Covers Android 5.0

Pro Android Games

THIRD EDITION

Massimo Nardone | Vladimir Silva

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

Introduction

Welcome to *Pro Android Games*, *Third Edition*. This book will help you create great games for the Android platform. You can find plenty of books out there that tackle this subject, but this book gives you a unique perspective by showing you how easy it is to bring native PC games to the platform with minimum effort.

To get the most out of this book, you must have a solid foundation in Java and ANSI C. However, even the most complicated concepts have been explained as clearly and as simply as possible using a combination of graphics and sample code. The real-world examples and source code provided for each chapter will help you understand the concepts in detail so you can excel as a mobile game developer.

The Green Robot Has Taken Over

It is hard to believe that only a few years have passed since Android came into the smartphone scene—and it has taken off with a vengeance. Take a look at the worldwide Smartphone OS Market Share[1] shown in Figure 1. In Q2 2014 Android lead with 84.7% and the stats just keep getting better and better. This opens a new frontier for developers looking to capitalize on the rocketing smartphone segment. This book will help you quickly build cutting-edge games for the Android platform.

[1]"IDC: Android has a heady 84.7% percent of world smartphone share," http://www.idc.com/prodserv/smartphone-os-market-share.jsp

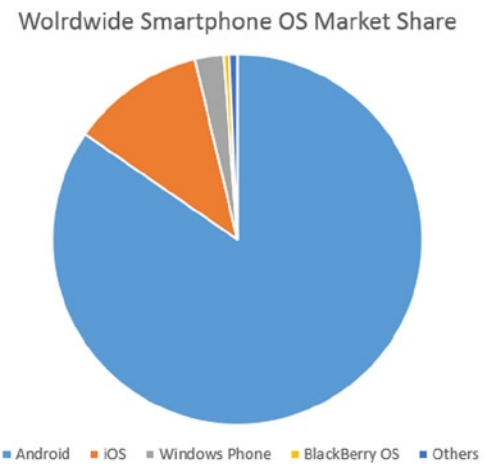

Figure 1. *Worldwide smartphone market share 2014*

Target Audience

This book targets seasoned game developers, not only in Java, but also in C. This makes sense, as performance is critical in game development. Other audiences include:

- *Business apps developers*: This book can be a valuable tool for business developers, especially those working on native applications.
- *Scientific developers*: In the science world, raw performance matters. The chapters dealing with JNI and OpenGL can help you achieve your goals.
- *Computer science students learning new mobile platforms*: Android is open and fairly portable, thus this book can be helpful for working on many platforms (iPhone, BlackBerry, Meego, and others).
- *Anyone interested in Android development*: Android has taken over the mobile market space at a furious pace. You must expand your skill set to include games and graphics or you may be left behind.

Needed Skills to Make the Most of This Book

The required skill set for Pro Android Games includes C/C++ and Java, plus some basic Linux shell scripting. Java provides elegant object-oriented capabilities, but only C gives you the power boost that game development needs.

A Solid Foundation of Android

This book assumes that you already know the basics of Android development. For example, you need to understand activities, views, and layouts. Consider the following fragment. If you understand what it does just by looking at it, then you are in good shape.

```
public class MainActivity extends Activity
{
public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);
setContentView(R.layout.main);
}
}
```

This fragment defines the main activity or class that controls the life cycle of the application. The `onCreate` method will be called once when the application starts, and its job is to set the content layout or GUI for the application.

You should also have a basic understanding of how GUIs are created using XML. Look at the next fragment. Can you tell what it does?

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent" android:layout_height="fill_parent">


<ImageView android:id="@+id/doom_iv" android:layout_width="fill_parent"
android:layout_height="fill_parent" android:background="@drawable/doom"
android:focusableInTouchMode="true" android:focusable="true"/>

<ImageButton android:id="@+id/btn_upleft" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:layout_alignParentBottom="true"
android:layout_alignParentLeft="true" android:src="@drawable/img1" />
</RelativeLayout>
```

This code defines a relative layout. In a relative layout, widgets are placed relative to each other (sometimes overlapping). In this case, there is an image view that fills the entire screen. This image displays as the background the file called `doom.png` stored in the `res/drawable` folder of the project, and it receives key and touch events. In the lower left of the screen, overlapping the image view, an image button with the ID `btn_upleft` is displayed.

Android development requires understanding of many concepts, and it is impossible to remember every detail about activities, views, and layouts. A handy place to access this information quickly is the Android tutorial at http://developer.android.com/.

The ultimate guide for Android developers—the latest releases, downloads, SDK Quick Start, version notes, native development tools, and previous releases—can be found at http://developer.android.com/sdk/index.html.

Throughout this book (especially in the chapters dealing with native code), I make extensive use of the Android software development kit (SDK) command tools (for system administrator tasks). Thus, you should have a clear understanding of these tools, especially the Android Debug Bridge (ADB). You should know how to implement the following tasks:

- *Create an Android Virtual Device (AVD)*: An AVD encapsulates settings for a specific device configuration, such as firmware version and SD card path. Creating an AVD is really simple and can be done from the integrated development environment (IDE) by using the AVD Manager (accessed by clicking the black phone icon in the toolbar).
- *Create an SD card file*: Some of the games in later chapters have big files (5MB or more). To save space, the code stores all game files in the device SD card, and you should know how to create one. For example, to create a 100MB SD card file called sdcard.iso in your home directory, use this command:

  ```
  $ mksdcard 100M $HOME/sdcard.iso
  ```

- *Connect to the emulator*: You need to do this for miscellaneous system administration, such as library extraction. To open a shell to the device, use this command:

  ```
  $ adb shell
  ```

- *Upload and pull files from the emulator*: These tasks are helpful for storing and extracting game files to and from the device. Use these commands:

  ```
  $ adb push <LOCAL_FILE> <DEVICE_FILE>
  $ adb pull <DEVICE_FILE> <LOCAL_FILE>
  ```

> **Note** Make sure the SDK_HOME/tools directory is added to your system PATH variable before running the commands to create an SD card file, connect to the emulator, or upload and pull files.

A Basic Knowledge of Linux and Shell Scripting

All the chapters in this book (except Chapter 1) use a hybrid combination of Java/C development, which requires knowledge about native development. In these chapters, you do the work within a Linux-like shell dubbed Cygwin, so dust off all those old Unix skills. You should know the basic shell commands, such as those for listing files, installing software components (this can be tricky, depending on your Linux distribution), and basic system administration. There are a few very simple shell scripts in this book. A basic knowledge of the `bash` shell is always helpful.

Tip If you need a refresher on your Linux and shell scripting, check out the following tutorial by Ashley J.S Mills: http://supportweb.cs.bham.ac.uk/documentation/tutorials/docsystem/build/tutorials/unixscripting/unixscripting.html.

Required Hardware and Software

The following sections cover the tools you will need to make the most of this book.

A Windows or Linux PC with a Java SDK, Properly Installed

This requirement is probably obvious, since most development for Android is done in Java. Note that I mentioned a Java SDK, not JRE. The SDK is required because of the JNI header files and command line tools used throughout the latter chapters.

Android Studio, Eclipse ADT, and Android SDK, Properly Installed

Android Studio is the de facto IDE for Android development. I used both the Eclipse Android Development Tools (ADT) and Android Studio with Android SDK for Windows. The reason for using both Eclipse ADT and Android Studio was that when this manuscript was updated Android Studio did not fully support NDK.

Even though Eclipse ADT as well as Android Studio was used to create the code workspace, you can use your favorite IDE. Of course that will require a bit of extra setup. You can get Eclipse from www.eclipse.org/.

For instructions on how to setup the Android SDK with other IDEs go to http://developer.android.com/tools/workflow/index.html. Refer to Chapter 1 for details about configuring the Android SDK in your PC.

Native Development Kit (NDK)

The NDK is the essential tool for any serious game developer out there. It provides the compiler chain, header files, and documentation required to bring your cutting-edge games to the mobile landscape. By using the NDK, developers can escape the shackles of the Java memory heap and unleash their creativity to build the most powerful C/C++ engines, limited only by what the hardware can provide. In *Pro Android Games*, you'll use the NDK extensively, thus a solid foundation of C programming is required to fully understand the concepts presented in each chapter. For this book, the NDK Revision 10d was used. Refer to Chapter 1 for details about setting up the latest NDK in your PC.

Chapter Source

This is an optional tool but it will help you greatly in understanding the concepts as you move along. I have made my best effort to describe each topic as simply as possible; nevertheless some of the games (especially Quake I and II) have very large core engines written in C (100,000 lines for Doom), which are poorly commented and very hard to understand. All in all, you will see how easily these great languages (Java and C) can be combined with minimal effort. You can download the companion source for the book from www.apress.com. It was built using the latest Eclipse SDK, Eclipse ADT, and Android Studio.

What Makes This Book Unique?

Even though Java is the primary development language for Android, Google has realized the need for hybrid Java/C development if Android is to succeed as a gaming platform—so much so that they released the NDK. I think that Google has been wise to support C development; otherwise it would have been overtaken by the overwhelming number of native games written for other mobile platforms such as the iPhone. PC games have been around for decades (mostly written in C); by using a simple ARM C compiler, you can potentially bring thousands of PC games to the Android platform.

This is what makes this book unique: why translate 100,000 lines of painfully complicated code from C to Java if you can just combine both languages in an elegant manner and save yourself lots of time and money in the process? In addition, this book provides plenty of examples about Android implementations such as Android Wear and Android TV, powered by the new SDK 5 version. Finally, this book does include chapters of pure Java games in a well-balanced layout to satisfy both the Java purist and the C lover in you.

What's Changed Since the Last Edition?

With the relentless pace of Android updates, many things have changed since the last iteration of *Pro Android Games*, including the following:

- *Updates*: This book includes the latest version of the Android SDK, NKD, OpenGL ES, the latest Eclipse ADT, and Android Studio.
- *Android Wear*: This edition helps you to understand how to build Android wearable applications to be tested with Android emulator and real Android Wear devices.
- *Android TV*: You'll learn all about how to start building TV apps or extend your existing app to run on TV devices.

Android SDK Compatibility

As a developer, you may wonder about the SDK compatibility of the code in this book, as new versions of the Android SDK come out frequently. At the time of this writing, Google released the Android SDK version 5.0 API 21. The code in this book has been fully tested with the following versions of the Android SDK:

- SDK version 5.0
- SDK version 4.4

The bottom line is that the code in this book will run in any version of the SDK, and that was my intention all along.

Chapter Overview

Here's a summary of what you'll find in *Pro Android Games*, Third Edition.

Chapter 1

This chapter provides the first steps to setting up a Windows system for hybrid game compilation, including

- Fetching the Android source
- Setting up the Eclipse ADT as well as Android Studio for development
- Installing the latest NDK
- Creating an emulator for testing or configuring a real device
- Importing the book's source into your workspace, which is critical for understanding the complex topics

Chapter 2

In this chapter you'll see how to combine Java and C code in an elegant manner by building a simple Java application on top of a native library. You'll learn exciting concepts about the Java Native Interface (JNI) and the API used to combine Java and C in a single unit, including how to load native libraries, how to use the native keywords, how to generate the JNI headers, plus all about method signatures, Java arrays versus C arrays, invoking Java methods, compiling and packing the product, and more.

Chapter 3

This chapter deals with 3D graphics with OpenGL. It presents a neat trick I stumbled upon that allows for mixing OpenGL API calls in both Java and C. This concept is illustrated by using the 3D cubes sample provided by Google to demonstrate OpenGL in pure Java and hybrid modes. This trick could open a new frontier of 3D development for Android with the potential to bring a large number of 3D PC games to the platform with enormous savings in development costs and time.

Chapter 4

This chapter tackles efficient graphics with OpenGL 2.0. It starts with a brief description of the most important features in OpenGL 2, including shaders, GLSL, and how they affect the Android platform. Then it takes a deeper look into GLSL by creating a neat Android project to render an icosahedron using OpenGL ES 2.0. As a bonus, it shows you how you can use single- and multi-touch functionality to alter the rotation speed of the icosahedron, plus pinching for zooming in or out.

Chapter 5

Chapter 5 takes things to the next level with the ground-breaking game for the PC: Doom. Doom is arguably the greatest 3D game ever created, and it opened new frontiers in 3D graphics. The ultimate goal of this chapter is not to describe the game itself, but to show you how easy it is to bring a complex PC game like Doom to the Android platform. The proof? Doom is over 100,000 lines of C code, but it was ported to Android with less than 200 lines of extra JNI API calls plus the Java code required to build the mobile UI. This chapter shows that you don't have to translate thousands of lines of C into Java; you can simply marry these two powerful languages in an elegant application. A must-read chapter!

Chapter 6

Here is where things start to get really exciting! This chapter brings you a first-person shooter (FPS) gem: Quake. You'll see how a powerful PC engine of this caliber can be brought to the Android platform with minimum effort—so much so that 95% of the original C code is kept intact! It only requires an extra 500–1000 lines of new, very simple Java wrapper code. Start playing Quake in all its glory on your smartphone now!

Chapter 7

This chapter builds upon the previous one to deliver the Quake II engine to your fingertips. It is remarkable how the highly complex OpenGL renderer of the Quake II engine can be kept intact thanks to a wonderful tool called NanoGL. NanoGL allows the developer to translate the complexity of the OpenGL immediate mode drawing into OpenGL ES transparently, keeping your original code intact. The chapter also shows how to make the Quake II engine behave properly in Android by creating custom audio and video handlers, at the same time demonstrating the great reusability features of the Java language. All in all, 99% of the original Quake II C code is kept intact, plus the thin Java wrappers of the previous chapter are reused without change.

Chapter 8

This chapter deals with Bluetooth controllers. You know it's difficult to play games (such as first-person shooters) with a touch screen interface or a tiny keyboard. Some games really require a gamepad controller. The hardcore gamer in you will smile at how easy is to integrate two popular gaming controllers, Wiimote and Zeemote, into your game. In the process, you'll also learn about the Bluetooth API, gain an insight into the inner workings of the Wiimote and Zeemote, pick up a little about JNI, asynchronous threads, and more.

Chapter 9

This chapter tackles two interesting new subjects coming soon to your mobile device and living room: augmented reality (AR) and Android TV. The future of casual gaming seems to be headed the AR way. Other platforms like the PS Vita are already taking the first step by providing a solid foundation of AR games out of the box. There is a lot of hype surrounding augmented reality, and Android developers are taking notice. This chapter shows you how to use the popular ARToolkit to build an AR-capable application/game using OpenGL and JNI. This chapter also looks at the rise of smart TVs, specifically Android TV. Android TV is powered by Android 5.0 and thus is fully compatible with the games you may be planning to create.

Chapter 10

This chapter covers deployment and compilation tips for Android applications. In addition to many time-saving goodies, this chapter includes tips for signing your application manually or using Android Studio, creating a key store for signature of your application package, and installing your signed application into a real Android device, as well as the most important tips you should remember when building hybrid games that use both Java and C/C++.

Chapter 11

This chapter introduces you to Android Wear. You'll learn what you need to install and configure to develop your first Wear application using the Android SDK 5.0 API 21.

More specifically you'll see how Android Wear works, the Android libraries needed for a Wear application, how to create a Wear Emulator with Android AVD Manager, and finally, how to create and run your first Wear application. After this chapter you'll find it easier to understand Android Wear apps and implement new ones, now that you know the basics.

Chapter 1

Welcome to Android Gaming

This chapter kicks things off by explaining how to set up your environment to compile hybrid (C/Java) games, including the engines described in the later chapters of the book. It also explains how to set up the latest versions of the integrated development environment (IDE), which is Android Studio 1.0.1 with SDK for Windows, plus the native development kit (NDK). These tools are required to build powerful games for Android. They let you combine the elegant object-oriented features of Java with the raw power of C for maximum performance. The chapter ends by showing how to import the workspace for the game engines included in the source code of this book (which can be obtained at www.apress.com). Let's get started.

Preparing the development environment includes having the following software installed on your desktop:

- *Android IDE*: This is the development IDE used to create your projects. I have used the Android Studio with SDK for Windows in this manuscript.
- *Android SDK (properly configured)*: At the time of this writing, the latest version of the SDK is 5.0.1 (API 21).
- *Java JDK 7*: This is required to run Android Studio with SDK itself (JRE alone is not sufficient).
- Apache Ant 1.8 or later

The next section tells you how to set up your machine step by step.

Setting Up Your Machine

In this book I will install and use Android SDK 5.0.1 for Windows 7.

There are a few steps to be completed before you can get to the nitty-gritty stuff of building games for Android. Follow these steps:

1. The first and most basic thing you need is a current Java JDK 7. Make sure you have the proper version installed before proceeding. This book uses JDK version 8 64-bit.
2. Download and install the Android SDK 5.0.1. The SDK contains the core resources to develop for Android.
3. Configure Android Studio 1.0.1.
4. Install the NDK if you don't have it. This is a critical component for any kind of game that uses native APIS such as OpenGL. At the time of this writing, I used r10c Windows 64-bit. All in all, keep in mind that Android 5.0 is not binary compatible with older NDK versions, which means that if you have an old NDK, you have to update it.
5. Create an emulator. This is an optional step that will help you with testing your games in many API versions and screen sizes.
6. Configure a real device. I prefer to work in a real device because it is so much faster than using an emulator and is the best way to work if you use OpenGL.

Download and Install the SDK

Download the latest version of the Android Studio 1.0.1 for Windows from http://developer.android.com/sdk/index.html#win-bundle and install it (in my case named android-studio-bundle-135.1641136.exe).

> **Note** Some of the code included in this book was developed, compiled and tested using the Eclipse IDE for Java Developers and Android SDK Tools instead of Android Studio 1.0.1 since when this manuscript was written, NDK and JNI were not supported by Android Studio.

Configure Android Studio

The very first step will be to run the Android SDK Manager to update all the needed APIs and libraries.

To run the SDK Manager from Android Studio, select Tools ➤ Android ➤ SDK Manager (see Figure 1-1).

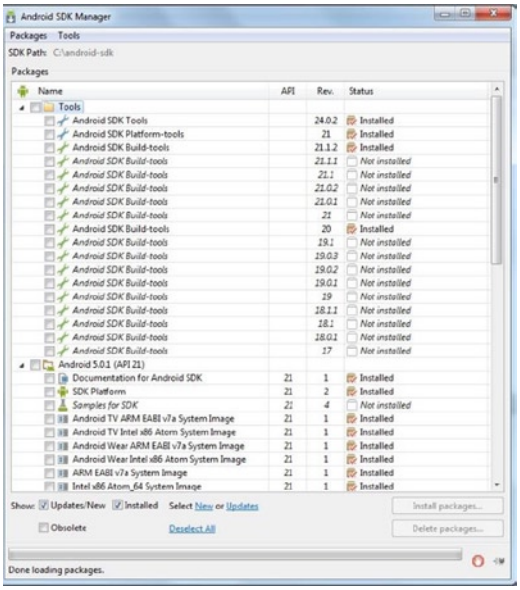

Figure 1-1. *Installing the SDK 5 API and Libraries*

To use Android SDK, you should install as a minimum the latest Android SDK tools and Android platform such as:

- Android SDK Platform-tools (ARM EABI v7a System Image)
- Android SDK Build-tools (highest version)

The Android Support Library provides an extended set of APIs that are compatible with most versions of Android such as:

- Android Support Repository
- Android Support Library

For the examples you also need to install the support library for additional APIs required for:

- Android Wear
- Android TV
- Google Cast

Finally, to develop with Google APIs, you need the Google Play services package such as:

- Google Repository
- Google Play services

Select all the packages just described, click Install packages, and follow the instructions. After installation completes, close the SDK Manager.

You are ready to get your Android Studio up and running with the Android development kit.

Click File ➤ New Project wizard to make sure the Android plug-in has been successfully installed. If so, you should see a folder to create an Android project, as shown in Figure 1-2.

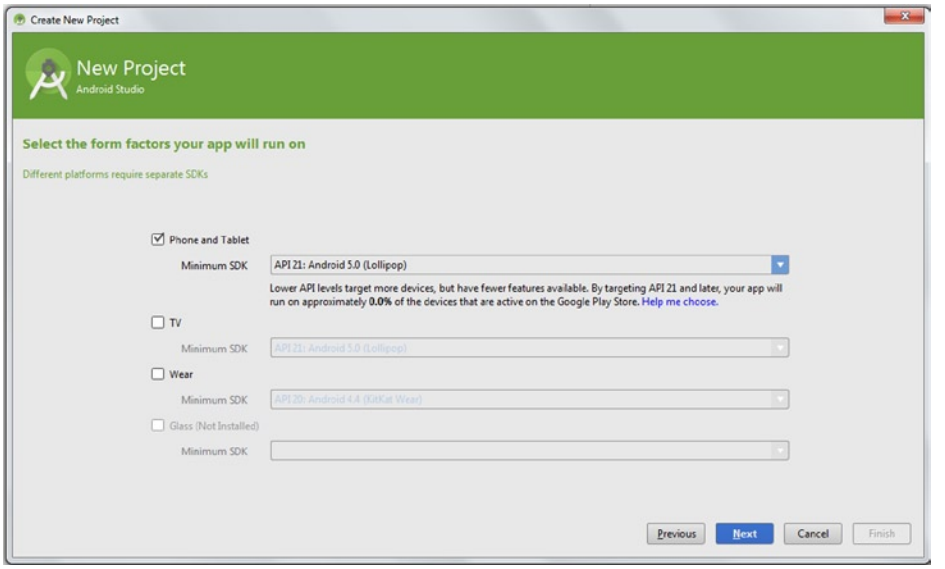

Figure 1-2. *New Project wizard showing the Android options after final configuration*

Android Studio 1.0.1 is ready for use. Now let's install the NDK.

Installing the Native Development Kit

The NDK is the critical component for creating great games. It provides all the tools (compilers, libraries, and header files) to build apps that access the device hardware natively.

As we said earlier in this chapter, when this manuscript was written, NDK and JNI were not supported by Android Studio so NDK and JNI must be used with the Eclipse IDE for Java Developers and Android SDK Tools.

> **Note** The NDK site is a helpful resource to find step-by-step instructions, API descriptions, changes, and all things related to native development. It is a must for all C/C++ developers. Go to `http://developer.android.com/sdk/ndk/index.html`.

The NDK installation requires two simple steps: downloading the NDK and installing Cygwin (a free tool to emulate a Linux-like environment on top of Windows, but more on this in the next sections).

NDK Install

Download and unzip the latest NDK from http://developer.android.com/sdk/ndk/index.html into your work folder (in this case C:\eclipse-SDK).

You now need to configure Eclipse so that it knows where the NDK is in order to use it when building your native application. Follow these steps to set the location of the NDK:

1. Launch Eclipse.
2. Open the Preferences window and select Android ➤ NDK to see the directory of the NDK we just installed (see Figure 1-3).

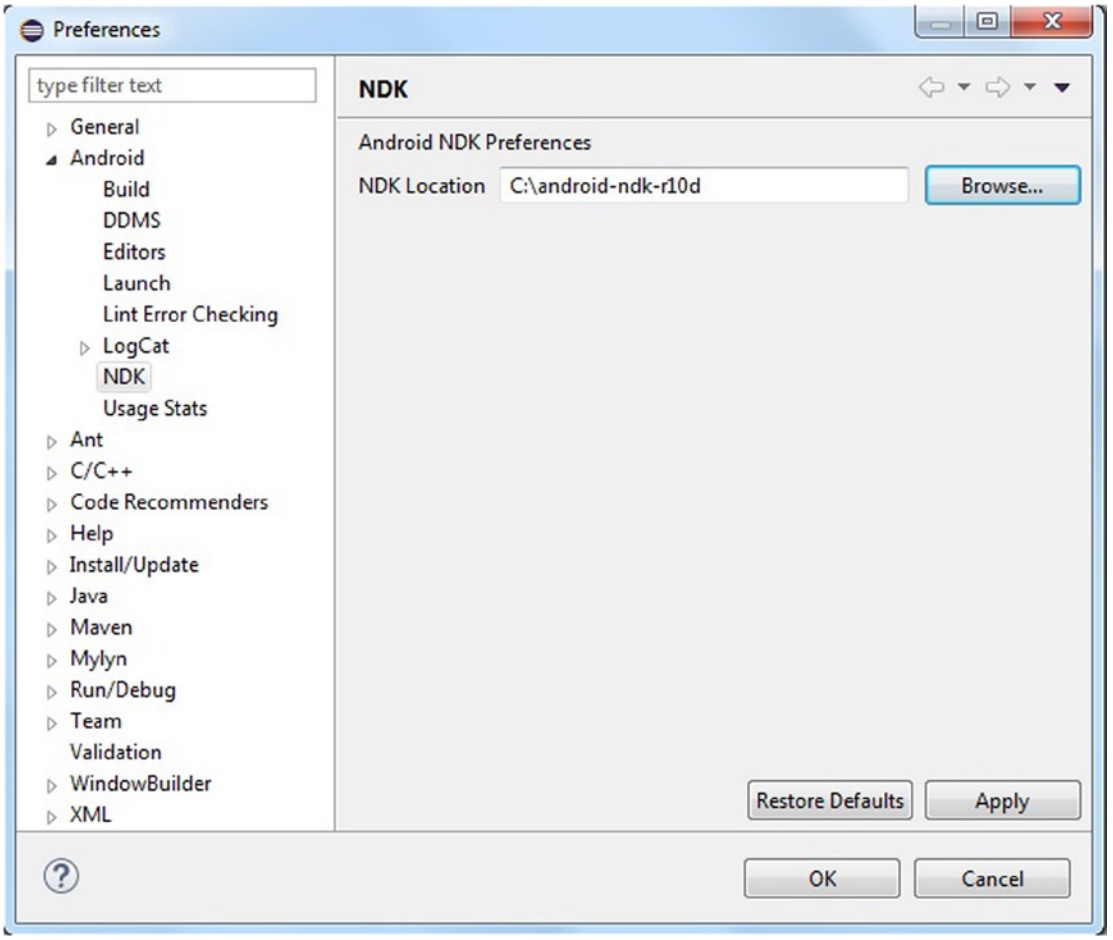

***Figure 1-3.** Preferences window showing where NDK is located*

To verify the installation in the Project Explorer pane, right-click the project name; the context menu should appear. Then, when selecting Android Tools, if you see the Add Native Support option, it means that NDK is fully installed and configured.

Install Cygwin

Android is built on top of Linux, which is not compatible with Windows. Enter Cygwin (version 1.7.9-1 or later). I have used the version 1.7.32 in this manuscript. This is a tool that provides a Linux environment and very useful software tools to do Linux-like work on Windows (such as building a program). It is necessary to run the NDK compilation scripts, and it is required if you are doing any type of native development.

For this manuscript I installed the Windows version of the 1.7.33 development packages such as Devel/make and Shells/bash.

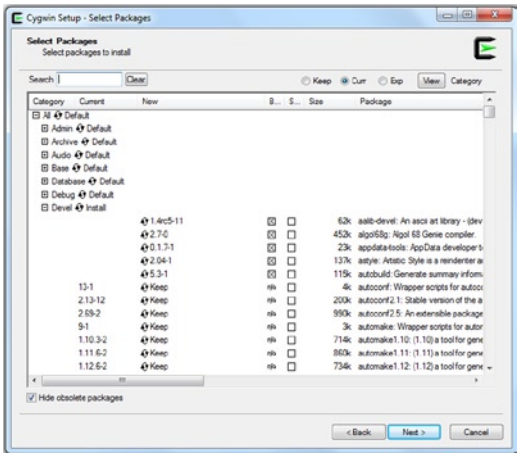

Figure 1-4. *Cygwin list of packages for installation*

For the examples we need to open the `Devel` (Development) category and at least select `gcc`, `g++`, `gdb`, and `make`, which are not part of the default selection.

> **Note** Cygwin is not required for native development in Linux.

As a last step you need to include the Cygwin Binary directory (bin) in the `PATH` environment variable. Because you installed Cygwin in the directory `c:\cygwin` you will need to select as follows in Windows:

Environment: Variables ➤ System Variables ➤ Select the variable named `PATH` ➤ Edit ➤ Add `c:\cygwin\bin;` in front of the existing `PATH` entry.

Let's verify Cygwin works properly by running the Cygwin Command shell (`bash` or `sh`) `cygwin.bat`. Cygwin Command shell information can be found at https://cygwin.com/cygwin-ug-net/using-utils.html.You should see the command prompt `$` as shown in Figure 1-5.

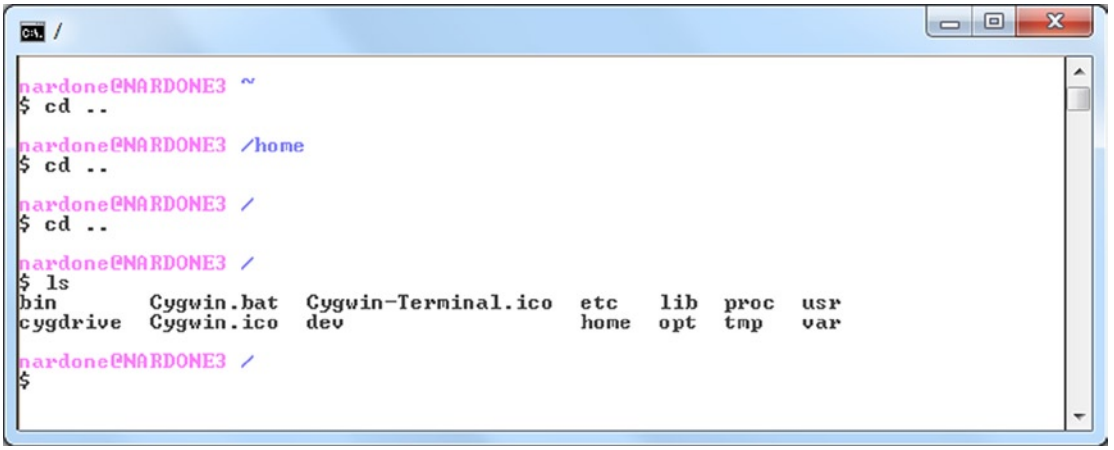

Figure 1-5. Cygwin console

Install MinGW

MinGW, which stands for Minimalist GNU for Windows, is a tool that can be used when developing native Microsoft Windows applications which do not depend on any third-party C-Runtime DLLs. You can download MinGW for Windows from the site http://sourceforge.net/projects/mingw/files/Installer/.

To verify that MinGW works properly run the MinGW executable file `mingw-get.exe` which is my case is located in `C:\MinGW\bin`. You should see the MinGW console as shown in Figure 1-6.

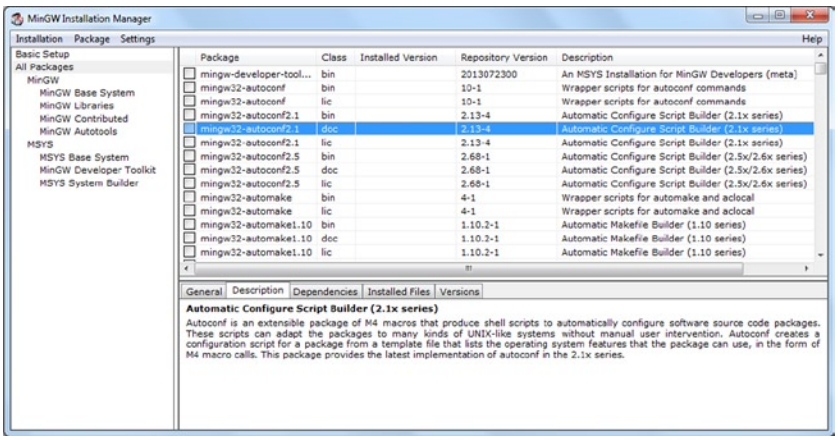

Figure 1-6. MinGW console

Install Ant

To compile projects from the command line, you need to install Ant, which is a Java-based build automation utility supported by the Android SDK.

1. Go to `http://ant.apache.org/bindownload.cgi` and download the Ant binaries, packed within a ZIP archive. In our case `apache-ant-1.9.4-bin.zip`.
2. Unzip Ant in the directory of your choice (for example, `C:\apache-ant-1.9.4`).
3. Verify the Ant installation by typing the following command:

```
ant -version
```

If properly installed you will see something like this:

```
Apache Ant(TM) version 1.9.4 compiled on April 29 2014
```

We can proceed to create the AVD to run the platform you just installed.

Creating an Android Emulator

Before you can start building your apps, you must create an Android Virtual Device (AVD), but you also need an SDK platform to run the AVD on. In this section you will learn how to do the following:

- Install the Android platform needed to test your apps
- Create the Virtual Device (emulator) that will run the above platform

> **Tip** This step is optional; however, I encourage you to do it because it helps with testing on multiple platforms. Nevertheless, if you have a real device such as a phone or tablet, you can skip this section and jump to the "Configuring a Real Device" section.
>
> For performance reasons, this is the best way to do it. AVDs are notoriously slow and lack many advanced features such as a robust implementation of OpenGL. I work on a laptop and running an OpenGL app in the emulator is painfully slow and full of missing API calls. I encourage you to test all of your code on a real device.

Creating an AVD

With version 1.5 and later of the SDK, Google introduced the concept of virtual devices (AVDs). An AVD is simply a set of configuration attributes applied to an emulator image that allows the developer to target a specific version of the SDK.

Follow these steps to create your AVD.

1. Click the AVD button on the Android Studio toolbar (see Figure 1-7).

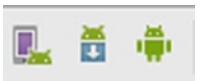

***Figure 1-7.** Android toolbar in Android Studio*

The AVD Manager is shown in Figure 1-8. The AVD Manager provides everything you need to:

- Create virtual devices (emulators)
- Delete/edit/start emulators

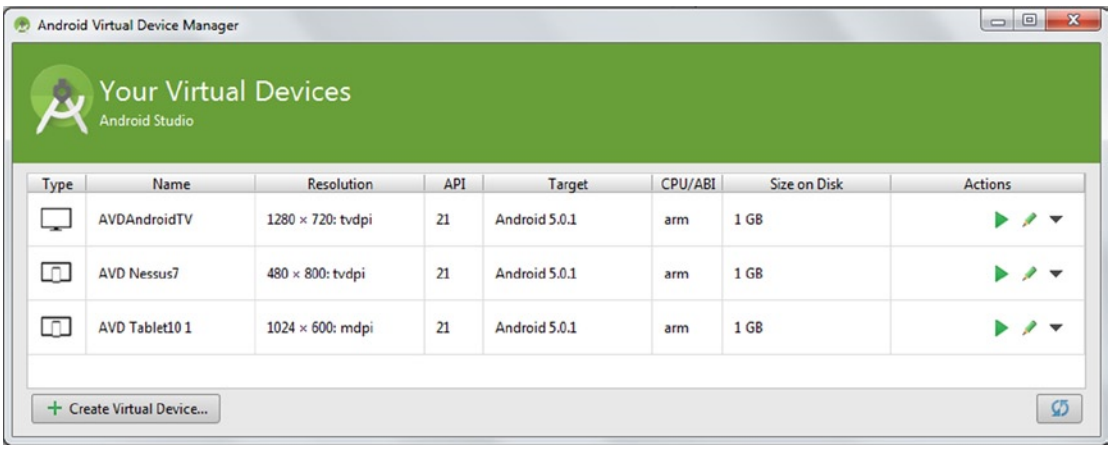

***Figure 1-8.** The AVD Manager*

2. To create an emulator, simply press the Create Virtual Device. . . button.
3. In the new device dialog, enter the device name and complete the following settings (Figure 1-9):
   - Select the device model you want to emulate. In this case the device Nexus model S was selected.
   - In the target box, select Android 5.0 – API Level 21 in this case.

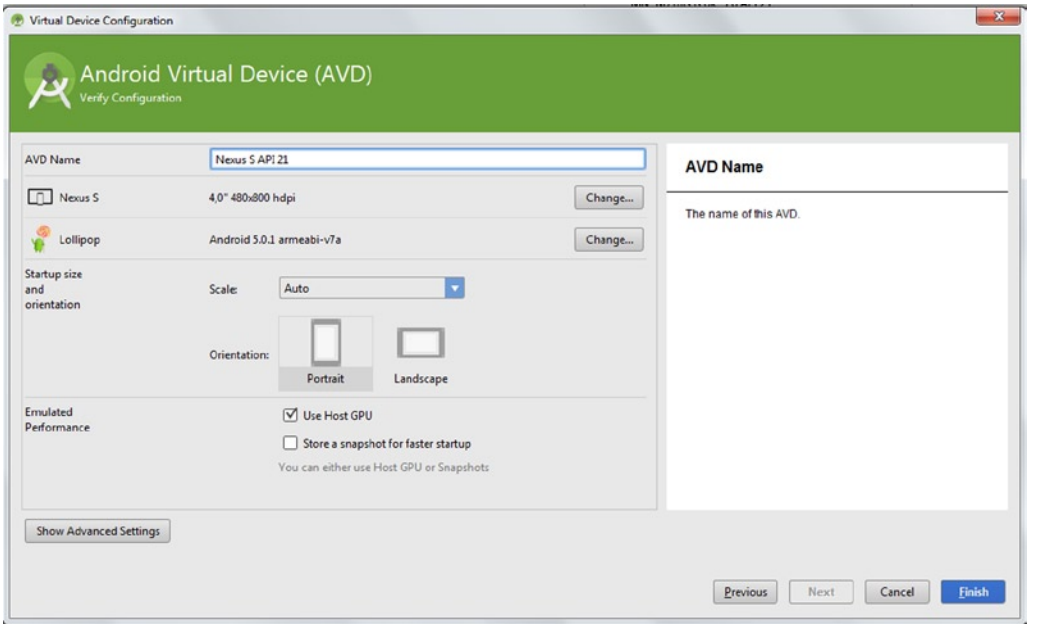

Figure 1-9. The Create new AVD dialog

4. Click Finish.

Now the emulator is ready for use (Figure 1-10).

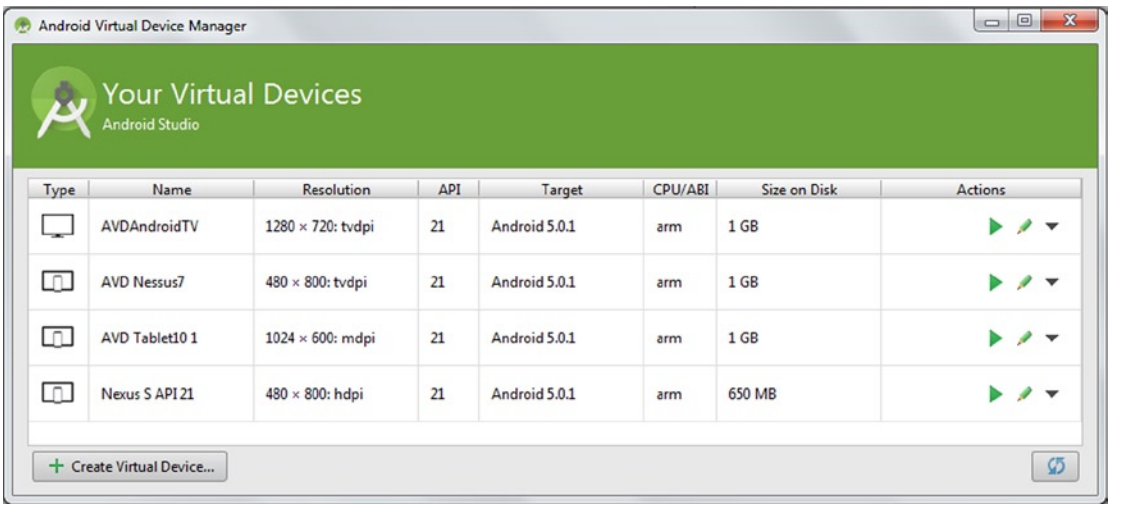

Figure 1-10. AVD Manager with brand new Nexus S device

> **Tip** You can create as many devices in as many versions or resolutions as you wish. This is helpful for testing in multiple API versions or screen sizes.

5. Run the emulator by selecting the device and pressing Start ➤ Launch. The emulator should boot up, and after a while you will be able to start playing with it (see Figure 1-11).

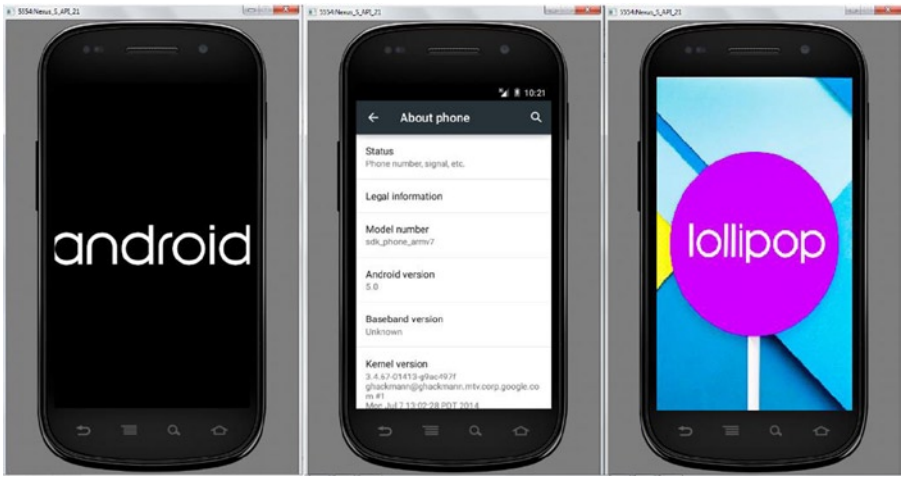

Figure 1-11. *Google Nexus S emulator (API level 21)*

In the same way we can create many other emulators, such as a 10.1 inch tablet for instance (see Figure 1-12).

Figure 1-12. *A 10.1-inch tablet (API level 21)*

Configuring a Real Device

Personally, I think using a real device is the best way to develop games and apps that use OpenGL. The emulator is painfully slow, plus not all the OpenGL API calls are implemented. A real device has many advantages the emulator does not, namely speed and API reliability. The only caveat is that the emulator gives you the chance to test multiple screen sizes and API versions. All in all, a real device is the way to go if you are building an OpenGL game; use the emulator as a backup test tool to check your game.

Before Android Studio can recognize your device, you need to install a USB driver required for communications.

> **Note** Before installing the USB driver, try to plug in in the device to your PC and see if it shows up in the Android Studio devices view (because the required driver may already be installed in your computer). If the device won't show up, then you need to install the USB driver.

MORE ON THE USB DRIVER

Notice that the Google USB driver is required for Windows only to perform Android Debug Bridge (ADB) debugging with any of the Google Nexus devices. The one exception is the Galaxy Nexus: the driver for Galaxy Nexus is distributed by Samsung (listed as model SCH-I515). Please take a look to the OEM USB drivers document at http://developer.android.com/tools/extras/oem-usb.html.

Windows drivers for all other devices are provided by the respective hardware manufacturer, as listed in the OEM USB Drivers document.

Here are some of the devices compatible with Google's USB driver:

- ADP1 / T-Mobile G1
- ADP2 / Google Ion / T-Mobile myTouch 3G
- Verizon Droid
- Nexus One, Nexus S, Nexus 4, Nexus 5, Nexus 7, and Nexus 10
- Galaxy Nexus
- PandaBoard (Archived)
- Motorola Xoom (U.S. Wi-Fi)
- Nexus S
- Nexus S 4G

Using the Nexus 7 that was originally sold with 4.1.2 or newer is not recommended. If you have other devices, you'll need to contact your vendor. A list of OEMs and more details on the USB driver for Windows is available at http://developer.android.com/sdk/win-usb.html.

To install the driver in your Windows host, follow these steps:

1. Connect your device to your computer's USB port. Windows detects the device and launches the Hardware Update Wizard.
2. Select Install from a list or select a specific location and click Next.
3. Select Search for the best driver in these locations, uncheck "Search removable media," and check "Include this location in the search."
4. Click Browse and locate the USB driver folder within your Android SDK installation (`PATH-TO-SDK\android-sdk-windows\extras\google\usb_driver\`).
5. Click Next to install the driver.

> **Tip** The USB driver can be downloaded or upgraded from the Android AVD Manager. To install Google, for instance, select Available Packages ➤ Google Inc ➤ Google USB Driver and click Install Selected, as shown in Figure 1-13.

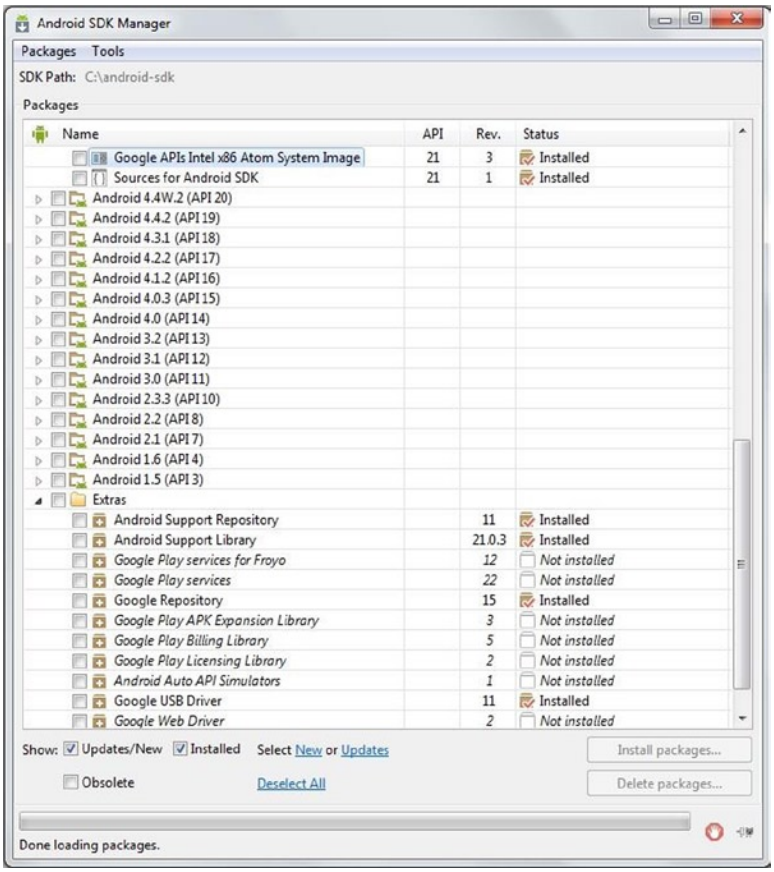

Figure 1-13. *Installing the USB driver from the AVD Manager*

6. After you have the USB driver properly installed, simply plug your phone into the USB port of your computer. Eclipse recognizes and displays it in the Android devices and log views, just as it does with the emulator.

The real device I used for this book was the Samsung Galaxy Grand 2, which, once the USB drivers are installed, will be listed as shown in Figure 1-14.

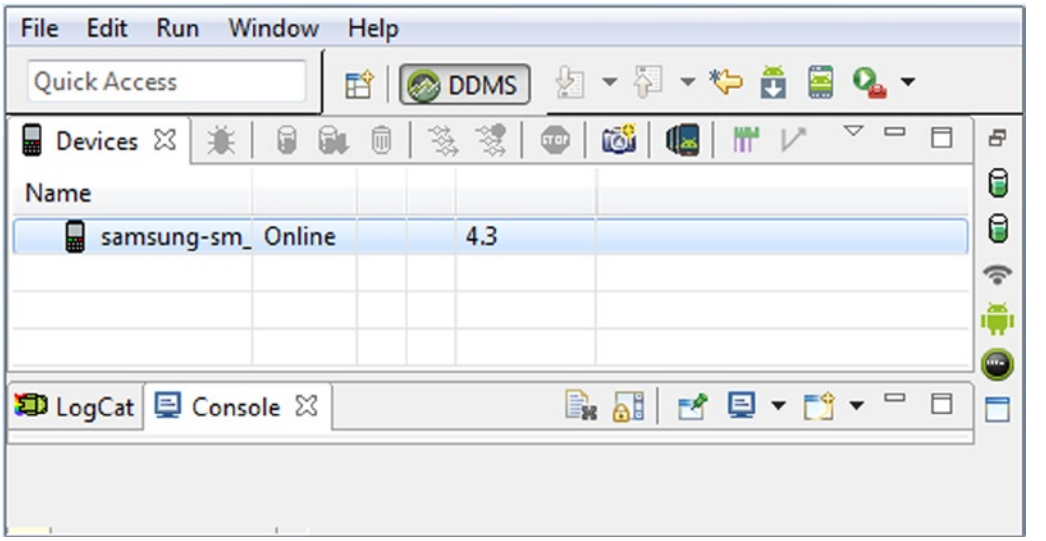

Figure 1-14. *A real device shown in the Android Studio Device view*

You can now use it to test your games!

At this point, your environment should be configured and ready for development, but before you finish, you must import the source code associated with this book into your Android Studio workspace.

Importing the Source Code into Android Studio

It is highly recommended that you import the companion source code for this book into your workspace. The later chapters describe game engines that are extremely complex, each with a minimum of 100K of source code in C/Java. I have tried my best to explain as simply and cleanly as possible the difficult concepts, and therefore most of the code listings have been stripped for simplicity. The source will allow you to understand what I am trying to emphasize on each chapter's listing. Each project has been named after its corresponding chapter number and game engine being described. To import the source code, follow these steps.

1. Obtain the source package (usually in ZIP format) from the Source Code/Download tab of the book's information page (www.apress.com/xxxxxxxxxxxxx) and decompress it into your working folder (in my case `C:\workspace`).
2. In the Android Studio main menu, click File ➤ Import Project. . . Point to the project you want to import from source package and click Ok.
3. The chapter project will be then loaded into Android Studio.

Each project in the source is named after a chapter number and engine being described. Look around each project to familiarize yourself with the project layout. You'll find that most projects have a large base of C code wrapped around thin Java wrappers. This workspace helps immensely for understanding the later chapters that describe native engines, which are big and difficult to explain.

Summary

Congratulations! You have taken the first step toward your mastery of Android game development. In this chapter, you learned how to set up your system to compile hybrid games, which included the following tasks:

- Installing the Android Studio 1.0.1
- Installing and configuring the Android NDK
- Creating emulators or real devices

This chapter has provided the foundation to compile the games described throughout this book. In Chapter 2, you will learn how to write and compile a basic native program (or shared library) and call it within a Java application.

Chapter 2

Gaming Tricks for Phones or Tablets

In this chapter, you will get your feet wet with development in Android. You'll learn how easy it is to build a simple Android app that loads a native library and executes methods written in C within Java. You'll also learn how to call back Java methods within C, so the communication goes both ways between the languages. Next you'll learn how to cascade audio, video, key, and touch events from Java to C by using thin Java wrappers that simply pass event information to the native library. And finally, you'll tackle multi-touch, which is a useful subject for advanced games such as first person shooters and others. Multi-touch can be a tricky subject, so check it out. But first, you'll learn how to add native code to your Android project.

> **Note** All the code included in this chapter was developed, compiled and tested using Eclipse IDE for Java Developers instead of Android Studio 1.0.1 since, when this manuscript was written, NDK and JNI were not supported by Android Studio.

Let's get started by setting up the Windows environment.

Setting Up Windows

For the examples in this chapter we will be using the following tools:

- The JDK (Java Development Kit)
- The Android SDK (Software Development Kit)
- The Android NDK (Native Development Kit)
- An IDE (Integrated Development Environment): Eclipse
- Cygwin

All components in this list were installed as shown in Chapter 1 so let's install Ant to complete the installation of the tool needed.

Windows System Environment Variables

Let's configure the Windows system environment variables needed in this chapter.

1. Open the Windows System Environment Variables
2. Create the following:
   - `%ANDROID_NDK%`
   - `%ANDROID_SDK%`
   - `%ANT_HOME%`
   - `%JAVA_HOME%`
3. Finally edit PATH and add the following to the existing path:

   %JAVA_HOME%\bin;%ANDROID_SDK%\tools;%ANDROID_SDK%\
   platform-tools;%ANDROID_NDK%;%ANT_HOME%\bin;
4. Save and close the window (see Figure 2-1).

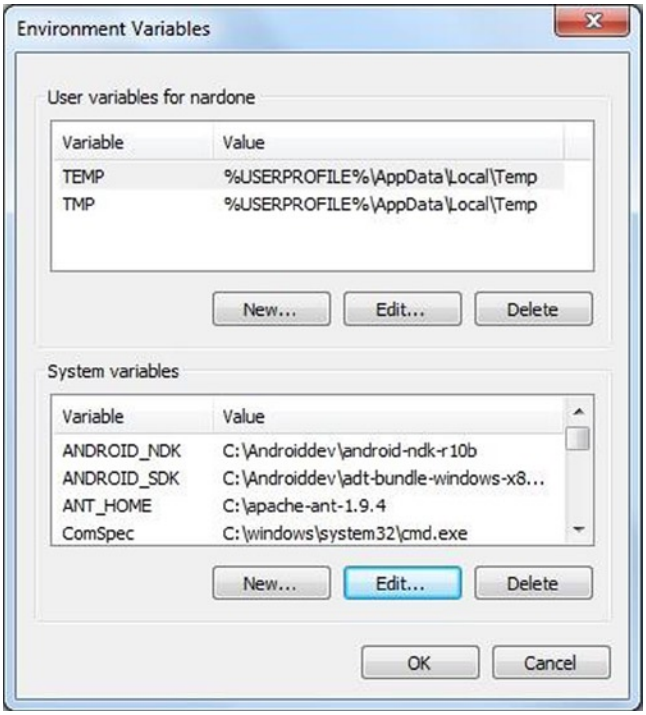

***Figure 2-1.** Windows system environment variables*

Configuring Cygwin

After you have installed Cygwin, to run any of its command you need to write the paths like this:

```
/cygdrive/<Drive letter>/<Path to your directory>.
```

So for instance, because I installed Ant in `c:\ apache-ant-1.9.4`, then the path must be indicated as `/cygdrive/c/ apache-ant-1.9.4.`

We can solve this by going to the Cygwin/home directory and editing the file named `.bash_profile`.

Edit the file and add the following at the end of the file:

```
export ANDROID_SDK='cygpath-u "$ANDROID_SDK"'
export ANDROID_NDK='cygpath-u "$ANDROID_NDK"'
export ANT_HOME='cygpath-u "$ANT_HOME"'
export JAVA_HOME='cygpath-u "$JAVA_HOME"'
```

Now you can call any of the tools included in those directories without having to add all the directory path. For instance, if you try to call `ant -version` from the Cygwin directory you can see that it runs properly: