is a platform that abstracts away the underlying Linux kernel and is programmed via Java. From a high-level view, Android possesses several nice features:

- An *application framework* providing a rich set of APIs to create various types of applications. It also allows the reuse and replacement of components provided by the platform and third-party applications.
- The *Dalvik virtual machine*, which is responsible for running applications on Android.
- A set of *graphics libraries* for 2D and 3D programming.
- *Media support* for common audio, video, and image formats such as Ogg Vorbis, MP3, MPEG-4, H.264, and PNG. There's even a specialized API for playing back sound effects, which will come in handy in our game development adventures.
- *APIs for accessing peripherals* such as the camera, Global Positioning System (GPS), compass, accelerometer, touch screen, trackball, and keyboard. Note that not all Android devices have all of these peripherals—hardware fragmentation in action.

There's of course a lot more to Android than the few features I just mentioned. For our game development needs, these features are the most relevant, though.

Android's architecture is composed of a stack of components, and each component builds on the components in the layer below it. Figure 1–1 gives an overview of Android's major components.

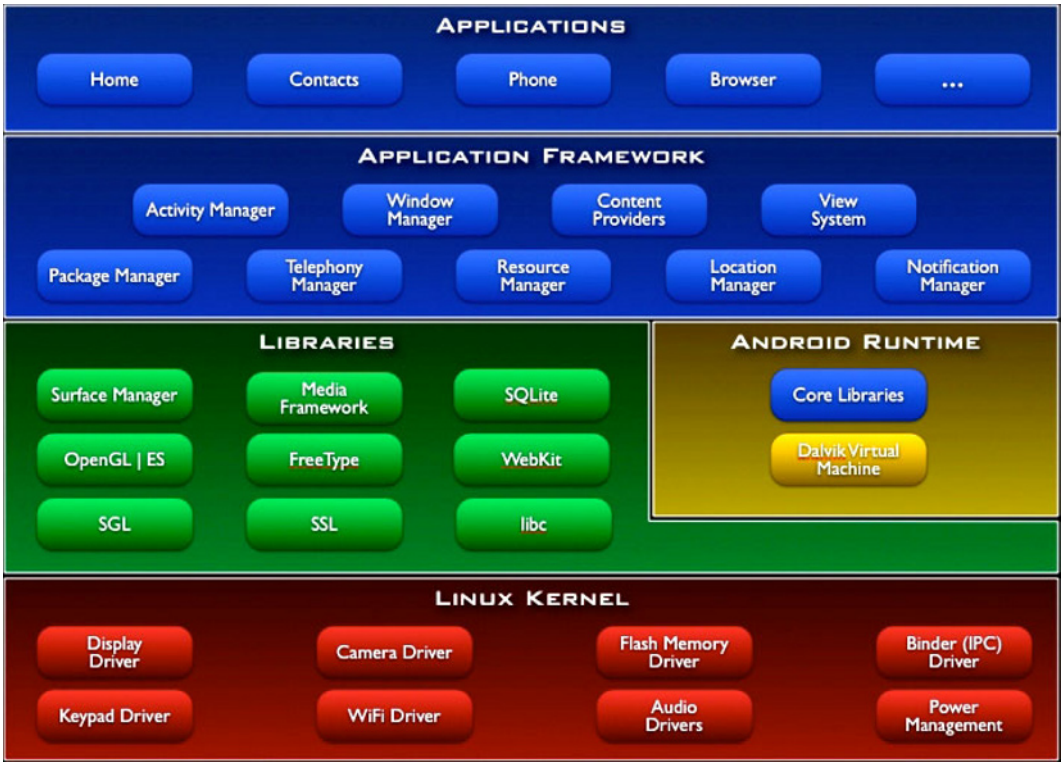

Figure 1–1. *Android architecture overview*

The Kernel

Starting from the bottom of the stack, you can see that the Linux kernel provides the basic drivers for the hardware components. Additionally, the kernel is responsible for such mundane things as memory and process management, networking, and so on.

The Runtime and Dalvik

The Android runtime is built on top of the kernel and is responsible for spawning and running Android applications. Each Android application is run in its own process with its own Dalvik virtual machine.

Dalvik runs programs in the DEX bytecode format. Usually you transform common Java `.class` files to the DEX format via a special tool called *dx* that is provided by the software development kit. The DEX format is designed to have a smaller memory footprint compared to classic Java `.class` files. This is achieved by heavy compression, tables, and merging of multiple `.class` files.

The Dalvik virtual machine interfaces with the core libraries, which provide the basic functionality exposed to Java programs. The core libraries provide some but not all of

the classes available in Java SE through the use of a subset of the Apache Harmony Java implementation. This also means that there's no Swing or Abstract Window Toolkit (AWT) available, nor any classes that can be found in Java ME. However, with some care, you can still use many of the third-party libraries available for Java SE on Dalvik.

Before Android 2.2 (Froyo), all bytecode was interpreted. Froyo introduces a tracing JIT compiler, which compiles parts of the bytecode to machine code on the fly. This increases the performance of computationally intensive applications considerably. The JIT compiler can use CPU features specifically tailored for special computations such as a dedicated Floating Point Unit (FPU).

Dalvik also has an integrated garbage collector (GC). It's a mark-and-sweep nongenerational GC that has the tendency to drive developers a tad bit mad at times. With some attention to details, you can peacefully coexist with the GC in your day-to-day game development, though. The latest Android release (2.3) has an improved concurrent GC, which relieves some of the pain. We'll investigate GC issues in more detail later in the book.

Each application running in an instance of the Dalvik VM has a total of 16MB to 24MB of heap memory available. We'll have to keep that in mind as we juggle our image and audio resources.

System Libraries

Besides the core libraries, which provide some Java SE functionality, there's also a set of native C/C++ libraries that build the basis for the application framework (located in the next layer of Figure 1–1). These system libraries are mostly responsible for the computationally heavy tasks such as graphics rendering, audio playback, and database access, which would not be so well suited for the Dalvik virtual machine. The APIs are wrapped via Java classes in the application framework, which we'll exploit when we start writing our games. We'll abuse the following libraries in one form or another:

Skia Graphics Library (Skia): This software renderer for 2D graphics is used for rendering the UI of Android applications. We'll use it to draw our first 2D game.

OpenGL for Embedded Systems (OpenGL ES): This is the industry standard for hardware-accelerated graphics rendering. OpenGL ES 1.0 and 1.1 are exposed in Java on all versions of Android. OpenGL ES 2.0, which brings shaders to the table, is supported from only Android 2.2 (Froyo) onward. It should be mentioned that the Java bindings for OpenGL ES 2.0 are incomplete and lack a few vital methods. Also, the emulator and most of the older devices that still make up a considerable share of the market do not support OpenGL ES 2.0. We'll be concerned with OpenGL ES 1.0 and 1.1, to stay compatible as much as possible.

OpenCore: This is a media playback and recording library for audio and video. It supports a good mix of formats such as Ogg Vorbis, MP3, H.264, MPEG-4 and so on. We'll be mostly concerned with the audio portion, which is not directly exposed to the Java side but wrapped in a couple of classes and services.

FreeType: This is a library to load and render bitmap and vector fonts, most notably the TrueType format. FreeType supports the Unicode standard, including right-to-left glyph rendering for Arabic and similar peculiarities. Sadly, this is not entirely true for the Java side, which to this point does not support Arabic typography. As with OpenCore, FreeType is not directly exposed to the Java side but is wrapped in a couple of convenient classes.

These system libraries cover a lot of ground for game developers and perform most of the heavy lifting for us. They are the reason why we can write our games in plain old Java.

Note: Although the capabilities of Dalvik are usually more than sufficient for our purposes, at times you might need more performance. This can be the case for very complex physics simulations or heavy 3D calculations—for which we would usually resort to writing native code. I do not cover this aspect in this book. A couple of open source libraries for Android already exist that can help you stay on the Java side of things. See `http://code.google.com/p/libgdx/` for an example. Also worth noting is the excellent book *Pro Android Games* by Vladimir Silva (Apress, 2009), which goes into depth about interfacing with native code in the context of game programming.

The Application Framework

The application framework ties together the system libraries and the runtime, creating the user side of Android. The framework manages applications and provides an elaborate framework within which applications operate. Developers create applications for this framework via a set of Java APIs that cover such areas as UI programming, background services, notifications, resource management, peripheral access, and so on. All core applications provided out of the box by Android, such as the mail client, are written with these APIs.

Applications, whether they are UIs or background services, can communicate their capabilities to other applications. This communication enables an application to reuse components of other applications. A simple example is an application that needs to take a photo and then perform some operations on it. The application queries the system for a component of another application that provides this service. The first application can then reuse the component (for example, a built-in camera application or photo gallery). This significantly lowers the burden on programmers and also enables you to customize a plethora of aspects of Android's behavior.

As game developers, we will create UI applications within this framework. As such, we will be interested in an application's architecture and life cycle as well as its interactions with the user. Background services usually play a small role in game development, which is why I will not go into details about them.

The Software Development Kit

To develop applications for Android, we will use the Android software development kit (SDK). The SDK is composed of a comprehensive set of tools, documentation, tutorials, and samples that will help you get started in no time. Also included are the Java libraries needed to create applications for Android. These contain the APIs of the application framework. All major desktop operating systems are supported as development environments.

Prominent features of the SDK are as follows:

- The *debugger*, capable of debugging applications running on a device or in the emulator
- A *memory and performance profile* to help you find memory leaks and identify slow code
- The *device emulator*, based on QEMU (an open source virtual machine to simulate different hardware platforms), which, although accurate, can be a bit slow at times
- *Command-line utilities* to communicate with devices
- *Build scripts* and tools to package and deploy applications

The SDK can be integrated with Eclipse, a popular and feature-rich open source Java integrated development environment (IDE). The integration is achieved through the Android Development Tools (ADT) plug-in, which adds a set of new capabilities to Eclipse to create Android projects; to execute, profile and debug applications in the emulator or on a device; and to package Android applications for their deployment to the Android Market. Note that the SDK can also be integrated into other IDEs such as NetBeans. There is, however, no official support for this.

> **NOTE:** Chapter 2 covers how to set up the development environment with the SDK and Eclipse.

The SDK and the ADT plug-in for Eclipse receive constant updates that add new features and capabilities. It's therefore a good idea to keep them updated.

Alongside any good SDK comes extensive documentation. Android's SDK does not fall short in this area and comes with a lot of sample applications. You can also find a developer guide and a full API reference for all the modules of the application framework at `http://developer.android.com/guide/index.html`.

The Developer Community

Part of the success of Android is its developer community, which gathers in various places around the Web. The most frequented site for developer exchange is the Android Developers group at `http://groups.google.com/group/android-developers`. This is the number one place to ask questions or seek help when you stumble across a seemingly unsolvable problem. The group is visited by all sorts of Android developers, from system programmers, to application developers, to game programmers. Occasionally, the Google engineers responsible for parts of Android also help out with valuable insights. Registration is free, and I highly recommend starting reading the group now! Apart from providing a place for you to ask questions, it's also a great place to search for already answered questions and solutions to problems. So, before asking a question, check whether it has been answered already.

Every developer community worth its salt has a mascot. Linux has Tux the penguin, GNU has its, well, gnu, and Mozilla Firefox has its trendy Web 2.0 fox. Android is no different and has selected a little green robot as its mascot of choice. Figure 1–2 shows you that little devil.

Figure 1–2. *Android's nameless mascot*

Although its choice of color may be disputable, this nameless little robot already starred in a couple of popular Android games. Its most notable appearance was in Replica Island, a free and open source platfom created by Google engineer Chris Pruett as a 20 percent project.

Devices, Devices, Devices!

Android is not locked into a single hardware ecosystem. Many prominent handset manufacturers such asHTC, Motorola, and Samsung have jumped onto the Android

wagon and offer a wide range of devices running Android. Besides handsets, there's also a slew of tablet devices coming to the market that build upon Android. Some key concepts are shared by all devices, though, which makes our lives as game developers a little easier.

Hardware

There are no hard minimum requirements for an Android device. However, Google has recommended the following hardware specifications, which virtually all available Android devices fulfill and most often surpass significantly:

ARM-based CPU: At the time of writing this book, this requirement was relaxed. Android now also runs on the x86 architecture. The latest ARM-based devices are also starting to feature dual-core CPUs.

128MB RAM: This specification is a minimum. Current high-end devices already include 512MB RAM, and 1GB RAM devices are expected in the very near future.

256MB flash memory: This minimum amount of memory is for storing the system image and applications. For a long time, this lack of memory was the biggest gripe among Android users because third-party applications could be installed only to flash memory. This changed with the release of Froyo.

Mini or Micro SD card storage: Most devices come with a few gigabytes of SD card storage, which can be replaced with bigger SD cards by the user.

16-bit color Half-Size Video Graphics Array (HVGA) TFT LCD with touch screen: Before Android version 1.6, only HVGA screens (480×320 pixels) were supported by the operating system. Since version 1.6, lower- and higher-resolution screens are supported. The current high-end devices have Wide Video Graphis Array (WVGA) screens (800×480, 848×480, or 852×480 pixels), and some low-end devices sport Quarter-Size Video Graphics Array (QVGA) (320×280 pixels) screens. Touch screens are almost always capacitive and are only single-touch capable on most older devices.

Dedicated hardware keys: These keys are used for navigation. Most phones to date have at least a menu, search, home, and a back key. Some manufacturers have started to deviate from this and are including a subset of these keys or no keys at all.

Of course, there's a lot more hardware in actual Android devices. Almost all handsets have *GPS*, an *accelerometer*, and a *compass*. Many also feature *proximity and light sensors*. These peripherals offer game developers new ways to let the user interact – with the game, and we'll have a look at some of them later on. A few devices have a full *QWERTY keyboard* as well as a *trackball.* The latter is most often found in HTC devices.

Cameras are also available on almost all current devices. Some handsets and tablets have two cameras, one on the back and one on the front for video chat.

Especially crucial for game development are dedicated *graphics processor units (GPUs)*. The earliest handset to run Android already had an OpenGL ES 1.0compliant GPU. More-modern devices have GPUs comparable in performance to the Xbox or PlayStation 2 and support OpenGL ES 2.0. If no graphics processor is available, a fallback in the form of a software renderer called PixelFlinger is provided by the platform. Many low-budget handsets rely on the software renderer, which is often sufficiently fast for low-resolution screens.

Along with the graphics processor, any currently available Android device also has dedicated *audio hardware*. Many hardware platforms also have special circuitry to decode different media formats such as H.264 in hardware. Connectivity is provided via hardware components for mobile telephony, Wi-Fi, and Bluetooth. All these hardware modules of an Android device are most often integrated in a single *system on a chip (SoC)*, a system design also found in embedded hardware.

First Gen, Second Gen, Next Gen

Given the differences in capabilities, especially in terms of performance, Android developers usually group devices into first-, second-, and next-generation devices. This terminology comes up a lot, even more so when it comes to game development for Android. Let's try to define these terms.

Each generation has a specific set of characteristics, mostly a combination of the Android version(s) used, the CPU/GPU, and the screen resolution of the devices within a generation. Although the hardware specifications are static, this might not be the case for the Android version used on a device.

In the Beginning: First Generation

First-generation devices are the current baseline and are best described by examining one of their most prominent specimens, the HTC Hero, shown in Figure 1–3.

Figure 1–3. *The HTC Hero*

This was one of the first Android phones that was said to be an iPhone killer, released in October 2009. The Hero was first shipped with Android version 1.5 installed, which was the standard for most Android handsets for most of 2009. The last official update for the Hero was to Android version 2.1. Newer updates can be installed only if the phone is *rooted*, a process that grants full system access.

The Hero has a 3.2-inch HVGA capacitive LCD touch screen, a 528MHz Qualcomm MSM7201A CPU/GPU combination, an accelerometer, and a compass, as well as a 5-megapixel camera. It also has the typical set of navigational hardware keys that most first-generation devices exhibit, along with a trackball.

The Hero is a prime example of first-generation devices. The touch screen has only limited support for multi-touch gestures such as the pinch zoom and no true multi-touch capability. Note that multi-touch gestures are not officially supported by the device and are also not exposed through the APIs of the official Android version 1.5. In this regard, the Hero was a major diasppointment for game developers who had hoped for similar multi-touch capabilities as those found on the iPhone.

Another common trait of first-generation devices is the screen resolution of 480×320 pixels, the standard resolution up until Android version 1.6.

In the CPU/GPU department, the Hero employs the very common MSM7201A series by Qualcomm. This chip does not support hardware floating-point operations, another feature of high importance to game developers. The MSM7201A is OpenGL ES 1.0 compliant, which translates to a fixed-function pipeline as opposed to a programmable,

shader-based pipeline. The GPU is reasonably fast but outperformed by the PowerVR MBX Lite chip found in the iPhone 3G, which was available at the same time. HTC used the same chip in a couple of other first-generation handsets, such as the famous HTC Dream (T-Mobile G1). The MSM7201A is considered the low end when it comes to hardware-accelerated 3D graphics and is thus your greatest enemy when you want to target all generations of Android devices.

First-generation devices can thus be identified by the following features:

- A CPU running at up to ~500MHz without hardware floating-point support
- A GPU, mostly in the form of the MSM7201A chip, supporting OpenGL ES 1.*x*
- A screen resolution of 480×320 pixels
- Limited multi-touch support
- Initially deployed with Android 1.5/1.6 or even earlier versions

This classification is of course not strict. Many low-budget devices just coming out share a similar feature set. Although they are not exactly first generation, we can still put them in the same category as the Hero and similar devices.

First-generation devices still have a considerable market share at the time of writing this book. If we want to reach the biggest possible audience, we have to consider their limitations and adapt our games accordingly.

More Power: Second Generation

At the end of 2009, a new generation of Android devices entered the scene. Spearheaded by the Motorola Droid and Nexus One (released in January 2010), this new generation of handsets demonstrated raw computational power previously unseen in mobile phones.

The Nexus One is powered by a 1GHz Qualcomm QSD8250, a member of the Snapdragon family of chips. The Motorola Droid uses a 550MHz Texas Instruments OMAP3430. Both CPUs support vector hardware floating-point operations via the Vector Floating Point (VFP) and NEON ARM extensions. The Nexus One has 512MB RAM, and the Motorola Droid has 256MB RAM. Figure 1–4 shows their designs.

Figure 1–4. *The Nexus One and Motorola Droid*

Both phones have a WVGA screen, an 800×480 pixel Active-Matrix Organic Light-Emitting Diode (AMOLED) screen (in the case of the Nexus One) or a 854×480 pixel LCD screen (in the case of the Motorola Droid). Both screens are capacitive multi-touch screens. Although both devices were advertised as multi-touch capable, they do not work as expected in a couple of situations. The most common problem is the reporting of false touch positions when two fingers are close on either the x- or y-axis on the screen.

The Nexus One was first shipped with Android version 2.1, and the Motorola Droid was shipped with version 2.0. Both phones have received updates to Android version 2.2.

Of special interest to game developers are the built-in GPUs. The PowerVR SGX530 is a very potent GPU also used in the iPhone 3GS. Note that the screen size of the iPhone 3GS is actually half that of the Motorola Droid, which gives the iPhone 3GS a slight performance advantage, because it has to draw fewer pixels per frame. The Adreno 200 chip used in the Nexus One is a Qualcomm product and slightly slower than the PowerVR SGX530. Depending on the rendered scene, both chips can be nearly a magnitude faster than the MSM7201A found in many first-generation devices.

Second-generation devices can be identified by the following features:

- A CPU running between 550MHz and 1GHz with hardware floating-point support
- A programmable GPU supporting OpenGL ES 1.*x* and 2.0
- A WVGA screen
- Multi-touch support
- Android version 2.0, 2.0.1, 2.1, or 2.2

Note that a few first-generation devices received updates to Android version 2.1, which has some positive impact on overall system performance but does not, of course, change the fact that their hardware specifications are inferior to second-generation devices. The distinction between first- and second-generation devices can thus be made only if all factors such as CPU, GPU, or screen resolution are taken into account.

Over the course of 2010, many more second-generation devices appeared, such as the HTC Evo or the Samsung i9200 Galaxy S. Although they feature some improvements over the Nexus One and Motorola Droid such as bigger screens and slightly faster CPUs/GPUs, they are still considered second-generation devices.

The Future: Next Generation

Device manufacturers try to keep their latest and greatest handsets a secret for as long as possible, but there are always some leaks of specifications.

General trends for all future devices are dual-core CPUs, more RAM, better GPUs, and higher screen resolutions. One such future device is the Samsung i9200 Galaxy S2, which is rumored to have a 1280×720 pixel AMOLED 2 display, a 2GHz dual-core CPU, and 1GB RAM. Not much is known about the GPU this handset will use. A possible candidate would be the new NVIDIA Tegra 2 family of chips, which promises a significant boost in graphics performance. The next generation is also expected to ship with the latest Android version (2.3).

Although mobile phones will probably remain the focus of Android for the immediate future, new form factors will also play a role in Android's evolution. Hardware manufacturers are creating tablet devices and netbooks, using Android as the operating system. Ports of Android for other architectures such as x86 are also already in the making, increasing the number of potential target platforms. And with Android 3.0, there's even a dedicated Android version for tablets available.

Whatever the future will bring, Android is here to stay!

Game Controllers

Given the differences of input methods available on various Android handsets, a few manufacturers produce special game controllers. Because there's no API in Android for such controllers, game developers have to integrate support separately by using the SDK provided by the game controller manufacturer.

One such game controller is called the Zeemote JS1, shown in Figure 1–5. It features an analog stick as well as a set of buttons.

Figure 1–5. *The Zeemote JS1 controller*

The controller is coupled with the device via Bluetooth. Game developers integrate support for the controller via a separate API provided by the Zeemote SDK. A couple of Android games already support this controller when available.

Users could in theory also couple the Nintendo Wii controller with their device via Bluetooth. A couple of prototypes exploiting the Wii controller exist, but there's no officially supported SDK—which makes integration a tad bit awkward.

The Game Gripper, shown in Figure 1–6, is an ingenious invention specifically designed for the Motorola Droid and Milestone. It is a simple rubber accessory that slides over the QWERTY keyboard of the phone and overlays a more or less standard game controller layout on top of the actual hardware keyboard. Game developers need only add keyboard controls to their game and don't have to integrate a special library to communicate with the Gripper. It's just a piece of rubber, after all.

Figure 1–6. *The Game Gripper in action*

Game controllers are still a bit esoteric in the realm of Android. However, some successful titles have integrated support for some controllers, a move generally well received by Android gamers. Integrating support for such peripherals should therefore be considered.

Mobile Gaming Is Different

Gaming was already huge way before the likes of the iPhone and Android started to conquer this market segment. However, with those new forms of hybrid devices, the landscape has started to change. Gaming is no longer something for nerdy kids. Serious businesspeople have been caught playing the latest trendy game on their mobile phones in public, newspapers pick up stories of successful small game developers making a fortune on mobile phone application markets, and established game publishers have a hard time keeping up with the developments in the mobile space. We game developers must recognize this change and adjust accordingly. Let's see what this new ecosystem has to offer.

A Gaming Machine in Every Pocket

Smartphones are ubiquitous. That's probably the key statement to take away from this section. From this, we can easily derive all the other facts about mobile gaming.

As hardware prices are constantly dropping and new cell phones have ever-increasing computational power, they also become ideal gaming devices. Mobile phones are a must-have nowadays, so market penetration is huge. Many people who are exchanging their old, classic mobile phones with the new generation of smartphones are discovering the new options available to them in the form of an incredibly wide range of applications.

Previously, people had to make the conscious decision to buy a video game system or a gaming PC in order to play video games. Now they get that functionality for free from their mobile phones. There's no additional cost involved (at least if you don't count the data plan you'll likely have), and your new gaming device is available to you at any time. Just grab it from your pocket or purse, and you are ready to go—no need to carry a second dedicated system with you, because everything's integrated in one package.

Apart from the benefit of having to carry only a single device for your telephony, Internet, and gaming needs, another factor makes gaming on mobile phones incredibly accessible to a much larger audience: you can fire up a dedicated market application on your phone, pick a game that looks interesting, and immediately start to play. There's no need to go to a store or download something via your PC only to find out, for example, that you lost the USB cable needed to transfer that game to your phone.

The increased processing power of current-generation smartphones also has an impact on what's possible for us as game developers. Even the middle class of devices is capable of generating gaming experiences similar to titles found on the older Xbox and PlayStation 2 systems. Given these capable hardware platforms, we can also start experimenting with more-elaborate games with physics simulations, an area offering great potential for innovation.

With new devices also come new input methods, which we have already discussed a little. A couple of games already exploit the GPS and/or compass available in most Android devices. The use of the accelerometer is already a mandatory feature of most games, and multi-touch screens offer new ways for the user to interact with the game world. Compared to classic gaming consoles (and ignoring the Wii for the moment), this is quite a change for game developers. A lot of ground has been covered already, but there are still new ways to use all this functionality in an innovative way.

Always Connected

Smartphones are usually bought along with data plans. They are not only used for pure telephony anymore but actually drive a lot of traffic to popular Internet sites. A user having a smartphone is very likely to be connected to the Web at any point in time (neglecting for a moment poor reception, for example, caused by hardware design failures).

Permanent connectivity opens up a completely new world for mobile gaming. People can challenge other people across the planet for a quick match of chess, explore virtual worlds together, or try fragging their best friend in another city in a fine death match of gentlemen. And all of this occurs on the go, on the bus or train or in their most beloved corner of the local park.

Apart from multiplayer functionality, social networks have also started to play a huge role in mobile gaming. Games provide functionality to tweet your latest high score directly to your Twitter account or to inform a friend of your latest achievements earned in that racing game you both love. Although growing social networks exist in the classical gaming world (for example, Xbox Live or the equivalent PlayStation service),

the market penetration of services such as Facebook and Twitter is a lot higher, and so the user is relieved of the burden of managing multiple networks at once.

Casual and Hardcore

The huge user adaption of smartphones also means that people who have never even touched a NES controller suddenly discover the world of gaming. Their mental image of a good game often deviates quite a bit from the one a hardcore gamer might have.

Given the use cases for mobile phones, users tend to lean toward the more casual sort of games that they can fire up for a couple of minutes while on the bus or waiting in line at their preferred fast food restaurant. These games are equivalent toall those small flash games on the PC that are forcing many people in the workforce to Alt+Tab frantically each time they sense the presence of someone watching their back. Ask yourself this: how much time would you be willing to spend playing games on your mobile phone? Can you imagine playing a "quick" game of Civilization on such a device?

Surely there are people who would actually offer their firstborn if only they could play their beloved Advanced Dungeons & Dragons variant on a mobile phone. But this group is a small minority, as evidenced by the top-selling games on the iPhone and Android Markets. The top-selling games are usually extremely casual but have a nice trick under their sleeves: The average time taken to play a round of such a game is in the range of minutes, but the games make you come back by employing various evil schemes. The game might provide an elaborate online achievement system that lets you virtually brag about your skills. But it could also be an actual hardcore game in disguise. Offer users an easy way to save their progress, and you are set to sell them your hardcore game as a casual game!

Big Market, Small Developers

The low entry barrier is a main attractor for many hobbyists and independent developers. In the case of Android, this barrier is especially low: just get yourself the SDK and program away. You don't even need a device, just use the emulator (although I highly recommend having at least one development device). The open nature of Android also leads to a lot of activity on the Web. Information on all aspects of programming for the system can be found for free online. There's no need to sign an Non-Disclosure Agreement or wait for some authority to grant you access to their holy ecosystem.

At the time of this writing, the most successful games on the market were developed by one-person companies and small teams. Major publishers have not yet set foot in the market, at least not successfully. Gameloft serves as a prime example. Although big on the iPhone, Gameloft couldn't get a hold of the Android market and decided to sell their games on their own website instead. Gameloft might not have been happy with the missing Digital Rights Managment scheme (which is available on Android now)—a move that of course lowers the number of people who actually know about their games considerably.

The environment also allows for a lot of experimentation and innovation as bored people surfing the market are longing for little gems, including new ideas and game play mechanics. Experimentation on classic gaming platforms such as the PC or consoles are often met with failure. However, the Android Market enables you to reach a much larger audience that is willing to try experimental new ideas, and to reach them with a lot less effort.

This doesn't mean, of course, that you don't have to market your game. One way to do so is to inform various blogs and dedicated sites on the Web about your latest game. Many Android users are enthusiasts and regularly frequent such sites, checking in on the latest and greatest.

Another way to reach a large audience is to get featured in the Android Market. Once featured, your application will appear to users in a list immediately after they start the market application. Many developers have reported a tremendous increase in downloads directly correlated to getting featured on the market. How to get featured is a bit of a mystery, though. Having an awesome idea and executing it in the most polished way is your best bet, whether you are a big publisher or a small one-person shop.

Summary

Android is an exciting little beast. You have seen what it's made of and have gotten to know its developer ecosystem a little. It offers us a very interesting system in terms of software and hardware to develop for, and the barrier of entry is extremely low given the freely available SDK. The devices themselves are pretty powerful for handheld devices and will enable us to present visually rich gaming worlds to our users. The use of sensors such as the accelerometer let us create innovative game ideas with new user interactions. And after we have finished developing our games, we can deploy them to millions of potential gamers in a matter of minutes. Sounds exciting? Then let's get our hands dirty with some code!