Part 2.2

Mobile Graphics Trends: Applications

Marco Agus, KAUST & CRS4

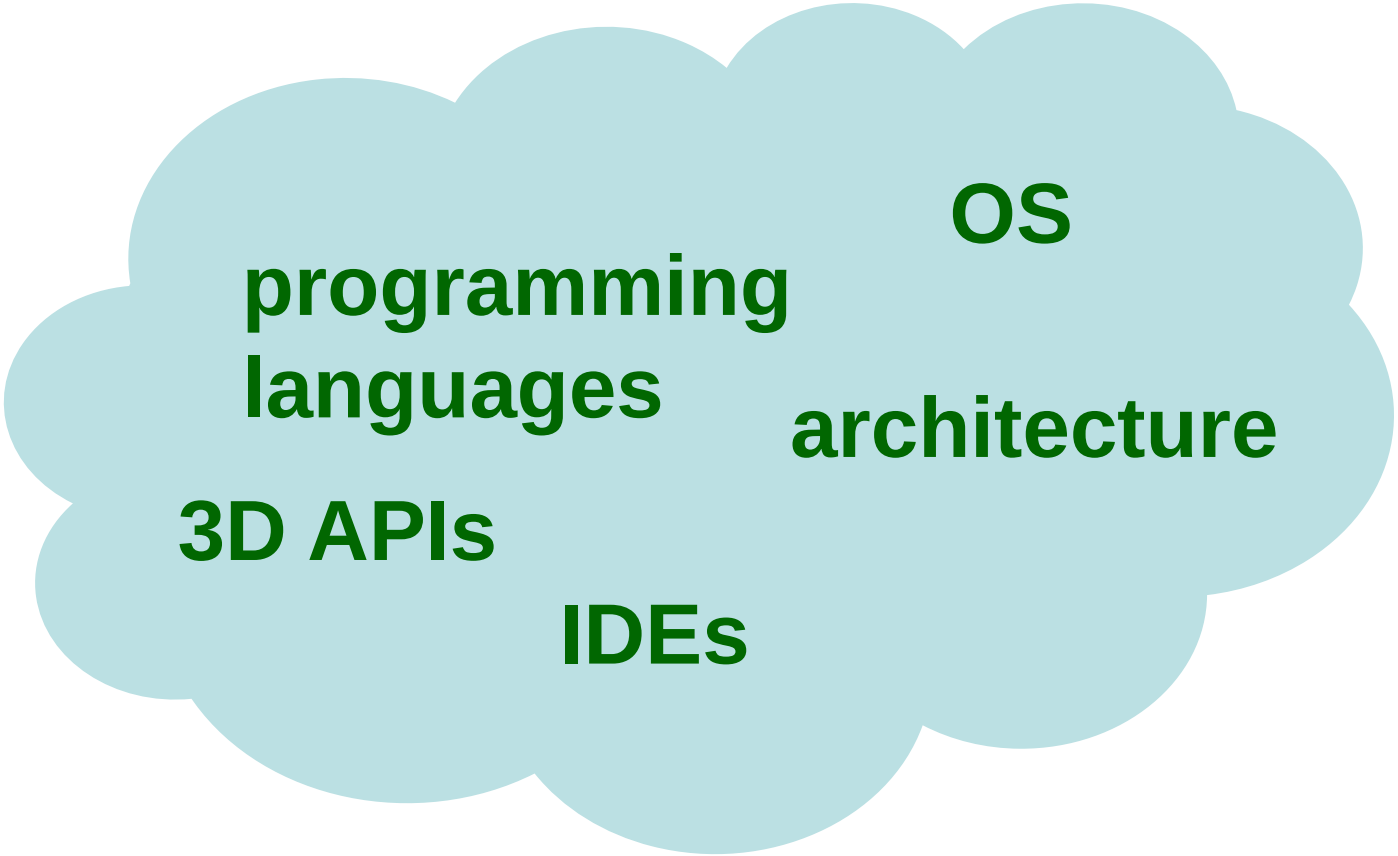
Part 3

Graphics development for mobile systems

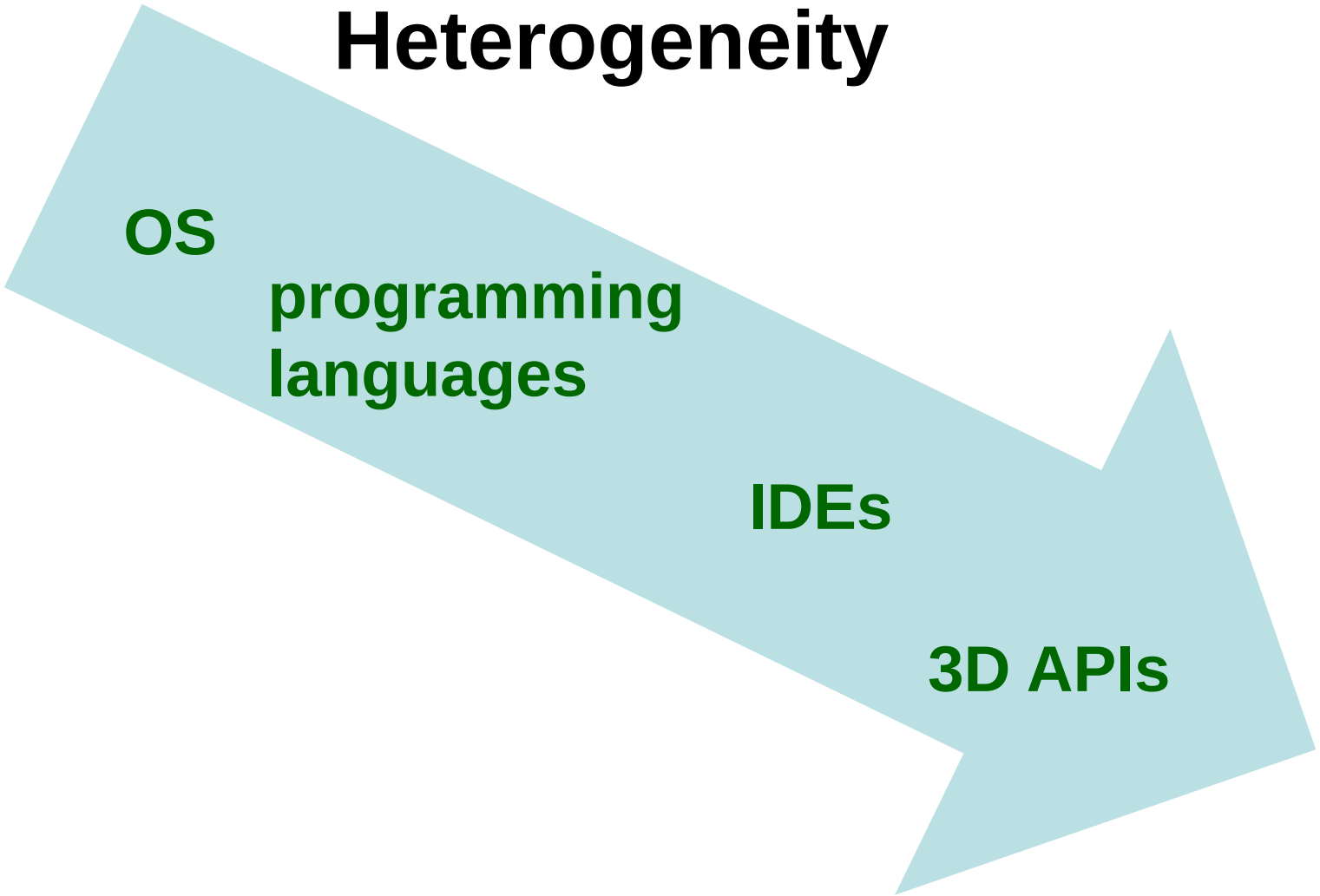
Marco Agus, KAUST & CRS4

Mobile Graphics

Heterogeneity



Mobile Graphics



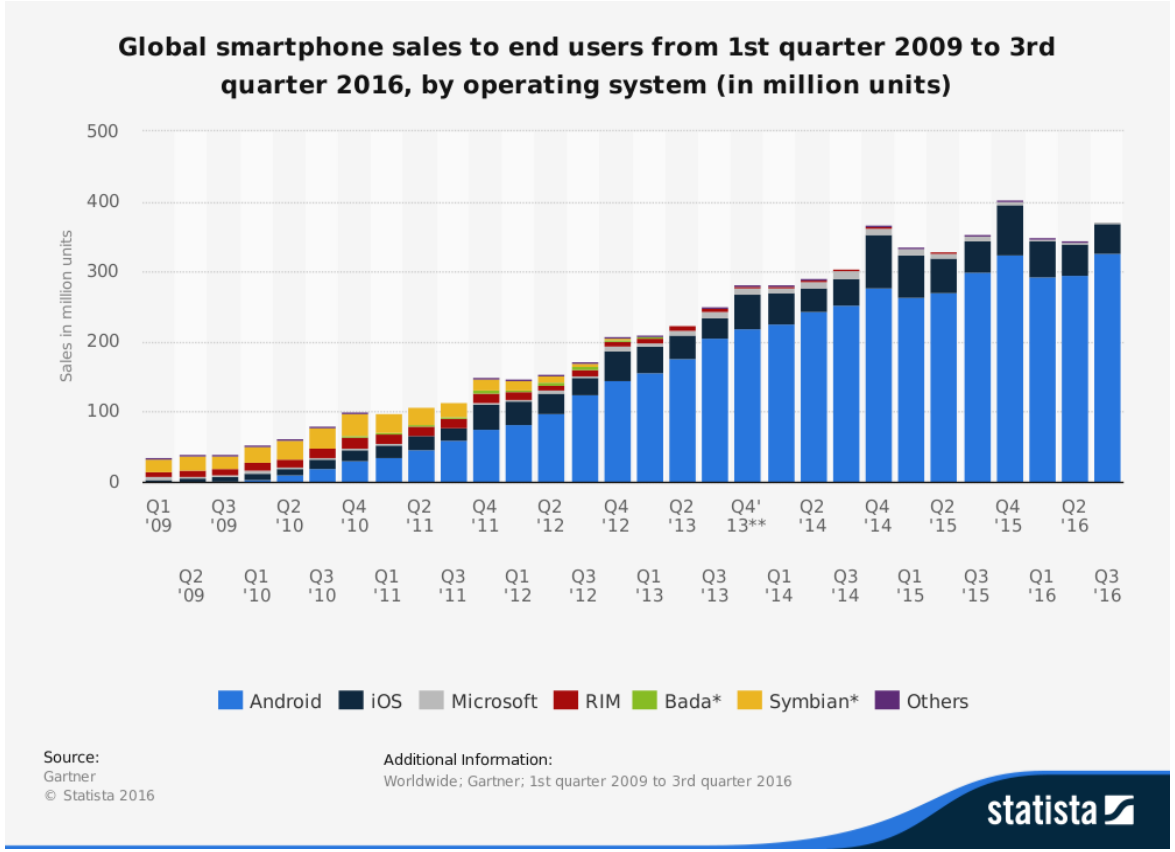
Mobile Graphics

- **OS**
  - Android
  - iOS
  - Windows Phone
  - Firefox OS, Ubuntu Phone, Tizen…
- **Programming Languages**
  - C++
  - Obj-C / Swift
  - Java
  - C# / Silverlight
  - Html5/JS/CSS
- **Architectures**
  - X86 (x86_64): Intel / AMD
  - ARM (32/64bit): ARM + (Qualcomm, Samsung, Apple, NVIDIA,…)
  - MIPS (32/64 bit): Ingenics, Imagination.
- **3D APIs**
  - OpenGL / GL ES
  - D3D / ANGLE
  - Metal / Mantle / Vulkan (GL Next)
- **Cross-development**
  - Qt
  - Marmalade / Xamarin /
  - Muio
  - Monogame / Shiva3D / Unity / UDK4 / Cocos2d-x

Operating Systems

Operating Systems

- **Linux based (Qt…)**
  - Ubuntu, Tizen, BBOS…
- **Web based (Cloud OS)**
  - ChromeOS, FirefoxOS, WebOS
- **Windows Phone**
- **iOS (~unix + COCOA)**
- **Android (JAVA VM)**

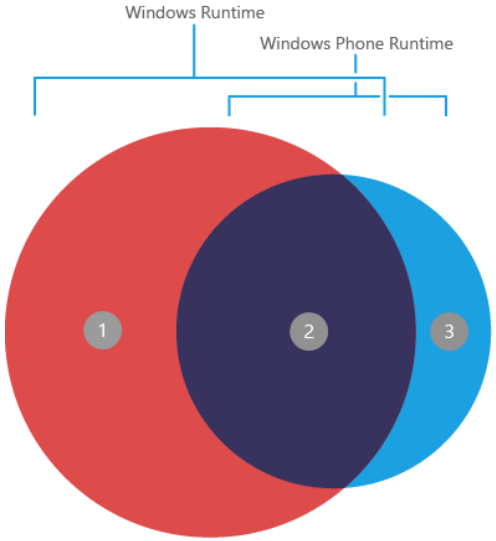


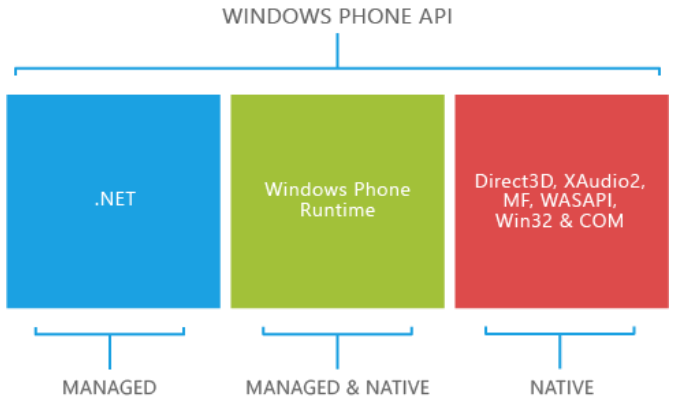
Development trends

- **Hard to follow the trends**
  - software does not follow hardware evolution
  - strong market oriented field where finance has strong impact on evolution
- **In general, for**
  - Mobile phones
    - Market drive towards Android, iOS
  - Tablets
    - Android, iOS, Windows 10
  - Embedded devices
    - Heterogenous (beyond the scopes of this course)
- **Here we focus on mobile phones and tablets**

Operating Systems

- **Windows 10**
  - Windows development – Visual Studio 2017
    - Good debugging / compiler / integration
  - Great integration and deployment
    - Universal Windows Platform (UWP)
  - API access
    - C#, VB.NET, and C++
  - 3D API
    - D3D
    - OpenGL access through ANGLE
  - Advantages
    - **Visual Studio, interoperability with iOS**
    - **HW is quite selected/homogeneous**
  - Disadvantages
    - **~OpenGL wrapper just recently!**





Operating Systems

- **iOS**
  - Development under MacOS
    - Xcode – good IDE/debug
    - **Clang** compiler!
  - API access
    - Objective-C, **swift**
  - Library programming
    - C++ support
  - Advantages:
    - **Homogeneous hardware** (biggest issues are resolution related)
    - **State-of-the-art CPU/GPU** (PowerVR SGX 54X/554, G6400)
    - **Good dev tools** (Xcode + Clang)
  - Inconvenients:
    - **Closed platform**
    - Requires **iDevice** for development/shipment (mostly)

Operating Systems

- **Android**
  - Development in Eclipse / AndroidStudio
    - Java-based – integrated debugging (non-trivial for NDK)
    - GCC / clang compilers
  - Advantages
    - Wide **variety** of hardware configurations (CPU/GPU)
    - Java based + C++ as dynamic library (JNI or NDK+NativeActivity)
    - Open source
    - Toolchain provided for Windows/Linux/MacOS (GCC + Clang)
    - Faster **access** to new hardware / functionality!
  - Inconvenients
    - **Heterogeneous** device base (hard to target all configurations)
    - **Not** so **integrated** IDE -- ~mixed pieces

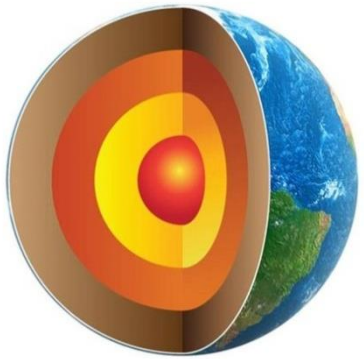
Operating Systems (comparison)

- **App development -- publishing**
  - WinPhone & iOS requires less effort for distribution
    - Easy to reach the whole user base
  - Android has a wide variety of configuration that require tuning
    - User base is typically reached in an incremental way (supporting more configs)
    - Many HW configurations (CPU/GPU) give more options to explore ☺
  - Windows has not yet the same market share
    - Variety of configurations

Programming Languages

- **C/C++**
  - Classic, performance, codebase, control
- **Objective C**
  - Bit different style (message based), well-documented API for iOS, mainly COCOA/iOS
- **Java**
  - Android is VM/JIT based, ~portability (API), well-known, extended, codebase
- **C#**
  - VM based, ~Java evolution, (Win, Android, iOS)
- **Swift**
  - Apple new language, simplicity, performance, easy, LLVM-based compilers
- **HTML5/JS**
  - Web technologies, extended, compatibility
- **Perl, Python, Ruby, D, GO (Google), Hack (facebook), …**
  - More options, not so popular ?

3D APIs



3D APIs

Cross Platform Challenge

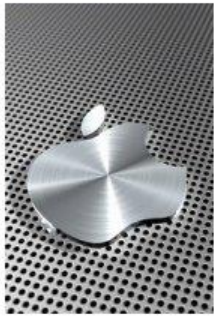
- An explicit API that is also cross-platform needs careful design

One family of GPUs

Mantle

One OS

Direct3D



One GPU on one OS

Metal

Vulkan™

OpenGL Next 5.0

KHRONOS GROUP

3D APIs

- **Direct 3D**
  - 3D API from MS for Win OS (XBOX)
  - ANGLE library provides GL support on top of D3D
- **Mantle**
  - AMD 3D API with Low-level access → **D3D12 | GL_NG**
- **Metal**
  - Apple 3D API with low-level access
- **OpenGL Desktop/ES/WebGL**
  - GL for embedded systems, now in version 3.2
    - GLES3.2 ~ GL4.5
- **GL Next Generation → Vulkan**
  - redesign to unify OpenGL and OpenGL ES into one common API (no backward compatibility)

3D APIs

- **Direct 3D**
  - Games on Windows (mostly) / XBOX
  - Define 3D functionality state-of-the-art
    - OpenGL typically following
    - 3D graphic cards highly collaborative
    - Multithread programming
  - Proprietary – closed source – **M$**
  - **Tested & stable – good support + tools**

- **Metal**
  - Apple 3D API with low-level access
  - Much in the way of **Mantle?**
    - buffer & image, command buffers, sync…
  - **Lean & mean → simple + ~flexible**

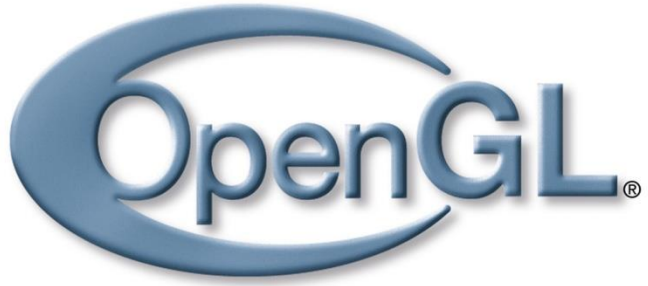
Win & Game research

Mac/iOS future ?

3D APIs

- **Mantle**
  - AMD effort – **low level – direct access** – 3D API
  - Direct control of memory (CPU/GPU) – multithreading done well
    - User-required synchronization
  - **API calls per frame <3k → 100K**
  - Resources: buffer & image ☺
  - Simplified driver → maintenance (vendors)
    - High level API/Framework/Engines will be developed ☺
  - Pipeline state
    - shaders + targets (depth/color…) + resources + geometry
  - **Command queues + synchronization**
    - Compute / Draw / DMA(mem. Copy)
  - Bindless – shaders can refer to state resources
  - **OpenGL NEXT seems to move into 'Mantle direction'**
  - **Direct 3D 12 already pursuing low-level access**

3D APIs



- **OpenGL (Desktop/ES/WebGL)**
  - Open / research / cross-platform
  - Lagging in front of D3D → Legacy support ☹
    - No more **FIXED PIPELINE (1992)**!! -- scientific visualization…
  - GLSL (2003)…GL 3.1(2009) → deprecation/no fixed pipeline
    - Compatibility profile → legacy again…(till GL 4)
    - Core profile
      - GLSL → shader required
      - VAO
        - » group of VBO
        - » we need a base VAO for using VBO!
      - Simplifying → VBO + GLSL only!

3D APIs

- OpenGL ES 1.1
  - Fixed pipeline – no glBegin/End – no GL_POLYGON -- VBO
- OpenGL ES 2 (OpenGL 1.5 + GLSL) ~ GL4.1
  - No fixed pipeline (shaders mandatory), ETC1 texture compress..
- OpenGL ES 3 ~ GL4.3
  - Occlusion queries + geometry instancing
  - 32bit integer/float in GLSL
  - Core 3D textures, depth textures, ETC2/EAC, many formats…
  - Uniform Buffer Objects (packed shader parameters)
- OpenGL ES 3.2 ~ GL4.5
  - Compute shaders (atomics, load/store)
  - Separate shader objects (reuse)
  - Indirect draw (shader culling…)
  - NO geometry/tessellation

3D APIs

- **Vulkan**
  - derived from and built upon components of AMD's Mantle API
  - with respect to OpenGL
    - lower level API, more balanced CPU/GPU usage, parallel tasking, work distribution across multiple CPU cores

OpenGL	**Vulkan**
Global state machine	No global state
State tied to context	Comman buffer instead of state
Sequential operations	Multithreaded programming
Limited control of GPU memory and sync	Explicit control of memory man. and sync
Extensive error checking	No error checking at runtime

3D APIs

- **GPGPU**
  - OpenCL
    - On Android it is not much loved
      - Use GPU vendor SDK provided libs ☺
    - On iOS is only accepted for system apps
      - Use old-school GPGPU (fragment shader -> FrameBuffer)
  - Compute shaders
    - GLES 3.2!!! **General solution**!!
  - DirectCompute on D3D

Cross-development

http://www.appian.com/blog/enterprise-mobility-2/are-mobile-platform-choices-limiting-enterprise-process-innovation

Cross platform

- **Unity Mobile (for gaming and VR)**
  - iOS/Android, integration with Tango
- **Unreal Engine 4 (for gaming and VR)**
  - iOS/Android
  - former Unreal Development Kit
  - free usage, payment only for shipping
- **Corona SDK**
  - iOS /Android
  - uses integrated Lua layered on top of C++/OpenGL to build graphic application
  - audio and graphics, cryptography, networking, device information and user input

Cross platform

- **Marmalade**
  - iOS/Android/Windows
  - two main layers
    - low level C API for memory management, file access, timers, networking, input methods (e.g. accelerometer, keyboard, touch screen) and sound and video output.
    - C++ API for higher level functionality for 2D (e.g. bitmap handling, fonts) 3D graphics rendering (e.g. 3D mesh rendering, boned animation), resource management system and HTTP networking.
  - Very successful but dismissing by March 2017
- **EdgeLib**
  - iOS/Android/Windows
  - high performance graphics engine in C++
  - support for 2D graphics, 3D graphics (OpenGL ES), input and sound

Cross platform

- **JMonkey Engine**
  - Android
  - written in Java and using shader technology extensively
  - uses LWJGL as its default renderer (another renderer based on JOGL is available, supporting OpenGL 4)
- **PowerVR**
  - iOS/Android/Windows
  - a cross-platform OS and API abstraction layer, a library of helper tools for maths and resource loading
  - optimized for PowerVR GPUs, with Vulkan support
- **ARM Developer Center**
  - Plenty of tools (computer vision and machine learning, OpenGL ES emulator, texture compression)

Cross-development

- **C++ use case: QtCreator**
  - Qt (~supports android, iOS, windows phone, linux, windows, mac)
  - Provides API abstraction for UI, in-app purchases, ~touch input
  - HOWTO (i.e. android):
    - Android SDK
    - Android NDK (native C++ support, toolchain, libraries, GL, CL…)
    - Point environment variables ANDROID_SDK, ANDROID_NDK to folders
    - Create new android project
    - Play!
  - Notes:
    - Go for Qt > 5.4 (touch events were tricky in previous versions)
    - Use QOpenGLWidget instead of QGLWidget
    - Enable touch events on each widget:
      - QWidget::setAttribute(Qt::WA_AcceptTouchEvents);

Mobile Graphics – Development

- **Conclusions**
  - 1) **Native + platform UI …**
    - C++ [any language] → LLVM compiler → target platform
    - Platform Framework **front-end → 1 for each platform**
    - **Performance + flexibility**
    - Call native code from platform code (JNI, Object C, …)
  - 2) **Native through framework …**
    - Qt | Marmalade …
    - C++ code uses framework API
      - **Framework** API **abstracts platform** API [N platforms]
      - **BUT** less flexible integration ?
  - 3) **Go web → HTML5/JS …**
    - JS code + WebGL
    - ~**Free portability** (chrome / firefox / IE … ?)
    - **BUT** performance is 0.5X at most with asm.js

Next Session

SCALABLE MOBILE VISUALIZATION