Google

intel Software

Android on Everything

Smooth Development of Cross-platform Native Android Games

Ian Ni-Lewis
Google

Orion Granatir
Intel Corporation

www.intel.com/software/gdc

Agenda

- How to abstract out hardware and when does it makes sense to do so
- Understand the memory alignment differences between ARM and x86, and how to exploit them
- How to use processor specific SIMD intrinsics (NEON and SSE)
- Understand the best compiler flags for different ABIs
- How to properly use OpenGL extensions
- How to package multiple platform specific .so files in a .APK file ("fat binaries")

OpenGL ES already abstracts the GPU

PowerVR	Tegra	Mali
OpenGL ES 1.1/2.0	OpenGL ES 1.1/2.0	OpenGL ES 1.1/2.0
OpenGL ES SL	OpenGL ES SL	OpenGL ES SL
PVRTC/ETC	S3TC (DX Style)/ETC	ETC

← Beware multiple texture compression formats

- Remember to check OpenGL extensions
  - Search for extension name in glGetString(GL_EXTENSIONS)
- Intel products are PowerVR based

Let the NDK abstract the CPU

ARM v5	ARM v7a	x86	
32-bit	32-bit	32-bit	
little-endian	little-endian	little-endian	
Soft FP	Hardware FP	Hardware FP	
64-bit vars aligned	64-bit vars aligned	64-bit vars packed	← Normally not a problem…
None	NEON	SSE	← This will require porting…

Let the NDK do the hard work

- Include all ABIs in jni/Application.mk

Application.mk — Edited

```
APP_ABI := armeabi armeabi-v7a x86
```

OGLTest — bash — 62×35

```
Orions-MacBook-Air:OGLTest AE$ cat jni/Application.mk
APP_ABI := armeabi armeabi-v7a x86

Orions-MacBook-Air:OGLTest AE$ ndk-build
Compile++ thumb  : gl2jni <= gl_code.cpp
Compile thumb  : gl2jni <= App.c
Compile thumb  : gl2jni <= file.c
Compile thumb  : gl2jni <= log.c
Compile thumb  : gl2jni <= shader.c
StaticLibrary  : libstdc++.a
SharedLibrary  : libgl2jni.so
Install        : libgl2jni.so => libs/armeabi/libgl2jni.so
Compile++ thumb  : gl2jni <= gl_code.cpp
Compile thumb  : gl2jni <= App.c
Compile thumb  : gl2jni <= file.c
Compile thumb  : gl2jni <= log.c
Compile thumb  : gl2jni <= shader.c
SharedLibrary  : libgl2jni.so
Install        : libgl2jni.so => libs/armeabi-v7a/libgl2jni.so
Compile++ x86    : gl2jni <= gl_code.cpp
Compile x86    : gl2jni <= App.c
Compile x86    : gl2jni <= file.c
Compile x86    : gl2jni <= log.c
Compile x86    : gl2jni <= shader.c
SharedLibrary  : libgl2jni.so
Install        : libgl2jni.so => libs/x86/libgl2jni.so
Orions-MacBook-Air:OGLTest AE$
```

← Build ARM v5...

← Build ARM v7a...

← Build x86...

You can also use *APP_ABI := all* with the latest NDK

Determine CPU type at compile

- android_getCpuFamily will return the CPU type at runtime, but most features need to be determined at compile

- In Android.mk you can use
  ifeq ($(TARGET_ARCH_ABI),*desiredabi*) … endif
  for examples
  ifeq ($(TARGET_ARCH_ABI),armeabi-v7a) … endif

- In source you can use…

Avoid memory alignment issues

One difference between x86 and ARM is the memory alignment requirements for data.
Let's look at a simple example:

```
#define  OFFSET(x, y)  &((x*)0)->y

struct TestStruct
{
    int mVar1;
    int mVar2;
    int mVar3;
};

JNIEXPORT void JNICALL Java_com_intel_TESTJNILib_run( JNIEnv* env, jobject thiz )
{
    LOGI("TestStruct (size: %d)", sizeof(TestStruct));
    LOGI("-- Var1 offset: %d", OFFSET( TestStruct, mVar1 ) );
    LOGI("-- Var2 offset: %d", OFFSET( TestStruct, mVar2 ) );
    LOGI("-- Var3 offset: %d", OFFSET( TestStruct, mVar3 ) );
}
```

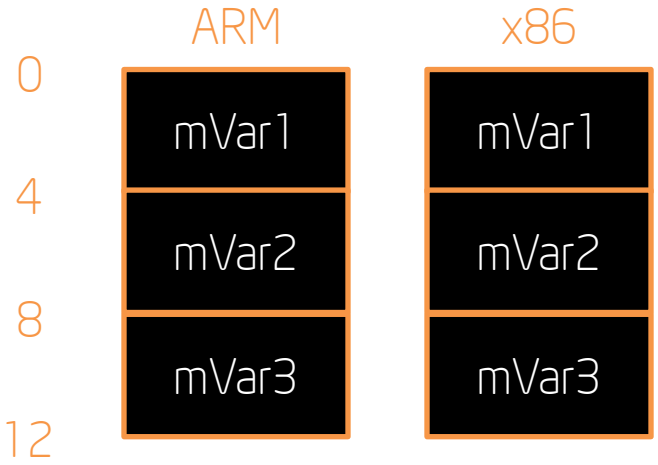
The output for this program isn't too surprising:

ARM

```
I/libtestjni( 5025): TestStruct (size: 12)
I/libtestjni( 5025): -- Var1 offset: 0
I/libtestjni( 5025): -- Var2 offset: 4
I/libtestjni( 5025): -- Var3 offset: 8
```

x86

```
I/libtestjni( 4175): TestStruct (size: 12)
I/libtestjni( 4175): -- Var1 offset: 0
I/libtestjni( 4175): -- Var2 offset: 4
I/libtestjni( 4175): -- Var3 offset: 8
```



Avoid memory alignment issues

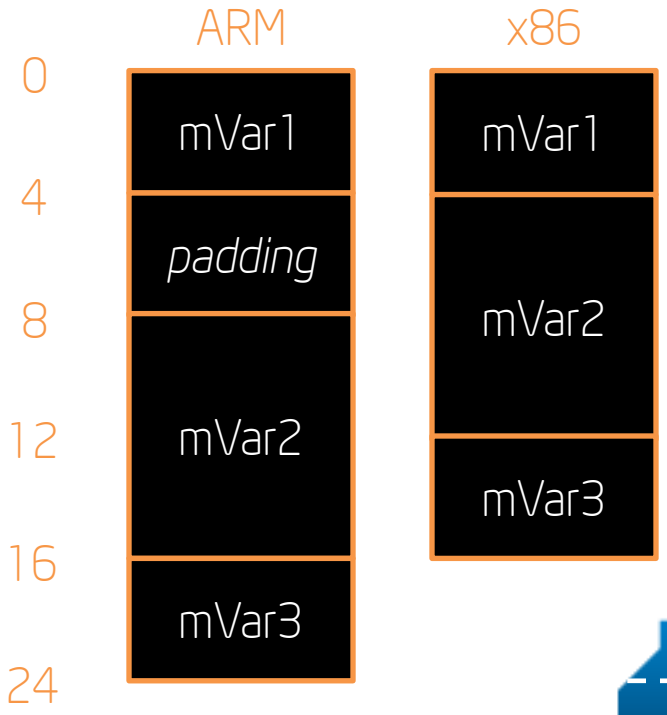
But now, let's change TestStruct to the following:

```
struct TestStruct
{
   int mVar1;
   long long mVar2;
   int mVar3;
};
```

The output is now:

ARM
I/libtestjni(4675): TestStruct (size: 24)
I/libtestjni(4675): -- Var1 offset: 0
I/libtestjni(4675): -- Var2 offset: 8
I/libtestjni(4675): -- Var3 offset: 16

x86
I/libtestjni(4079): TestStruct (size: 16)
I/libtestjni(4079): -- Var1 offset: 0
I/libtestjni(4079): -- Var2 offset: 4
I/libtestjni(4079): -- Var3 offset: 12



Avoid memory alignment issues

The 64-bit mVar2 results in different layout for TestStruct. This is because ARM requires 8-byte alignment for 64-bit variables like mVar2. In most cases, this won't cause problems because building for x86 vs ARM requires a full rebuild.

If this is an issue, you can force the same alignment between ARM and x86 Android:

```
struct TestStruct
{
   int mVar1;
   long long mVar2 __attribute__ ((aligned(8)));
   int mVar3;
};
```

NEON vs SSE

NEON	SSE
128-bit wide	128-bit wide
16 registers	8 register
ASM/Intrinsics	ASM/Intrinsics
Optional feature	On all platforms

- NEON is an *optional* feature on armeabi-v7a
  - NOT ALL ARMv7-BASED ANDROID DEVICES WILL SUPPORT NEON
- All Atom processors support SSSE3
  - SSE is the same as the desktop, so you probably already have code to port

How to build SIMD support

- For ARM, add NEON support to Android.mk:

```
ifeq ($(TARGET_ARCH_ABI),armeabi-v7a)
    include $(CLEAR_VARS)
    LOCAL_MODULE    := mylib-neon
    LOCAL_SRC_FILES := mylib-neon.c
    LOCAL_ARM_NEON  := true
    include $(BUILD_STATIC_LIBRARY)
endif # TARGET_ARCH_ABI == armeabi-v7a
```

- For x86, SSE support is always available.

Properly detect NEON support


```
#include <cpu-features.h>
...

#ifdef __arm__
if (android_getCpuFamily() == ANDROID_CPU_FAMILY_ARM &&
    (android_getCpuFeatures() & ANDROID_CPU_ARM_FEATURE_NEON) != 0)
{
    // use NEON-optimized routines
    ...
}
else
{
    // use non-NEON fallback routines instead
    ...
}
#endif
```

Build and load separate libraries for standard C, NEON, and SSE at runtime.

The right compiler flags make a big difference!!

- Again, the NDK does all the hard work
- You only need to care about this if you are using the standalone toolchain.
  - Start with the NDK flags
- -ffast-math is generally good for games
- -Make sure you have the right -mtune and -march
- Using things like -mssse3 is a good addition for x86

Compiler flags (android-ndk-r7\toolchains)

ARM	ARM v7a	x86
-fpic -ffunction-sections -funwind-tables -fstack-protector -march=armv5te -mtune=xscale -msoft-float -O2 -fomit-frame-pointer -fstrict-aliasing -funswitch-loops -finline-limit=300	-fpic -ffunction-sections -funwind-tables -fstack-protector -march=armv7-a -mfloat-abi=softfp -mfpu=vfp -O2 -fomit-frame-pointer -fstrict-aliasing -funswitch-loops -finline-limit=300	-ffunction-sections -funwind-tables -O2 -fomit-frame-pointer -fstrict-aliasing -funswitch-loops -finline-limit=300

Fat Binaries

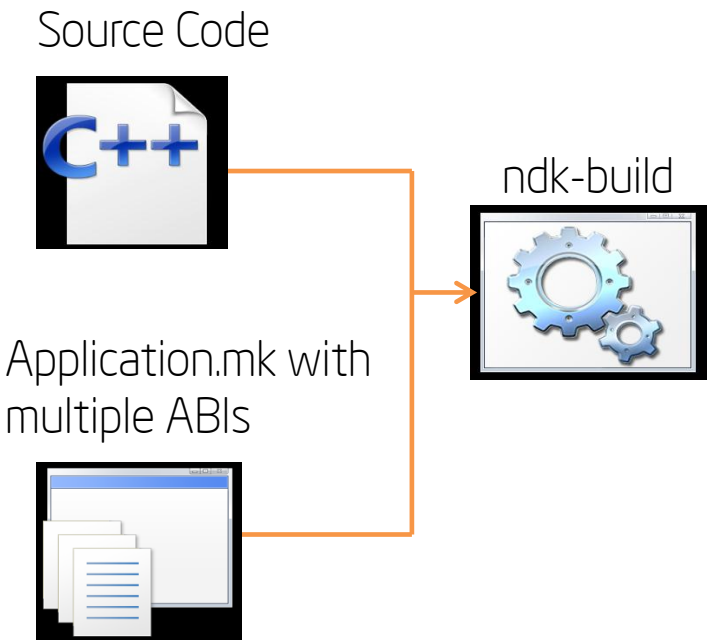
- Fat Binaries are .apk files that supports multiple ABIs
- Remember, this will increase the size of your .apk
- Let's look at how to create a fat binary...

Add desired ABIs to Application.mk

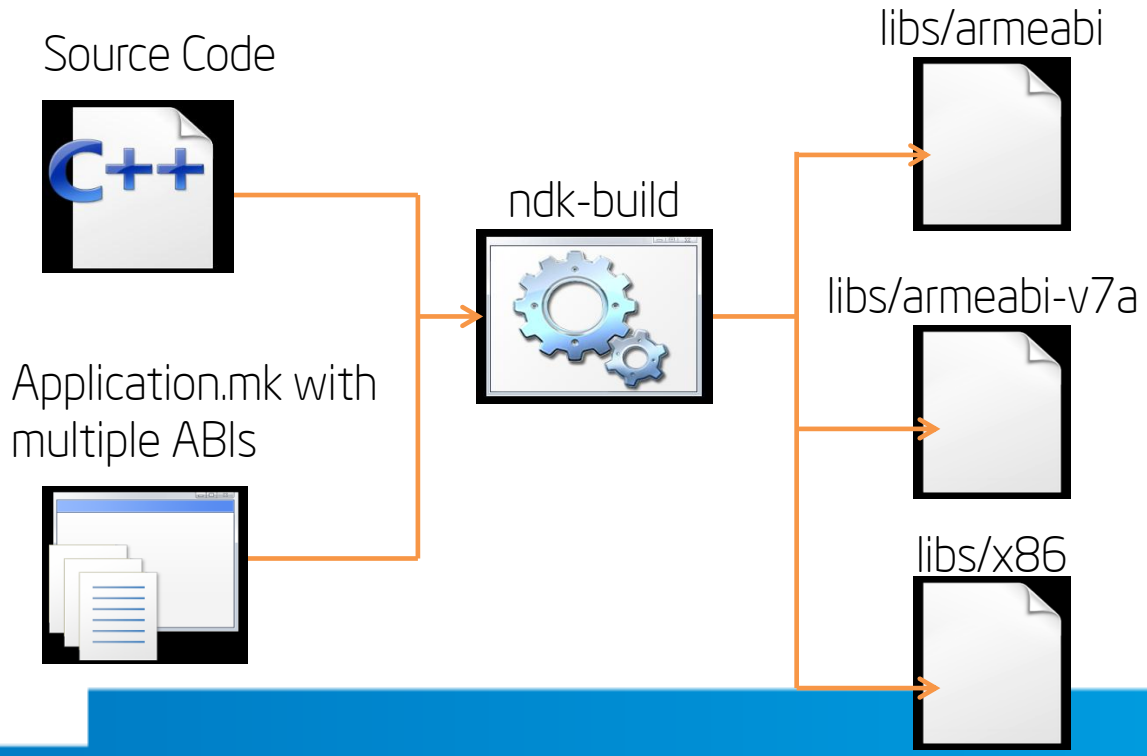
Source Code

Application.mk with multiple ABIs

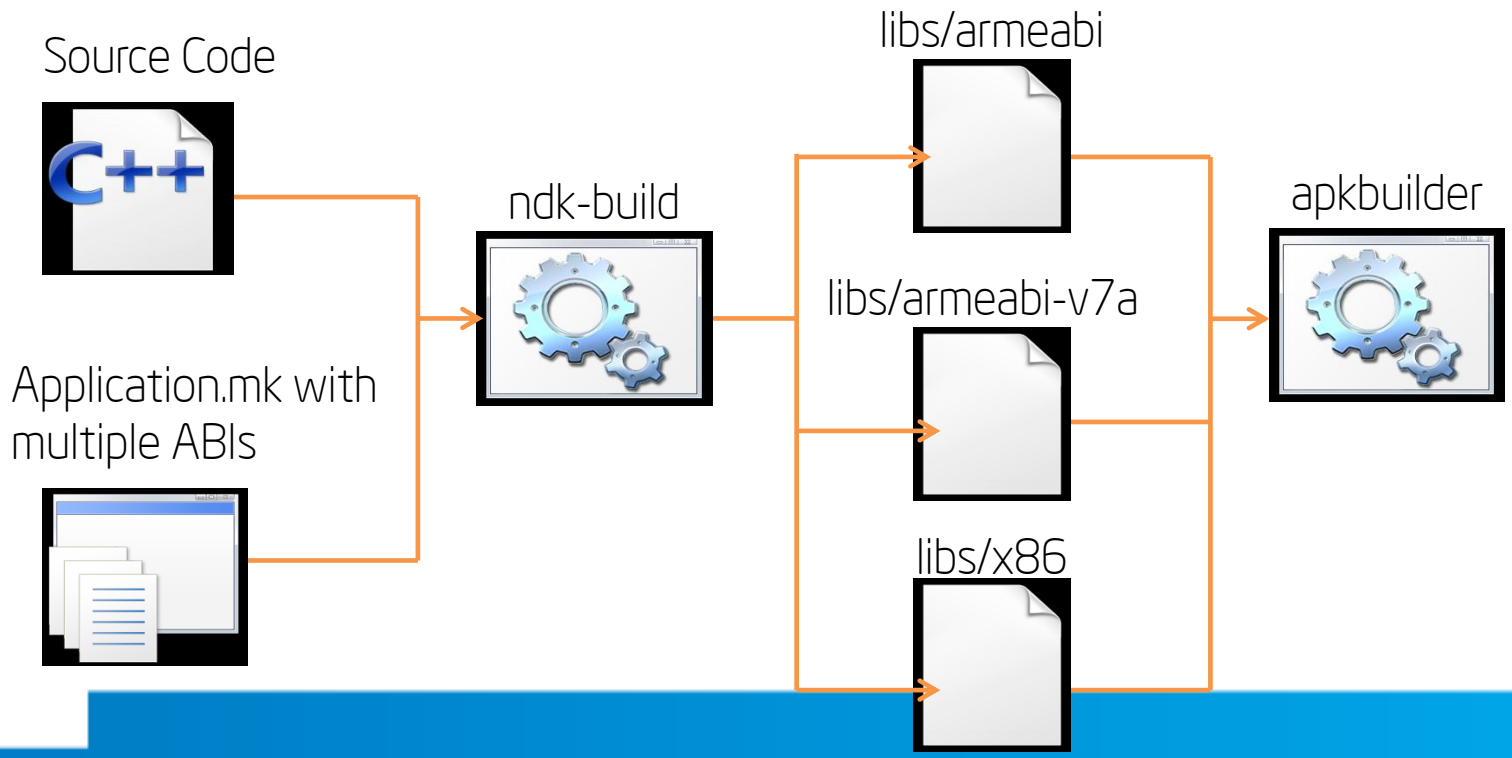
NDK will generate all libs



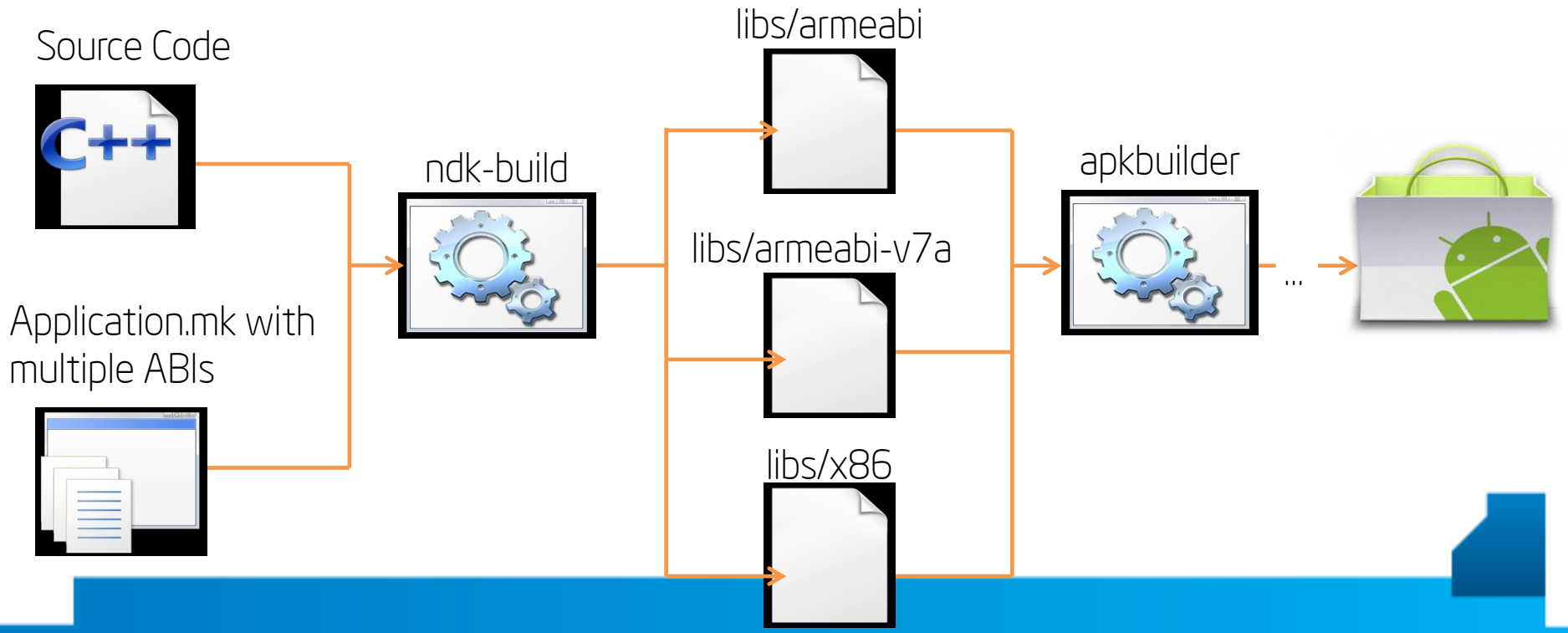
One .so is created per ABI



All .so files are included in the .apk



One .apk is hosted on the Market



Install will select the right .so

- Android Market will automatically filter applications based on ABI support

Case Study: Porting to x86 Android

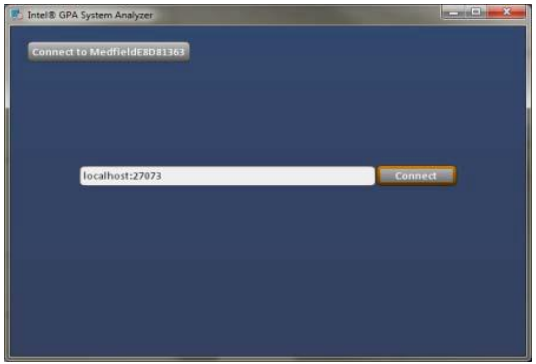
Goal: Port a major game engine to support ARM and x86 Android

Problem…	Solution…
Hardcoded to ARM (standalone) tools	Reworked scripts to include x86
NEON code for skinning	Used existing SSE code for PC
Pre-built libraries for ARM	Rebuild for x86 and link to proper version
rand() mismatch at link time	Inlining difference, wrote a stub for x86
Other bugs	Fixed in latest NDK!

Profile: Intel® Graphics Performance Analyzer

1. Install APK

2. On dev machine, run System Analyzer



3. Profile

Come see this at the Intel booth!!

Conclusions

- The NDK is your friend!! They already did most of the hard work.

- Go get the latest and start making Apps the run everywhere ☺

Your Single Source for Visual Computing Info, Articles, Samples, SDKs and Tools

- Free Downloads of Intel® Visual Computing Tools
- Code Samples
- Tech Articles
- Case Studies
- Forums
- Beta Programs

www.intel.com/software/vcsource

Q&A

Please fill out your evaluation forms

Win

- SSD drives
- Ultrabook in the Intel booth #1024
  Drawings: Wed/Thu @ 5:30pm, Fri @ 2:30pm

Legal Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Intel may make changes to specifications, product descriptions, and plans at any time, without notice.

The Intel processor and/or chipset products referenced in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

All dates provided are subject to change without notice. All dates specified are target dates, are provided for planning purposes only and are subject to change.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

* Other names and brands may be claimed as the property of others.

Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel® Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101

Backup

Case Study 2: Porting to x86 Android

Goal: Port a major game engine to support ARM and x86 Android

Problem…	Solution…
Hardcoded to ARM (standalone) tools	Reworked scripts to include x86
Memory alignment issue	Data is read differently at runtime
Compiler flags different	Copy from NDK for x86
rand() mismatch at link time	Rebuilding all fixed this issue
Other bugs	Fixed in latest NDK!


intel
Software