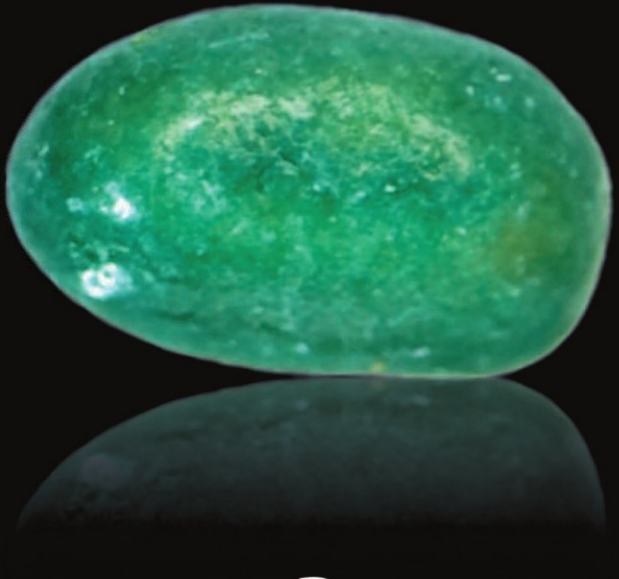
Create and port cool 3D games using Android



Pro Android Games

SECOND EDITION

Vladimir Silva

Apress®

Pro Android Games
Second Edition

Vladimir Silva

Apress®

Pro Android Games, Second Edition

ISBN 978-1-4302-4797-5

ISBN 978-1-4302-4798-2 (eBook)

President and Publisher: Paul Manning
Lead Editor: Steve Anglin
Technical Reviewer: Jim Graham
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Louise Corrigan, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Tom Welsh
Coordinating Editor: Christine Ricketts
Copy Editor: Mary Behr
Compositor: Bytheway Publishing Services
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `www.springeronline.com`.

For information on translations, please e-mail `rights@apress.com`, or visit `www.apress.com`.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at `www.apress.com/bulk-sales`.

Any source code or other supplementary materials referenced by the author in this text is available to readers at `www.apress.com`. For detailed information about how to locate your book's source code, go to `www.apress.com/source-code`.

Contents at a Glance

Contents

About the Author

Vladimir Silva was born in Quito, Ecuador. He received a systems analyst degree from Ecuador's Army Polytechnic Institute in 1994. The same year, he came to the United States as an exchange student pursuing a master's degree in computer science at Middle Tennessee State University. After graduation, he joined the IBM WebAhead technology think tank. His interests include grid computing, neural networks, and artificial intelligence. He also holds numerous IT certifications including Oracle Certified Professional (OCP), Microsoft Certified Solution Developer (MCSD), and Microsoft Certified Professional (MCP). He has written many technical articles on security and grid computing for IBM developerWorks.

About the Technical Reviewer

Jim Graham received a Bachelor of Science in Electronics with a specialty in telecommunications from Texas A&M and graduated with his class (Class of '88) in 1989. He was published in the International Communications Association's 1988 issue of ICA Communique ("Fast Packet Switching: An Overview of Theory and Performance"). His work experience includes working as an Associate Network Engineer in the Network Design Group at Amoco Corporation in Chicago, IL; a Senior Network Engineer at Tybrin Corporation in Fort Walton Beach, FL; and as an Intelligence Systems Analyst at both 16th Special Operations Wing Intelligence and HQ US Air Force Special Operations Command Intelligence at Hurlburt Field, FL. He received a formal Letter of Commendation from the 16th Special Operations Wing Intelligence on December 18, 2001.

Introduction

Welcome to *Pro Android Games, Second Edition*. This book will help you create great games for the Android platform. There are plenty of books out there that tackle this subject, but this book gives you a unique perspective by showing you how easy is to bring native PC games to the platform with minimum effort. This is done using real world examples and source code in each chapter. To get the most out of this book, you must have a solid foundation in Java and ANSI C. I have made a great effort to explain the most complicated concepts as clearly and as simply as possible using a combination of graphics and sample code. The 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 is has been just a few years since Android came into the smartphone scene. And it has taken off with a vengeance. Take a look at the US Smartphone platform market share survey by IDC[1] shown in Figure 1. Android now commands a whopping 59% of the world smart phone share (up from 36% in 2011), which is not too shabby for a two-year-old OS. 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 59 percent of world smartphone share," www.engadget.com/2012/05/24/idc-q1-2012-world-smartphone-share/

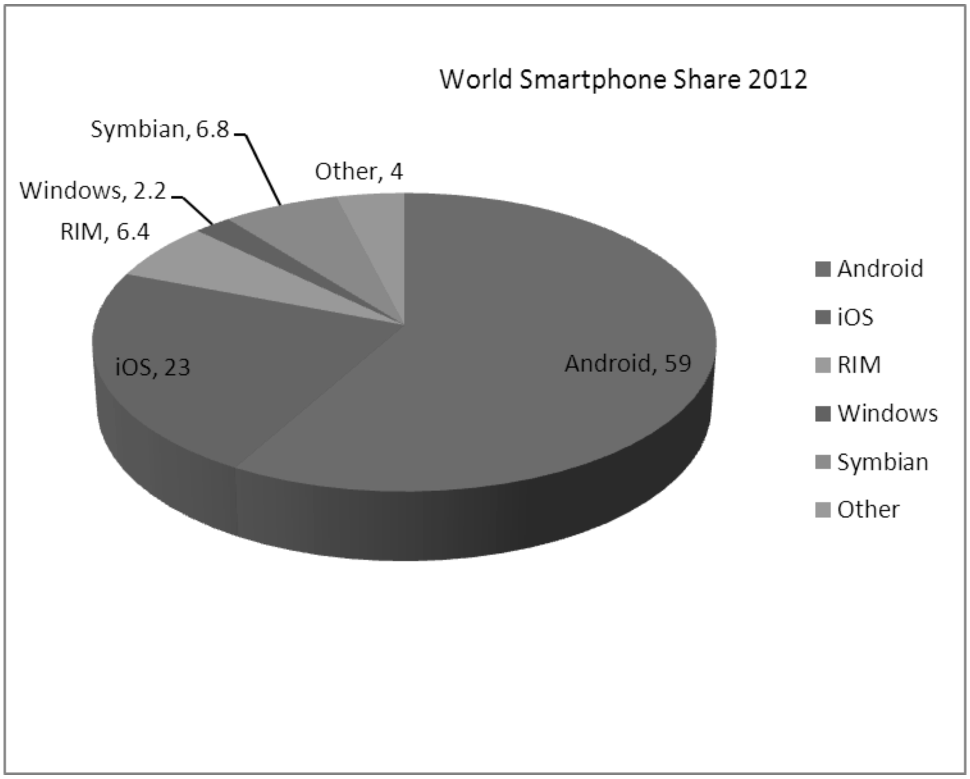


Figure 1. World smartphone share 2012

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*: Especially if they work on native applications, this book can be a valuable tool for business types.
- *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 help students in many platforms (iPhone, Blackberry, Meego, and others).
- *Anybody 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 will display as the background the file called `doom.png` stored in the `res/drawable` folder of the project, and will receive key and touch events. In the lower left of the screen, overlapping the image view, an image button with the ID `btn_upleft` will be displayed.

NEED AN ANDROID TUTORIAL?

There are a lot of concepts related to Android development, 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/1.6_r1/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 do the following:

- *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 (except Chapter 1) in this book use a hybrid combination of Java/C development, which requires knowledge about native development. For these chapters, you will 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

I guess this is kind of obvious, as 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.

Eclipse IDE and Android SDK, Properly Installed

Eclipse is the de facto IDE for Android development. I used Eclipse Galileo to create the workspace for the book; Eclipse Ganymede should work as well.

NEED A DEVELOPMENT IDE?

Even though Eclipse Helios 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 IDE's such as IntelliJ, or a basic editor, go to `http://developer.android.com/guide/developing/other-ide.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 will use the NDK extensively, thus a solid foundation of C programming is required to fully understand the concepts presented in each chapter. 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 chapter as simply as possible; nevertheless some of the games (especially Quake I & 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. Get the companion source for the book from the publisher. It was built using the latest Eclipse SDK.

What Makes This Book Unique?

I think it is important for the reader to understand my goal with this manuscript and what I believe sets this book apart. 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 like 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? This is my goal and what makes this book stand out. However, the 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*: Including the latest version of the Android SDK, NKD and the latest Eclipse IDE.
- *Greater focus on the native side*: I think it's fair to say that Java has fallen out of grace with 3D game developers, especially for powerful games. Java's lack of performance and memory constraints are the main culprits. Therefore *Pro Android Games* puts greater emphasis in native game development and hardware-accelerated graphics.
- *Bigger and better real world engines*: My goal is not simply to provide you with some tricks to develop games, but to show you with real, powerful, and bigger-than-life samples. To that end, this book will show you how powerful PC-caliber game engines such as Quake I and II can be brought to your mobile device with almost no change whatsoever. This book also includes an oldie from the previous edition: Doom.

Android SDK Compatibility

As a developer, you may ask yourself about the SDK compatibility of the code in this book. This is an important question, as new versions of the Android SDK come out frequently. At the time of this writing, Google released the Android SDK version 3.1. The code in this chapter has been fully tested with the following versions of the Android SDK:

- SDK version 4.1
- SDK version 3.x

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 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 IDE 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 will learn how to combine Java and C code in an elegant manner by building a simple Java application on top of a native library. You will 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 keyword, how to generate the JNI headers, plus all about method signatures, Java arrays vs. 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 by coincidence 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 will show 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. Consider the potential savings in development time and costs! This chapter is a must read.

Chapter 6

Here is where things start to get really exiting! This chapter brings to you a first-person shooter (FPS) gem: Quake. You will learn 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. You will also learn 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. This chapter will show you how a simple combo of very powerful tools can tame the mighty Quake II OpenGL renderer. Check it out!

Chapter 8

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

Chapter 9

This chapter tackles to interesting new subjects coming soon to your mobile device and living room: augmented reality and Google TV. The future of casual gaming seems to be headed the AR way. Other platforms like the PS Vita have 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 will show 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 Google TV. Google TV is powered by Android 3.1 and thus fully compatible with the games you may be planning to create.

Contacting the Author

Should you have any questions or comments—or should you spot a mistake you think I should know about—you can contact the author at `vladimir_silva@yahoo.com`.

Chapter 1

Welcome to the World of the Little Green Robot

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 Eclipse in this case, and the Android software development kit (SDK), 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:

- *Eclipse*: This is the development IDE used to create your projects. I have used Eclipse Indigo version 3.7in this manuscript; however, Version 3.6 (Helios) or 3.5 (Galileo) will work as well.
- *Android SDK (properly configured)*: At the time of this writing, the latest version of the SDK is 4.0.3.

- *Java SDK:* This is required to run Eclipse and the Android SDK itself. Any version of Java after 5.0 should work just fine.

The next section will go through the process of setting up your machine step by step.

Setting Up Your Machine

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 very first and most basic thing you need is a current Java SDK/JRE (5.x or 6.x will do). Make sure you have the proper version installed before proceeding. The steps here assume that you already do. Note that at the time of this writing Android does not support Java 7. It is unclear when this will be fixed.
2. Download and Install the Android SDK. The SDK contains the core resources to develop for Android.
3. Configure Eclipse. You need to install the Eclipse plug-in for Android before you can build anything at all.
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, the latest version is r8b. All in all, keep in mind that Android 4 is binary compatible with older NDK versions. This means that if you have an old NDK, it will work just fine. Nevertheless, it is always good to use the latest version; it may provide a performance boost to your native code.
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 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 SDK Starter Package for windows from `http://developer.android.com/sdk/index.html` and unzip it to a working folder such as `C:\eclipse-SDK`.

TIP: Try to keep the SDK, NDK, and Eclipse in the same working folder, such as `C:\eclipse-SDK`. I find this helpful when working on multiple projects at the same time. Thus my development folder `C:\eclipse-SDK` contains the subfolders `android-sdk-windows` (for the SDK), `android-ndk-r6b` (for the NDK), and `eclipse` (for Eclipse 3.7). Now let's configure your Eclipse environment.

Configure Your Eclipse

You are ready to get your IDE up and running with the Android development kit. Let's go through the installation of the Android 4 SDK (available from `http://developer.android.com/sdk/index.html`) over Eclipse 3.7 (Indigo, available from `www.eclipse.org`).

1. Start Eclipse and select Help ➤ Check for Updates, as shown in Figure 1-1. The Available Software window will display.

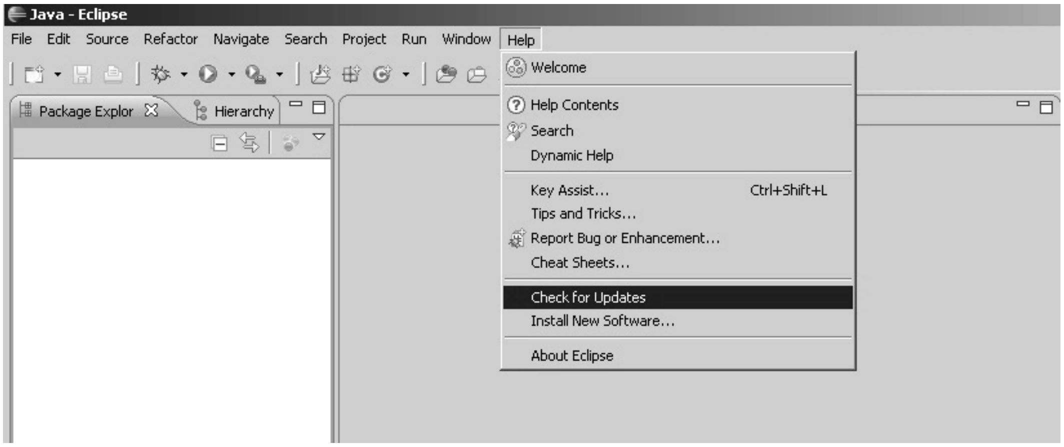
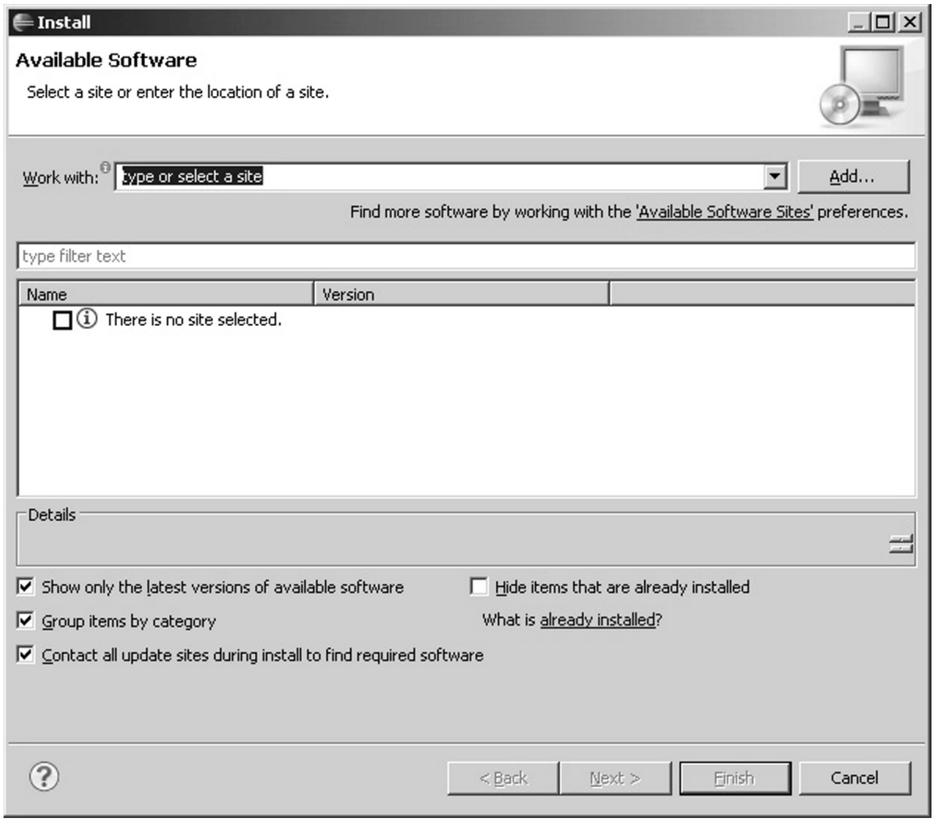


Figure 1-1. *Choosing Check for Updates from the Eclipse 3.7 workbench's Help menu*

2. In the Available Software window, shown in Figure 1-2, click the Add button to install new software. The Add Site dialog will show up.


Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the 'Available Software Sites' preferences.

type filter text

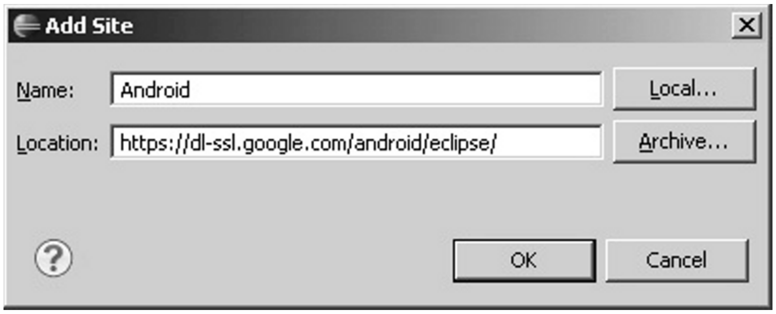
Name	Version	
☐ ⓘ There is no site selected.		

Details

☑ Show only the latest versions of available software
☐ Hide items that are already installed
☑ Group items by category
What is already installed?
☑ Contact all update sites during install to find required software

Figure 1-2. *Adding software*

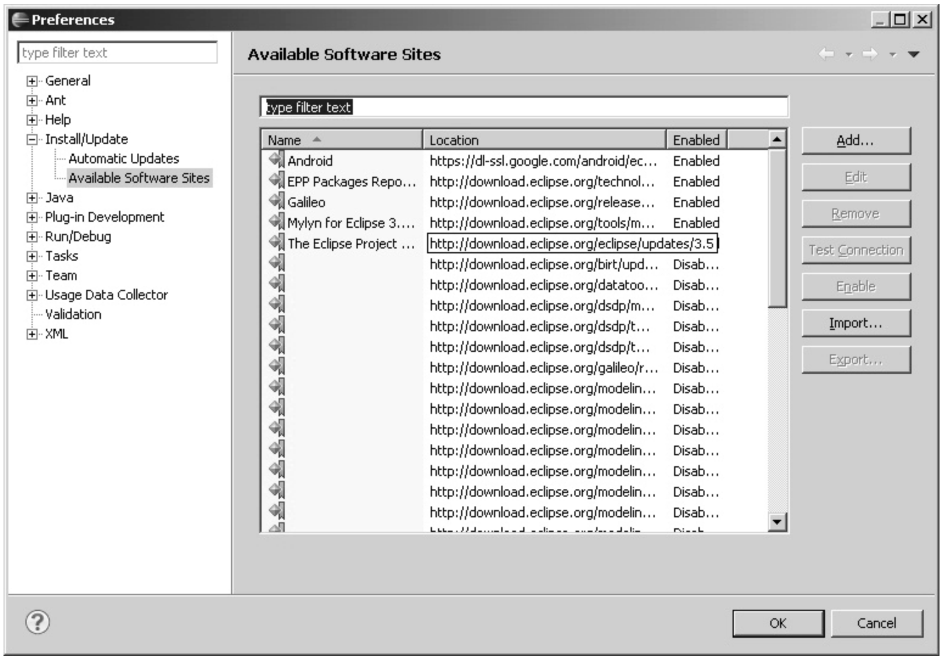
3. In the Add Site dialog box, enter **Android** for the name and `https://dl-ssl.google.com/android/eclipse` for the location, as shown in Figure 1-3.



Name: Android
Location: https://dl-ssl.google.com/android/eclipse/

Figure 1-3. *Adding the Android site*

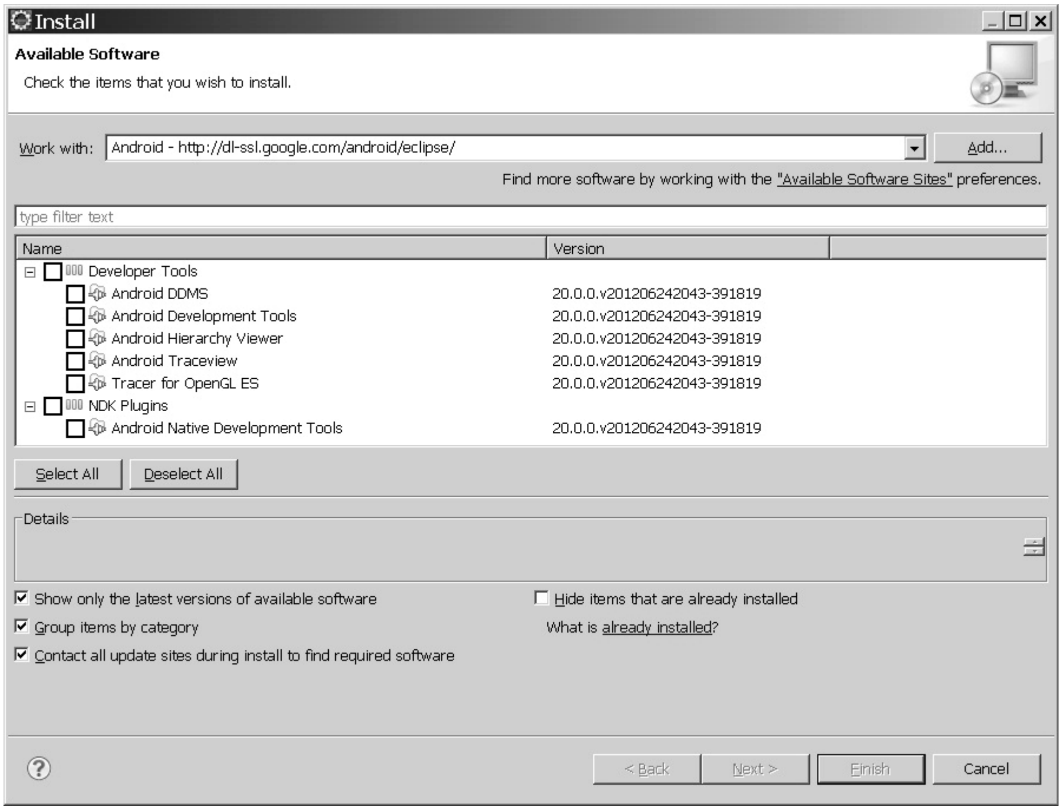
4. From the Available Software window (Figure 1-2), select the Android site you just added from the Work with combo box. If the name is not shown in the list, click the Available Software Sites preferences link, and then click Add to insert the site into the list (see Figure 1-4).



Name	Location	Enabled
Android	https://dl-ssl.google.com/android/ec...	Enabled
EPP Packages Repo...	http://download.eclipse.org/technol...	Enabled
Galileo	http://download.eclipse.org/release...	Enabled
Mylyn for Eclipse 3....	http://download.eclipse.org/tools/m...	Enabled
The Eclipse Project ...	http://download.eclipse.org/eclipse/updates/3.5	
	http://download.eclipse.org/birt/upd...	Disab...
	http://download.eclipse.org/datatoo...	Disab...
	http://download.eclipse.org/dsdp/m...	Disab...
	http://download.eclipse.org/dsdp/t...	Disab...
	http://download.eclipse.org/dsdp/t...	Disab...
	http://download.eclipse.org/galileo/r...	Disab...
	http://download.eclipse.org/modelin...	Disab...
	http://download.eclipse.org/modelin...	Disab...
	http://download.eclipse.org/modelin...	Disab...
	http://download.eclipse.org/modelin...	Disab...
	http://download.eclipse.org/modelin...	Disab...
	http://download.eclipse.org/modelin...	Disab...
	http://download.eclipse.org/modelin...	Disab...

Figure 1-4. *The Available Software Sites Preferences window shows the recently added Android site.*

5. You should now see the Android Developer Tools in the list, as shown in Figure 1-5. Check the Developer Tools checkbox, and then click Next.



Available Software

Check the items that you wish to install.

Work with: Android - http://dl-ssl.google.com/android/eclipse/

Find more software by working with the "Available Software Sites" preferences.

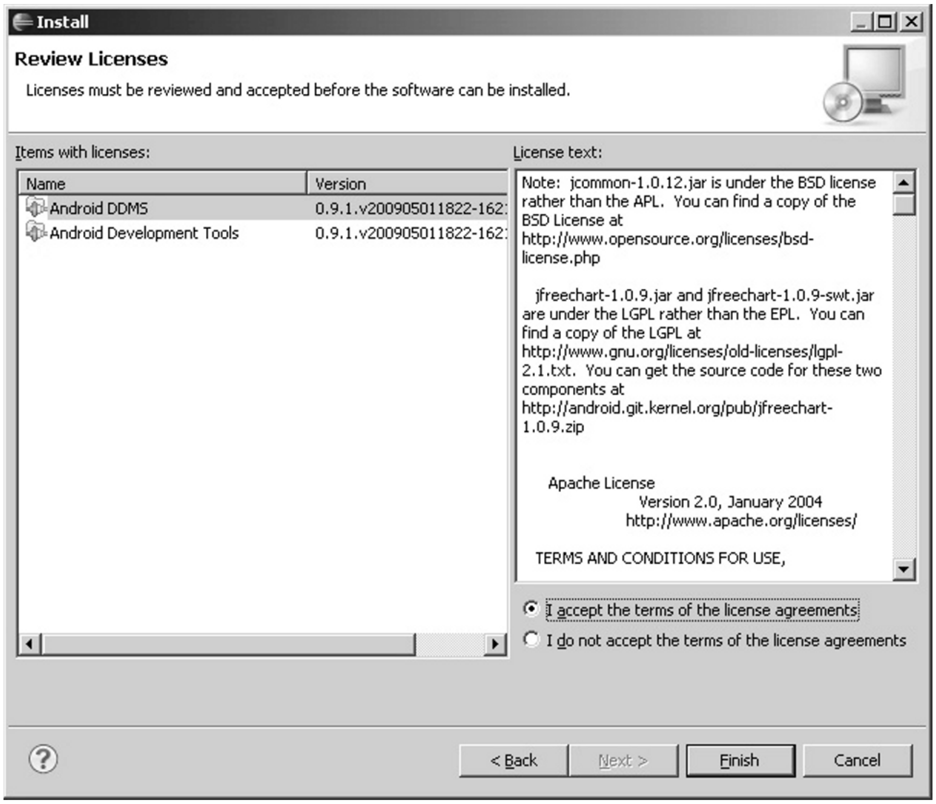
type filter text

Name	Version
☐ Developer Tools	
☐ Android DDMS	20.0.0.v201206242043-391819
☐ Android Development Tools	20.0.0.v201206242043-391819
☐ Android Hierarchy Viewer	20.0.0.v201206242043-391819
☐ Android Traceview	20.0.0.v201206242043-391819
☐ Tracer for OpenGL ES	20.0.0.v201206242043-391819
☐ NDK Plugins	
☐ Android Native Development Tools	20.0.0.v201206242043-391819

☑ Show only the latest versions of available software
☐ Hide items that are already installed
☑ Group items by category
What is already installed?
☑ Contact all update sites during install to find required software

Figure 1-5. *Available Software window with the Android plug-in selected*

6. Follow the wizard installation instructions, accept the license agreement (as shown in Figure 1-6), and then complete the installation. At the end, the workbench should ask for a restart.


Install
Review Licenses
Licenses must be reviewed and accepted before the software can be installed.
Items with licenses:
License text:

Name	Version
Android DDMS	0.9.1.v200905011822-162
Android Development Tools	0.9.1.v200905011822-162

Note: jcommon-1.0.12.jar is under the BSD license rather than the APL. You can find a copy of the BSD License at http://www.opensource.org/licenses/bsd-license.php

jfreechart-1.0.9.jar and jfreechart-1.0.9-swt.jar are under the LGPL rather than the EPL. You can find a copy of the LGPL at http://www.gnu.org/licenses/old-licenses/lgpl-2.1.txt. You can get the source code for these two components at http://android.git.kernel.org/pub/jfreechart-1.0.9.zip

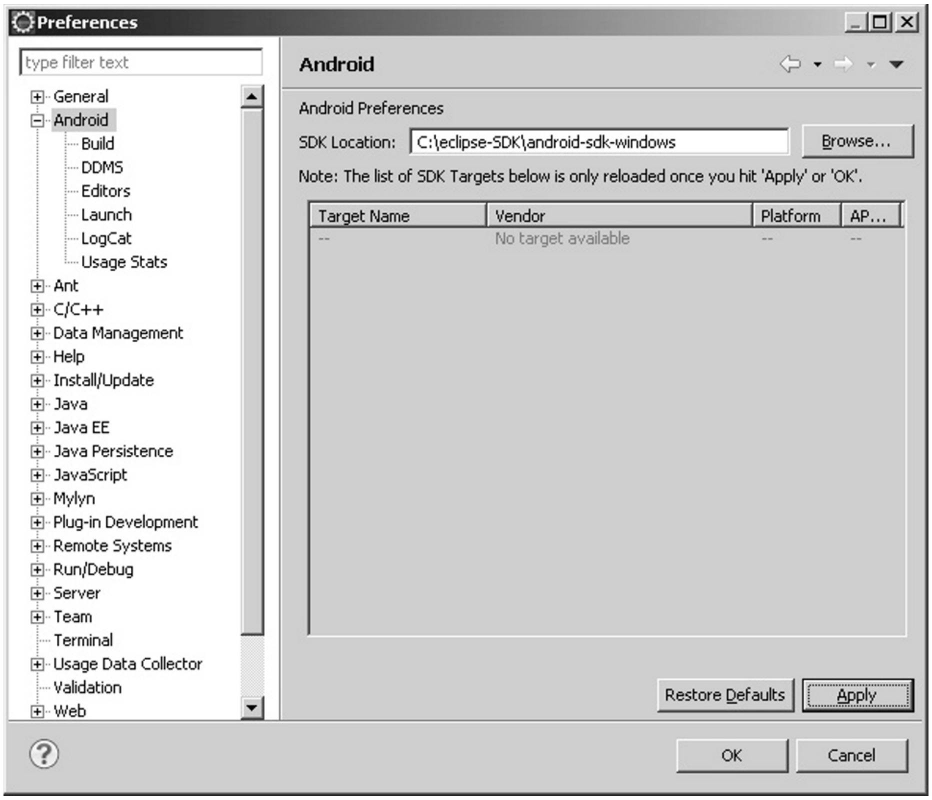
Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE,

[✓ I accept the terms of the license agreements]
I do not accept the terms of the license agreements

Figure 1-6. *Software license agreement from the installation wizard*

7. After the workbench restarts, select Window ➤ Preferences to open the workbench Preferences window, and select the Android option from the left navigation tree. In the Preferences section, set the location of your SDK, as shown in Figure 1-7. Make sure all the build targets are shown. Then click Apply.



Android

Android Preferences

SDK Location: C:\eclipse-SDK\android-sdk-windows

Note: The list of SDK Targets below is only reloaded once you hit 'Apply' or 'OK'.

Target Name	Vendor	Platform	AP...
--	No target available	--	--

Figure 1-7. *Workbench Preferences window showing Android options*

8. Click OK, and then open the 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 1-8.

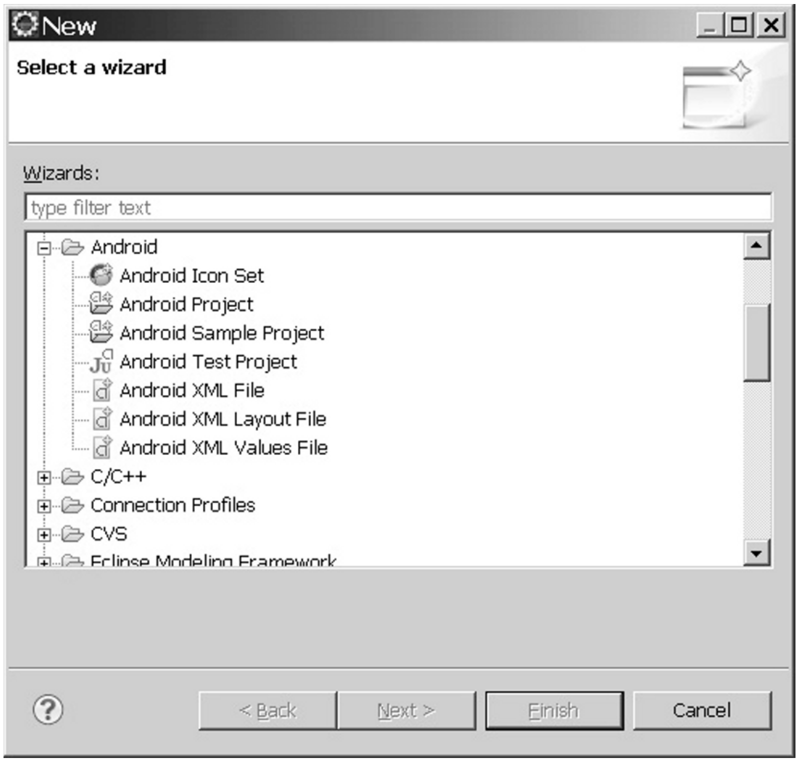


Figure 1-8. *New Project wizard showing the Android options after final configuration*

Eclipse is ready for use. Now let's install the NDK.

Installing the Native Development Kit

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

NOTE: The NDK site is a very 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

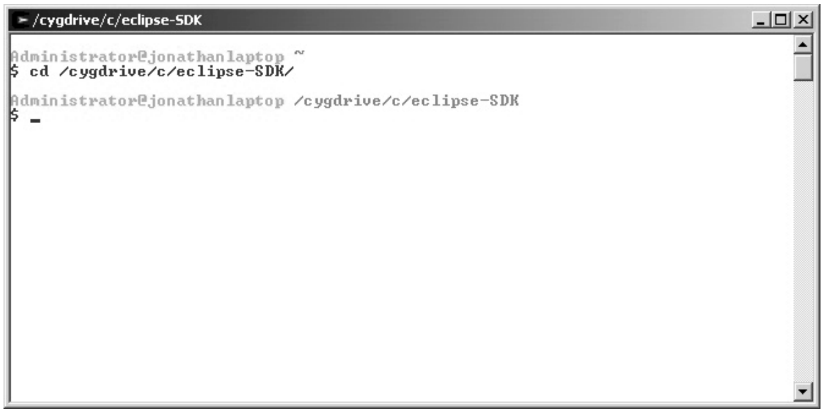
Download and unzip the latest NDK from `http://developer.android.com/sdk/ndk/index.html` into your work folder (in my case `C:\eclipse-SDK`).

Install Cygwin

As we all know, Android is built on top of Linux, which is not compatible with Windows. Enter Cygwin (version 1.7.9-1 or later). 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.

NOTE: Cygwin is not required for native development in Linux.

To Install Cygwin, download and run the installer (`setup.exe`) from the Cygwin site available at `www.cygwin.com/`. Follow the wizard instructions. After the installer completes you should see the Cygwin icon in your desktop. Double-click it and test by changing to your work folder (type `cd /cygdrive/c/eclipse-SDK`, as shown in Figure 1-9).



```
Administrator@jonathanlaptop ~
$ cd /cygdrive/c/eclipse-SDK/

Administrator@jonathanlaptop /cygdrive/c/eclipse-SDK
$ _
```

Figure 1-9. *Cygwin console*