부록 B. NDK r3 활용

류광

참고

이 부록은 역자가 작성한, 원서에는 없는 번역서만의 추가 내용이다. NDK r3의 최신 기능 자체가 이 부록의 초점은 아니다. 이 부록의 주된 목적은 제2장의 작은 예제(개발 환경 점검이 목적인)를 제외한 이 책의 주요 예제들을 독자가 따로 리눅스를 준비하지 않고 전적으로 Windows에서 빌드하고 실행할 수 있게 한다는 것이다. 다만, 저자 서문의 첫 역주와 역자의 글에서 언급했듯이, Windows+NDK 조합만으로는 이 책을 최대한 향유하기 힘들다는 점도 기억하기 바란다.

Google은 2010년 3월에 NDK r3을 내놓았다. NDK r3의 주요 변경 사항들을 들자면 다음과 같다(http://android-developers.blogspot.com/2010/03/android-ndk-r3.html 참고).

- NDK 1.6의 여러 버그들이 수정되었다.
- GCC 버전이 좀 더 작고 효율적인 이진 코드를 생성하는 4.4.0로 올라갔다. 이전 버전(4.2.1)도 남아 있으므로, 필요하다면 이전 버전을 사용하는 것도 가능하다.
- OpenGL ES 2.0 네이티브 라이브러리를 지원한다.
- NDK r3부터는 1.5, 1.6 같은 SDK 버전 번호 대신 독자적인 번호를 사용한다. r3은 이것이 NDK의 세 번째 개정판(revision)임을 뜻한다.

NDK 1.6과 마찬가지로, NDK r3의 설치는 아주 간단하다. 그냥 http://developer.android.com/sdk/ndk/에서 내려 받은 압축 파일을 원하는 디렉터리에 풀기만 하면 된다. 이하의 내용에서 NDK r3이 설치된 디렉터리를 <NDK_R3>이라고 칭하겠다. 부록 A에서 말한 것처럼 Cygwin과 GNU Make가 설치되어 있으며, 역주에서 언급한 build/ host-setup.sh도 실행했다고 가정하겠다.

다음 절들에서는 NDK r3을 이용해서 제5장 OpenGL 입방체 예제와 제6장 Wolfenstein 3D 예제의 네이티브 라이브러리를 컴파일, 링크하는 방법을 살펴본다. 그럼 OpenGL 예제부터 시작하자.

[1] http://occamsrazr.net/tt/230

NDK r3으로 제5장 OpenGL 입방체 예제 컴파일하기

부록 A에 NDK 1.5와 안드로이드 소스를 이용해서 제5장 OpenGL 입방체 예제의 네이티브 라이브러리를 컴파일, 링크하는 방법이 나왔는데, NDK 1.6부터는 NDK가 OpenGL ES를 공식적으로 지원하므로 안드로이드 소스에 의존하지 않아도 된다. 다음은 제5장 OpenGL 입방체 예제의 네이티브 라이브러리를 NDK r3을 이용해서 빌드하는 과정이다.

폴더 구조 준비

가장 먼저 할 일은 <NDK_R3>의 apps 폴더에 OpenGL 예제를 위한 폴더 구조를 만들고 프로젝트 파일들을 복사하는 것이다. 구체적인 과정은 다음과 같다.

1. <NDK_R3>\apps에 gltest-jni라는 이름의 폴더를 만든다.

2. 웹 예제 코드의 ch05.OpenGL 폴더를 <NDK_R3>\apps\gltest-jni 폴더 안에 복사한다(ch05.OpenGL이 <NDK_R3>\apps\gltest-jni의 한 하위 디렉터리가 되도록).

3. <NDK_R3>\apps\gltest-jni\ch05.OpenGL 폴더의 이름을 project로 바꾼다. 즉, <NDK_R3>\apps\gltest-jni\project가 되게 한다. 이후 폴더를 Eclipse 작업공간의 한 프로젝트로 도입할 것이다.

4. <NDK_R3>\apps\gltest-jni\project\native 폴더의 이름을 jni로 바꾼다. 즉, <NDK_R3>\apps\gltest-jni\project\jni가 되게 한다. NDK는 jni라는 이름의 폴더에서 네이티브 라이브러리 소스 파일들을 찾는다.

여기까지 마쳤다면 <NDK_R3>\apps\gltest-jni 폴더가 그림 B-1과 같은 모습일 것이다.

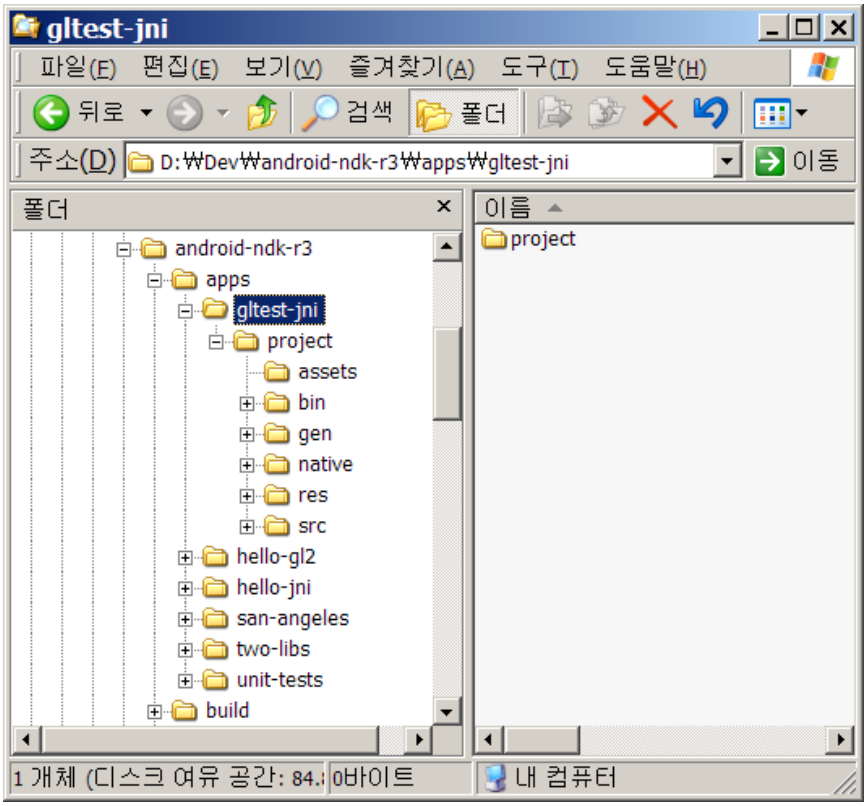

그림 B.1. 폴더 구조 준비

Eclipse 프로젝트 준비

다음으로 할 일은 준비된 프로젝트 폴더를 Eclipse의 작업공간으로 도입하고 네이티브 라이브러리의 빌드에 관련된 프로젝트 설정을 조정하는 것이다.

5. Eclipse를 띄우고, 주 메뉴의 File - Import를 이용해서 <NDK_R3>\apps\gltest-jni\project를 작업 공간에 도입한다(그림 B-2). 이때 Copy projects into workspace는 체크하지 말기 바란다. 그래야 <NDK_R3>\app\gltest-jni에서 계속해서 작업을 진행할 수 있다.

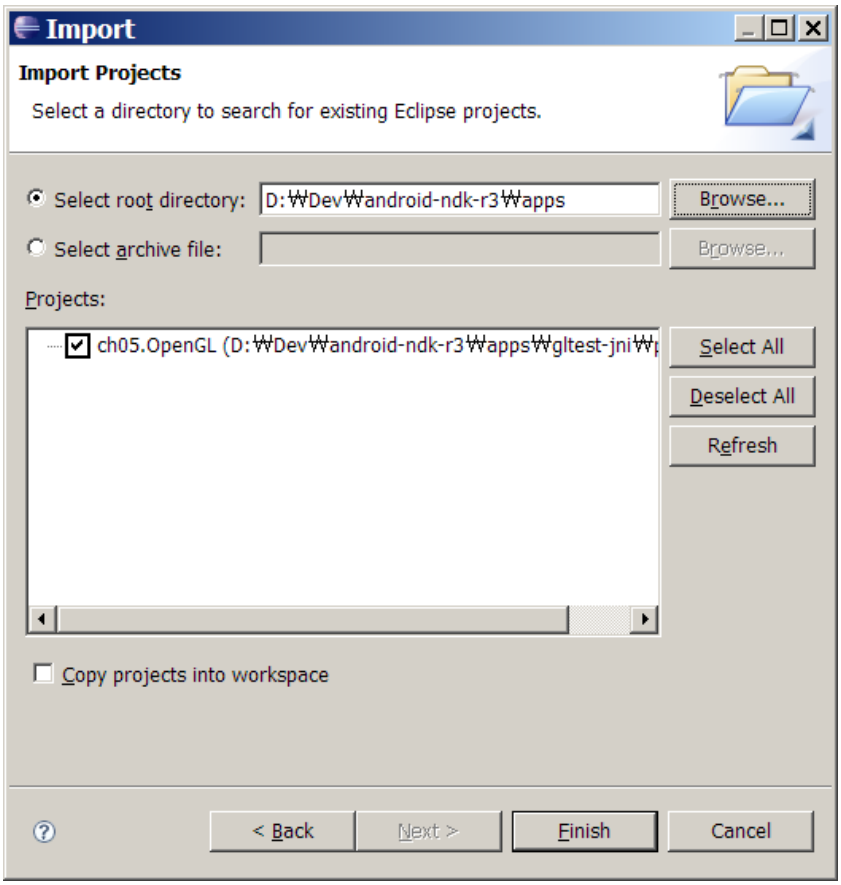

그림 B.2. Eclipse로 프로젝트 도입

6. 다음으로, 프로젝트의 빌드 대상을 SDK 1.6 이상으로 설정한다. ch05.OpenGL 프로젝트가 선택된 상태에서 주 메뉴 Project - Properties - Android를 선택하고 1.6 이상의 대상을 선택하면 된다. NDK는 프로젝트의 빌드 대상이 1.6 (API 수준 4) 이상일 때에만 OpenGL 라이브러리를 제공하기 때문에 이 과정이 반드시 필요하다.

7. 응용프로그램의 최소 SDK 설정을 방금 변경한 빌드 대상에 맞게 갱신한다. 6번에서 1.6을 선택했다면, AndroidManifest.xml의 <uses-sdk>의 android:min SdkVersion 요소를 4로 설정하면 된다. 다음은 수정된 AndroidManifest.xml 파일이다.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="opengl.test"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon"
                 android:label="@string/app_name">
        <activity android:name=".NativeGLActivity"
                  android:label="OpenGL Native">
            <intent-filter>
```

```
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".JavaGLActivity"
                  android:label="OpenGL Java">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="4"/>
</manifest>
```

최소 SDK 설정뿐만 아니라 두 <activity> 요소들의 순서도 변했는데, 네이티브 라이브러리의 빌드와 직접 관련된 것은 아니지만 이렇게 하면 응용프로그램을 기기에 올릴 때마다 매번 OpenGL Java를 닫고 홈에서 다시 OpenGL Native를 실행할 필요가 없다.

네이티브 라이브러리의 컴파일 및 링크

이제 네이티브 라이브러리를 실제로 컴파일하는 과정으로 들어가자. javah 명령으로 JNI 헤더들은 이미 만들어 두었다고 가정한다.

8. Eclipse 또는 즐겨 사용하는 텍스트 편집기를 이용해서 다음 두 줄을 담은 텍스트 파일을 만들고, <NDK_R3>\apps\gltest-jni 폴더에 Application.mk라는 이름으로 저장한다.

```
APP_PROJECT_PATH := $(call my-dir)/project
APP_MODULES      := gltest-jni
```

여기서 중요한 것은 APP_MODULES로, 이 변수에 설정된 이름(지금 예에서는 gltest- jni)이 이후에도 계속 등장한다.

9. 다음과 같은 내용을 담은 텍스트 파일을 만들어서 <NDK_R3>\apps\gltest-jni\ project \jni에 Android.mk라는 이름으로 저장한다.

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE    := gltest-jni
LOCAL_CFLAGS    := -Wall -O2 -fpic
LOCAL_SRC_FILES := cuberenderer.c cube.c
LOCAL_LDLIBS    := -lGLESv1_CM
include $(BUILD_SHARED_LIBRARY)
```

이 파일은 NDK r3에 기본으로 포함된 예제들 중 하나인 San Angeles의 Android. mk를 기본 틀로 하고 제5장 예제의 Makefile을 참고해서, 그리고 약간의 시행착오를 거쳐서 만들어낸 것이다. include $(BUILD_SHARED_LIBRARY)에 의해 실질적인 컴파일 및 링크 과정이 실행된다. LOCAL_PATH, LOCAL_MODULE, LOCAL_CFLAGS 등 주요 변수들에 대해서는 잠시 후에 간단히 설명하겠다.

10. 필요하다면 JNI 헤더들을 생성한다. 웹에서 내려 받은 예제 코드에 미리 생성된 헤더들이 포함되어 있고, 또 본문의 예제를 충실히 따라 했다면 이미 만들어져 있겠지만, 혹시 지워졌거나 다시 만들고 싶을 수도 있다. javah 명령이 시스템 경로에 포함되어 있다고 할 때, 현재 디렉터리가 <NDK_R3>\apps\ gltest-jni\project\jni인 상태에서 다음 명령을 실행하면 된다.

```
> javah -classpath ../bin -d include opengl.jni.Natives
```

<NDK_R3>\apps\gltest-jni\project\jni\include 폴더에 두 개의 .h 파일이 만들어졌으면 성공이다.

11.이제 Cygwin 콘솔을 띄우고(Cygwin이 설치된 곳의 cygwin.bat을 실행), <NDK_R3> 폴더로 가서 다음 명령을 실행한다. 아래는 NDK r3이 D:\dev\android- ndk-r3에 설치되어 있다고 가정한 것이다. 참고로 Windows의 C:\Some\Folder는 Cygwin에서 /cygdrive/c/Some/Folder에 해당한다.

```
$ cd /cygdrive/d/dev/android-ndk-r3
$ make APP=gltest-jni
```

여기서 APP= 다음에 지정된 gltest-jni는 <NDK_R3>\apps\ 폴더의 해당 폴더 이름이다. 이번 예에서는 폴더 이름과 모듈 이름(APP_MODULES 등)이 동일하지만, 꼭 이렇게 일치시켜야 하는 것은 아니다(다음 절에 다른 이름들을 사용하는 예가 나온다).

make 명령의 출력 마지막 부분에 다음과 같은 줄들이 있으면 성공인 것이다. 만일 실패했다면 오류 메시지들을 잘 살펴보기 바란다. 소스 코드 자체에 문제는 없으므로(경고 메시지들이 좀 나오긴 하지만), 대부분의 경우 문제의 원인은 프로젝트 경로나 파일 이름, 모듈 이름의 오타일 것이다.

```
.
.
.
Compile thumb  : gltest-jni <= apps/gltest-jni/project/jni/cube.c
SharedLibrary  : libgltest-jni.so
Install        : libgltest-jni.so => apps/gltest-jni/project/libs/armeabi
```

Eclipse의 패키지 탐색기에서 프로젝트의 libs/armeabi 폴더에 libgltest-jni.so가 추가되었는지도 점검하기 바란다(그림 B-3).

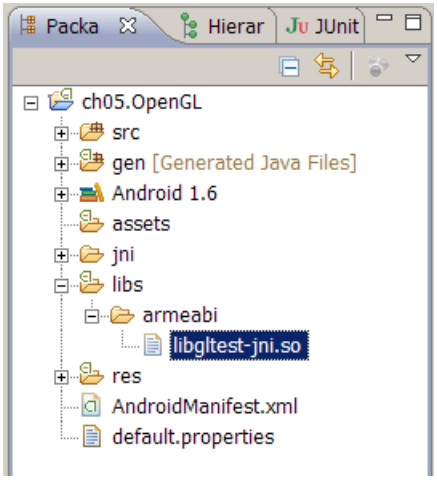

그림 B.3. Eclipse 프로젝트에 추가된 DSO 파일

다음 과정으로 넘어가기 전에, 단계 9에서 만든 Android.mk의 변수들을 좀 더 살펴보자.

- LOCAL_PATH는 이 Android.mk에 언급된 여러 지역 파일들의 기준 경로이다. 이 변수의 값으로 쓰인 $(call my-dir)는 현재 폴더(이 Android.mk 파일이 있는 폴더)를 뜻한다. 이에 의해, make는 LOCAL_SRC_FILES에 나열된 소스 파일들을 현재 폴더에서 찾는다. 다음 절에는 현재 폴더의 한 하위 폴더를 LOCAL_PATH로 설정하는 예가 나온다.

- LOCAL_MODULE이 제일 중요하다. 이 변수의 값은 make 명령의 주된 빌드 대상 이름으로 쓰이며, 출력 파일(공유 라이브러리 파일)의 이름에도 포함된다. 이 이름은 Application.mk의 APP_MODULES에 지정된 이름과 반드시 일치해야 한다.

- LOCAL_CFLAGS은 컴파일 플래그들로, 현재 라이브러리에 필요한 것들만 지정하면 된다. 원래의 Makefile의 CCFLAGS에 있던 -DNORMALUNIX 등은 따로 지정하지 않아도 된다(적어도 이 예에서는).

- LOCAL_SRC_FILES은 컴파일할 C 소스 파일들을 나열한 것으로, 원래 예제의 Makefile의 MAIN_OBJS에 나열된 목적 파일들의 확장자만 바꾼 것이다.
- 마지막으로 LOCAL_LDLIBS은 이 라이브러리에 링크할 외부 라이브러리들을 지정하는 것으로, 지금 예의 경우 OpenGL ES 라이브러리가 필요하므로 -lGLESv1_CM를 지정했다.

<NDK_R3>/docs의 ANDROID-MK.TXT와 APPLICATION-MK.TXT에 Android.mk와 Application.mk에 대한 좀 더 자세한 내용이 나와 있다. <NDK_R3>/docs의 다른 텍스트 문서들도 유용한 정보를 담고 있으므로 읽어보기 바란다.

Java 소스 수정 및 실행

마지막으로, Java 코드 중 네이티브 라이브러리를 적재하는 부분을 NDK 관례에 맞게 수정해야 한다. 고칠 부분은 네이티브 버전의 주 활동 클래스인 NativeGLActivity 클래스(opengl.test.NativeGLActivity.java)의 도입부이다. 기존 코드에서는 공유 라이브러리 파일의 절대 경로와 System.load() 메서드를 이용했지만, 이제는 공유 라이브러리 이름과 System.loadLibrary() 메서드를 사용한다. 굵은 글씨체로 된 부분이 변한 부분이다.

```
public class NativeGLActivity extends Activity
{
 private GLSurfaceView mGLSurfaceView;

 {
  final String LIB_NAME = "gltest-jni";

  System.out.println("Loading JNI lib using name: " + LIB_NAME);
  System.loadLibrary(LIB_NAME);
 }
```

적재할 라이브러리 이름이 앞에 나온 두 Makefile 파일(.mk)들에 쓰인 모듈 이름과 일치함을 주목하기 바란다.

이제 Eclipse에서 평소대로 응용프로그램을 에뮬레이터나 기기에 올려서 실행해 보기 바란다. LogCat 창에 다음과 같은 로그들이 뜨면 제대로 된 것이다.

```
INFO/System.out(3730): Loading JNI lib using name: gltest-jni
DEBUG/dalvikvm(3730): Trying to load lib /data/data/opengl.test/lib/libgltest-
jni.so 0x437aab98
DEBUG/dalvikvm(3730): Added shared lib /data/data/opengl.test/lib/libgltest-jni
.so 0x437aab98
DEBUG/dalvikvm(3730): No JNI_OnLoad found in /data/data/opengl.test/lib/libglte
st-jni.so 0x437aab98
INFO/System.out(3730): GLSurfaceView::setRenderer setting natives listener
DEBUG/EGL(3730): requestGPU() failed
ERROR/libEGL(3730): h/w accelerated eglGetDisplay() failed (EGL_SUCCESS)
DEBUG/ddm-heap(3723): Got feature list request
INFO/System.out(3730): Vendor:Android
INFO/System.out(3730): Renderer:Android PixelFlinger 1.0
INFO/System.out(3730): Version:OpenGL ES-CM 1.0
```

NDK r3으로 제6장 Wolfenstein 3D 컴파일하기

부록 A에 NDK 1.5로 제6장 Wolfenstein 3D의 네이티브 라이브러리를 컴파일, 링크하는 방법이 나왔는데, 현재 NDK 1.5는 구하기가 힘들고(안드로이드 개발자 사이트에서 파일이 아예 사라졌다) NDK r3을 사용하자니 폴더 구조 등이 변했기 때문에 그대로 따라 하기가 힘들었을 것이다. 그럼 제6장 예제의 네이티브 라이브러리를 NDK r3을 이용해서 빌드하는 방법을 살펴보자. 이전 절에 나온 제5장 예제의 과정과 거의 같으므로, 좀 더 간결하게 이야기하겠다.

폴더 구조 준비

이전 예제에서와 마찬가지로, 가장 먼저 할 일은 <NDK_R3>의 apps 폴더에 Wolf3D 예제를 위한 폴더 구조를 만들고 프로젝트 파일들을 복사하는 것이다. 구체적인 과정은 다음과 같다.

1. <NDK_R3>\apps에 wolf3d라는 이름의 폴더를 만들고, 웹 예제 코드의 ch06. Wolf3D.SW 폴더를 <NDK_R3>\apps\wolf3d 폴더 안에 복사한다.

2. <NDK_R3>\apps\wolf3d\wolf3d\ch06.Wolf3D.SW 폴더의 이름을 project로 바꾸고, <NDK_R3>\apps\wolf3d\project\native 폴더의 이름을 jni로 바꾼다.

Eclipse 프로젝트 준비

다음으로 할 일은 준비된 프로젝트 폴더를 Eclipse의 작업공간으로 도입하는 것이다. Wolf3D 예제는 OpenGL을 사용하지 않으므로, 프로젝트 빌드 대상은 굳이 변경할 필요가 없다.

3. Eclipse를 띄우고 주 메뉴의 File - Import를 이용해서 <NDK_R3>\apps\wolf3d\ project를 작업 공간에 도입한다. 이전과 마찬가지 방법으로 Copy projects into workspace는 체크하지 않도록 한다.

네이티브 라이브러리의 컴파일 및 링크

4. 다음 내용을 담은 텍스트 파일을 <NDK_R3>\apps\wolf3d 폴더에 Application. mk라는 이름으로 저장한다.

```
APP_PROJECT_PATH := $(call my-dir)/project
APP_MODULES      := wolf_jni
```

wolf_jni라는 이름은 Java 쪽에서 라이브러리를 적재하는 데 쓰이는 문자열 상수(wolf.util.WoolfTools의 WOLF_LIB)와 일치한다. 이렇게 이름을 일치시킨 덕분에 이후에 Java 쪽 코드를 수정할 필요가 없다. 이 이름이 폴더 이름 wolf3d와는 다름을 주목할 것. 이렇게 다른 이름을 무방하다. 혼란을 피하려면 같은 이름을 사용하는 게 좋겠지만, 예를 들어 버전 관리나 호환성 관리를 위해 라이브러리 이름은 고정시키되 폴더 이름에는 버전 번호 등을 붙여야 하는 경우도 있을 것이다.

5. 다음 내용을 담은 텍스트 파일을 만들어서 <NDK_R3>\apps\wolf3d\project\ jni에 Android.mk라는 이름으로 저장한다.

```
LOCAL_PATH := $(call my-dir)/gp2xwolf3d

include $(CLEAR_VARS)

LOCAL_MODULE  := wolf_jni

INC := -Iinclude
OPTS := -O6 -ffast-math -fexpensive-optimizations \
     -funroll-loops -fomit-frame-pointer

LOCAL_CFLAGS := $(OPTS) $(INC)
```

```
LOCAL_SRC_FILES := objs.c misc.c id_ca.c id_vh.c id_us.c \
  wl_act1.c wl_act2.c wl_act3.c wl_agent.c wl_game.c \
  wl_inter.c wl_menu.c wl_play.c wl_state.c wl_text.c wl_main.c \
  wl_debug.c vi_comm.c sd_comm.c \
  wl_draw.c jni_wolf.c vi_null.c sd_null.c

include $(BUILD_SHARED_LIBRARY)
```

이전 절의 예와 원래 예제의 Makefile를 참고하면 그리 어렵지 않게 이해할 수 있을 것이다. 한 가지 주목할 것은 LOCAL_PATH 설정인데, 잠시 후에 좀 더 이야기하겠다.

6. 필요하다면 javah 명령으로 JNI 헤더들을 생성한다. native 폴더 대신 jni 폴더에서 javah 명령을 실행한다는 점만 빼면 본문에서와 동일하다.

7. 마지막으로, Cygwin 콘솔에서 make 명령을 실행한다. 아래는 NDK r3이 D:\ dev\android-ndk-r3에 설치되어 있다고 가정한 것이다.

```
$ cd /cygdrive/d/dev/android-ndk-r3
$ make APP=wolf3d
```

이전과 마찬가지로, make의 출력 마지막에 다음과 같은 줄들이 있으면 성공인 것이다.

```
.
.
.
Compile thumb  : wolf3d <= apps/wolf3d/project/jni/cube.c
SharedLibrary  : libwolf3d.so
Install        : libwolf3d.so => apps/wolf3d/project/libs/armeabi
```

Eclipse의 패키지 탐색기에서 프로젝트의 libs/armeabi 폴더에 libwolf3d.so가 추가되었는지도 점검하기 바란다.

LOCAL_PATH 설정이 이전 예제와 좀 다른데, objs.c 등의 실제 소스 파일들이 현재 폴더(jni)가 아니라 현재 폴더의 한 하위 폴더인 gp2xwolf3d에 있기 때문에 이렇게 한 것이다(사실 application.mk에서도 이런 방식으로 APP_PROJECT_PATH를 현재 폴더의 한 하위 폴더로 설정했었다). gp2xwolf3d 폴더의 소스 파일들을 jni로 모두 옮기거나(부록 A의 예에서처럼) LOCAL_SRC_FILES 변수 설정 부분에 일일이 gp2xwolf3d/objs.c 형태로 폴더 이름을 덧붙이는 것보다는 이 방식이 더 깔끔하다.

컴파일할 소스 파일이 현재 폴더와 하위 폴더 모두에 있는 경우라면 좀 더 본격적인 해법이 필요할 것이다(하위 폴더에 개별적인 Makefile을 두고, include 명령으로 그 Makefile을 포함시키는 등). 더 나아가서, make의 내장 명령 wildcard 등을 이용해서 소스 파일들을 좀 더 간편하게 지정하는 것도 가능한데, 자세한 내용은 GNU make 매뉴얼(http://www.gnu.org/software/make/manual/html_node/index.html) 등을 참고하기 바란다.

실행

이번 경우에는 Java 소스를 고칠 필요가 없다. Java 소스가 애초에 라이브러리 이름과 System.loadLibrary()를 이용해서 네이티브 라이브러리를 적재하며, 앞에서 mk 파일들을 만들 때 Java 소스에 쓰이는 라이브러리 이름을 그대로 사용했기 때문이다. 지금까지의 과정이 모두 성공했다면, Eclipse에서 평소대로 응용프로그램을 에뮬레이터나 기기에 올려서 실행할 수 있을 것이다.