Tegra - Developing Killer Content for Advanced Mobile Platforms

Lars M. Bishop

Mobile Developer Technologies

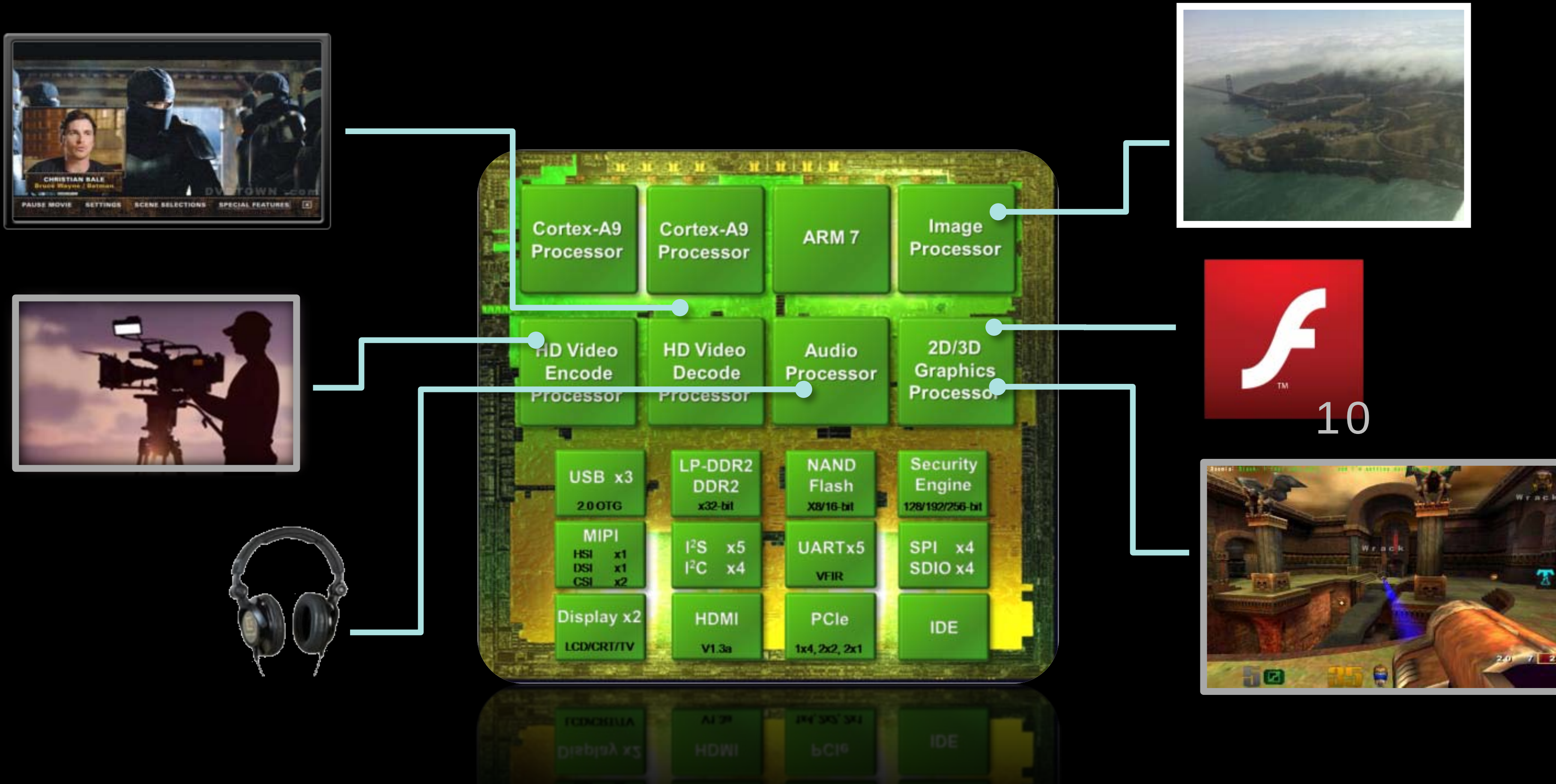
NVIDIA

Agenda

- Overview of Tegra
- User Experience and mobile multimedia apps
- Platform integration (and Android)
- Using the Khronos APIs to create compelling multimedia content
- Developer Tools and Site
- Along with a few demos!

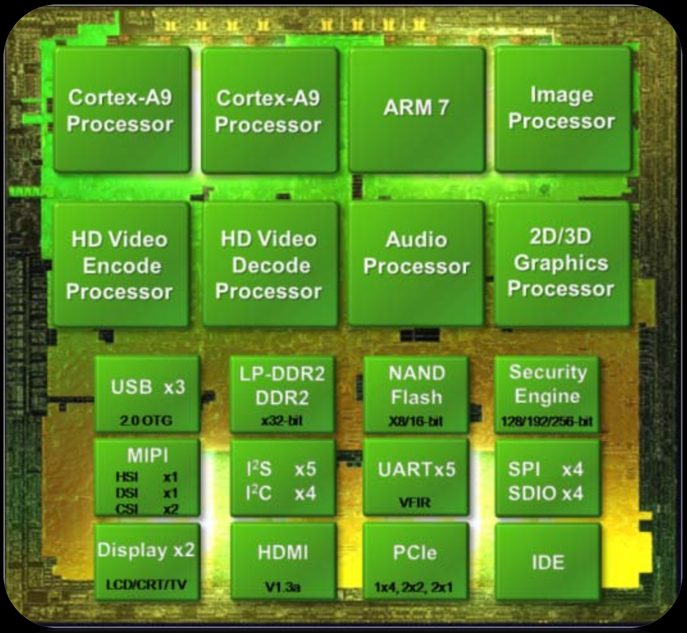
What is Tegra?

- Advanced, mobile System-on-a-Chip (SoC)
- *Soul of the Machine*: Low-power, top performance



Tegra Basic Feature Blocks

- Dual-core ARM Cortex-A9 CPU
- Shader-based, high-performance 3D GPU
- Dedicated HD video encode/decode HW
- High-resolution, dual-display support
- Built in, advanced dual-camera support, including onboard imaging pipeline (ISP)



Operating Systems

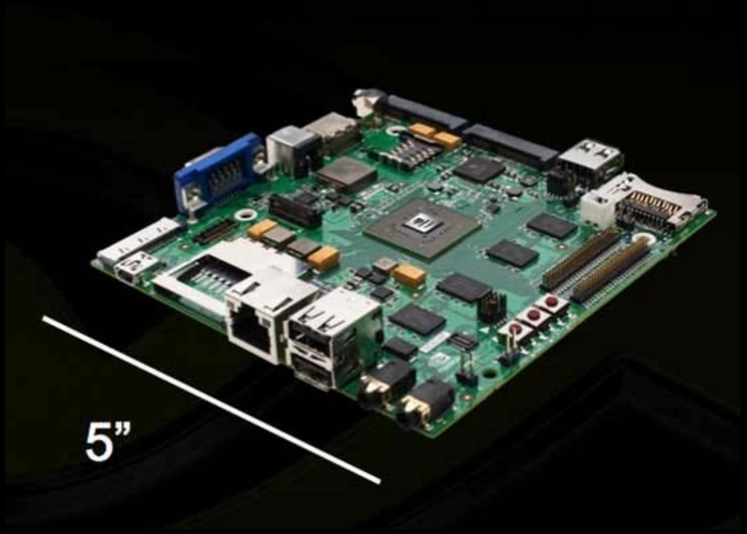
- Android
  - Available publicly on Developer site!
- Chromium
  - Smartbook-focused
- Linux
  - Soon to be available on Developer site
- Windows CE
  - Available publicly on Developer site!

Markets and Products

- Handheld
  - Microsoft Zune HD
- Embedded / Auto
  - Audi, Maserati, etc
- Tablets / Smartbooks
  - Notion Ink Adam
- ~50 designs in progress for 2010

Tegra 250 Developers' Kit

- A full Tegra system on a board!
  - Tegra 250 SoC
  - 1GB RAM
  - 512MB of flash
- VGA / HDMI display support
- USB keyboard/mouse support
- Built-in Ethernet, Wifi, Bluetooth
- Lots of expansion ports
  - PCIe
  - SD Card
  - External UART



User Experience

- User Experience on a mobile device is complex
  - Integration with and respect for the device’s core function
    - User should be confident that everything will “just work”
  - Power efficiency
    - Don’t waste the user’s battery
    - Convergence devices are “shared functionality” devices
  - Rendering quality and performance
    - User expectations are set by non-mobile devices!
    - Media crossover apps

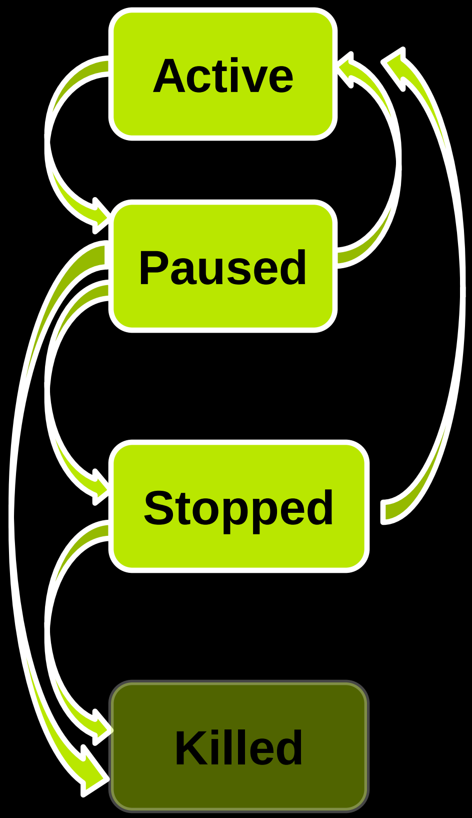
Platform Integration

- Know the platform events to be handled
  - Do not want a user trying to pause their game manually when a call comes in!
  - But also don’t want user to lose their progress!
- Be prepared to be swapped out or shut down
  - Mobile OSes can be very aggressive on managing memory
  - Know if and how your OS gives warnings before using The Hammer

Platform Integration: Android

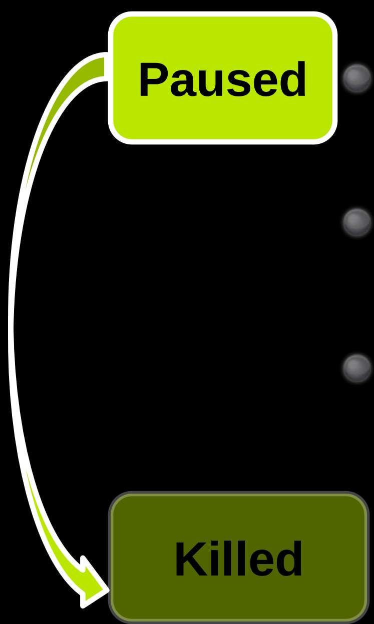
- Android applications
  - Are Java components at the top level
  - *Normally* run in their own processes/JVMs
    - App components can each run in their own process
    - And other apps can invoke an app’s components
  - Can call down into native code using the NDK
    - Can only a set of approved APIs (avoids fragmentation)
    - Runs in the process of the app
  - Consist of several building blocks, but we’ll talk briefly about the visual ones: Activities
    - These provide visual components and user interactions
    - Tend to have at least one window, often fullscreen
    - Can create more windows as needed

Android Activity States



- Active
  - On top of the stack of visible activities
  - Does not mean the user is actively interacting with it…
- Paused
  - (Partially) visible, but covered by another transparent or part-screen app
- Stopped
  - Completely invisible (likely no rendering surface)
- Shut Down/Killed
  - Process no longer running

Android and “Paused” Activities



- Note the direct Paused→Killed arrow!
- A paused Activity can be killed!
    - “*at any time* without another line of its code being executed”
- Would you see this in testing? Maybe not
    - But the spec allows it
- This is why onPause documentation recommends saving persistent state
- If you don’t, a user could lose their data
    - E.g. game progress
- Implement important system callbacks in your apps
    - `onCreate`
    - `onPause`
    - `onResume`

Static Data and Activity Lifespan

- **Even if an Activity stops, its process may be left resident if there are enough resources available**
- **When the Activity restarts, static variables may *or* may not be re-initialized**
- **Code such as the following may *not* represent your model of the first call in a given Activity's instance**

```
static int firstCallToThisFunction = 1;
if (firstCallToThisFunction)
    // Do critical operation such as
    // resetting/initializing or loading…
```

Power Management

- Use cycles wisely!
- Apps that drain the battery won’t be popular for long
- Don’t spin needlessly
  - Be event-driven
  - Throttle frame rate reasonably
- Use the efficient subsystem for the job
  - Vertex shaders instead of CPU for skinning
  - Video core instead of CPU for video decode

Power Management: Android

- Example: Keeping the screen on and bright
- Android includes multiple methods:
  - Activity timers: fine if the app is interaction-focused
  - “Wake Locks”: very aggressive, not recommended
  - Window flag: more integrated with app focus
- Use the right one; the least invasive for your needs
- Don’t keep brightness on all the time in your game
  - Clear brightness flags between levels, in menus, etc
  - Or put these modes in windows with no power hints
  - Consider the needs of the game in each interaction phase
- But don’t ignore them or skip them

Multimedia Quality

- Users have growing expectations for rendering and multimedia quality
- Tegra supports media acceleration via standard APIs including:
  - Shader-based 3D
  - Video playback and encode
  - Camera support
- This makes it easier to port high-quality content across the supported OSes
- Tegra focuses on the Khronos Media APIs

Khronos APIs

- Open standard for multimedia acceleration
- Includes:
  - Display/Buffer Management: EGL
  - 3D: OpenGL ES
  - Multimedia: OpenMAX
  - Platform Abstraction: OpenKODE Core

EGL

- Replaces all of the per-platform WGL, AGL, XGL's
  - Makes porting Khronos apps a LOT easier
- Buffer, context and configuration management
- Not just a "setup API"
  - Serves as the Khronos media API hub!
  - Also makes interesting cross-API use cases possible and standardized (EGLImage)

EGL Config Confusions

- **EGLConfigs define the pixel depth, aux buffer formats, API support, etc for surfaces and contexts**
- **Querying and selecting a config can be confusing:** `eglChooseConfig(disp, attribs, &config, 1, &count);`
- **Don't be tempted to just grab first matching config**
  - **See the EGL spec – the sorting method required by the spec ended up being confusing to some developers**
  - **E.g. requesting 16bpp RGB can return 32bpp RGB** ***FIRST*** **even if an exact 16bpp config existed**
  - **Spec requires that the deeper config be returned first!**
  - **Other surprises in there, too**
- **Request a long array of matches, and sort in the app**

OpenGL ES 2.0

- Becoming widely supported on major smartphone OSes / platforms and other mobile platforms
- Current and next-generation mobile 3D hardware is generally built for ES 2.0
- Availability of powerful vertex and pixel shaders are an important upgrade:
  - *Performance*: avoid per-vertex CPU work that was common when shoehorning modern content into ES 1.x
  - *Differentiation*: huge range of effects now possible
  - *Power*: Use the right core for the job; dedicated vertex units avoid lighting up CPU's FPU as much

OpenGL Optimizations

- Good, compelling content tends to be large
  - Memory bandwidth can be tight
- Optimize all aspects of memory bandwidth usage
  - Vertex formats/layout
    - Normals are particularly ripe for small formats
  - Texture formats, mipmapping
    - Compression is (still) king
    - Use deep textures, but only where they’re needed
    - Use 1- and 2-component textures where possible
  - Maximize use of static VBOs/IBOs
    - Use indexed primitives, sort for vertex caching
    - IBOs+VBOs to allow for maximal GPU parallelism
- Packing multiple 1-3D attribs into a 4D attrib

OpenGL Optimizations (2)

- Shaders can be “heavy state”
  - Uniforms are shader state and must be restored on shader swap
  - Avoid shader thrashing; group by shaders
- Screen densities on newer mobile devices are high
  - Lots more pixels to fill now (854x480, 1024x600, etc)
  - Do work in the (likely under-utilized) vertex unit if possible
  - Consider rough depth sort or a depth pre-pass to optimize later color pass for expensive shaders
- How much shader precision do you need?
  - Use lowp and midp where possible
  - Important for varyings and locals

3D Game Demos

OpenMAX IL

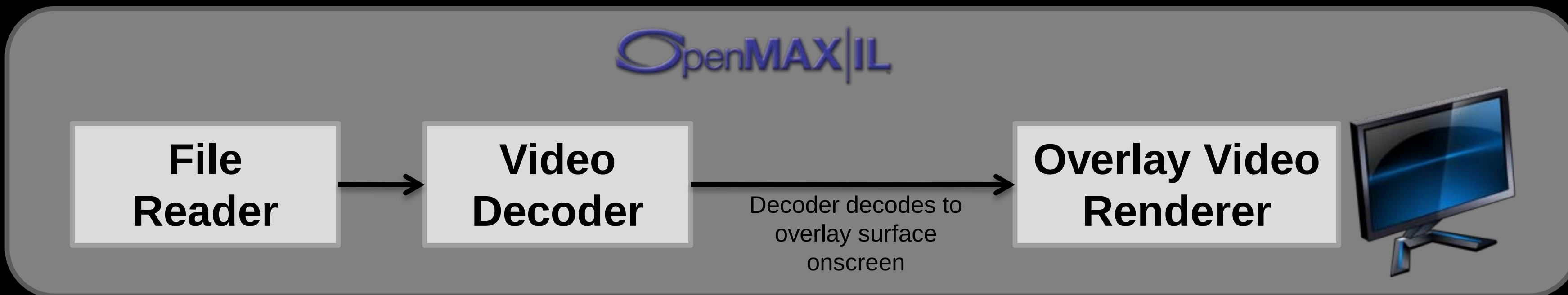
- More and more, video, camera and multimedia integration are becoming core features of next-gen mobile entertainment apps
- OpenMAX IL is a graph-based media API
  - Readers / streaming sources / camera devices
  - Decoders
  - Processors
  - Renderers
  - Encoders
  - Writers
- The result is an advanced API, capable of much more than basic “play video to screen” use cases

Multimedia Integration

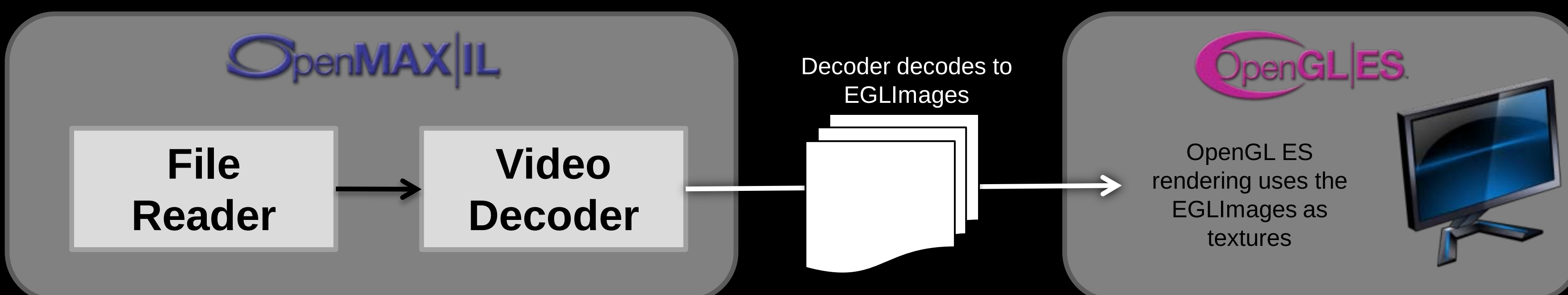
- Khronos APIs make multimedia-and-3D integration possible in a standard way
- EGL and its EGLImage extensions are key
- EGLImages allow image data created in one Khronos API to be used in another API
  - OpenMAX image buffers and OpenGL ES textures
- Tegra and its driver stack accelerates these “cross API” use cases
- Tegra Khronos Apps SDK includes interop sample code (NVIDIA Tegra developers’ website)

OpenGL ES + OpenMAX + EGLImage

- **OpenMAX IL is often used “tunneled”, e.g. connecting video decode to video rendering:**



- **But OMX IL can also decode to EGLImages, which can be used as textures in OpenGL ES 2.0**

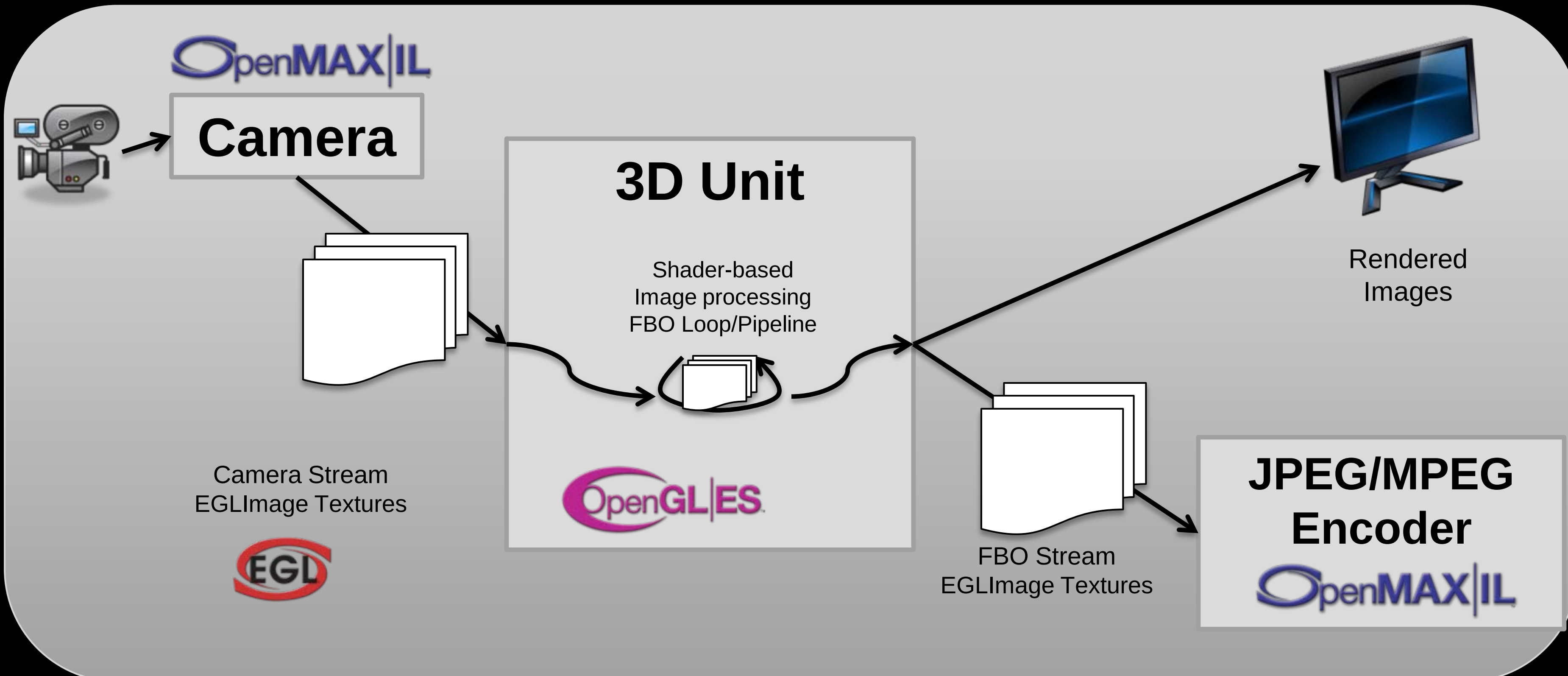


- **In addition, a camera can stream to EGLImages…**

Application: GPU Image Processing

- **Connects the camera directly to the 3D unit for processing effects**
- **“3D Camera” demo**
  - **GPU-based shader/FBO “pipeline”**
  - **FBOs to pass images stage-to-stage**
  - **Supports 1-4-channel images**
  - **Allows for 8 bit fixed-point or 16-bit floating point buffers between stages**
- **Output can be:**
  - **Drawn to the screen**
  - **Sent to the image/video encoder**

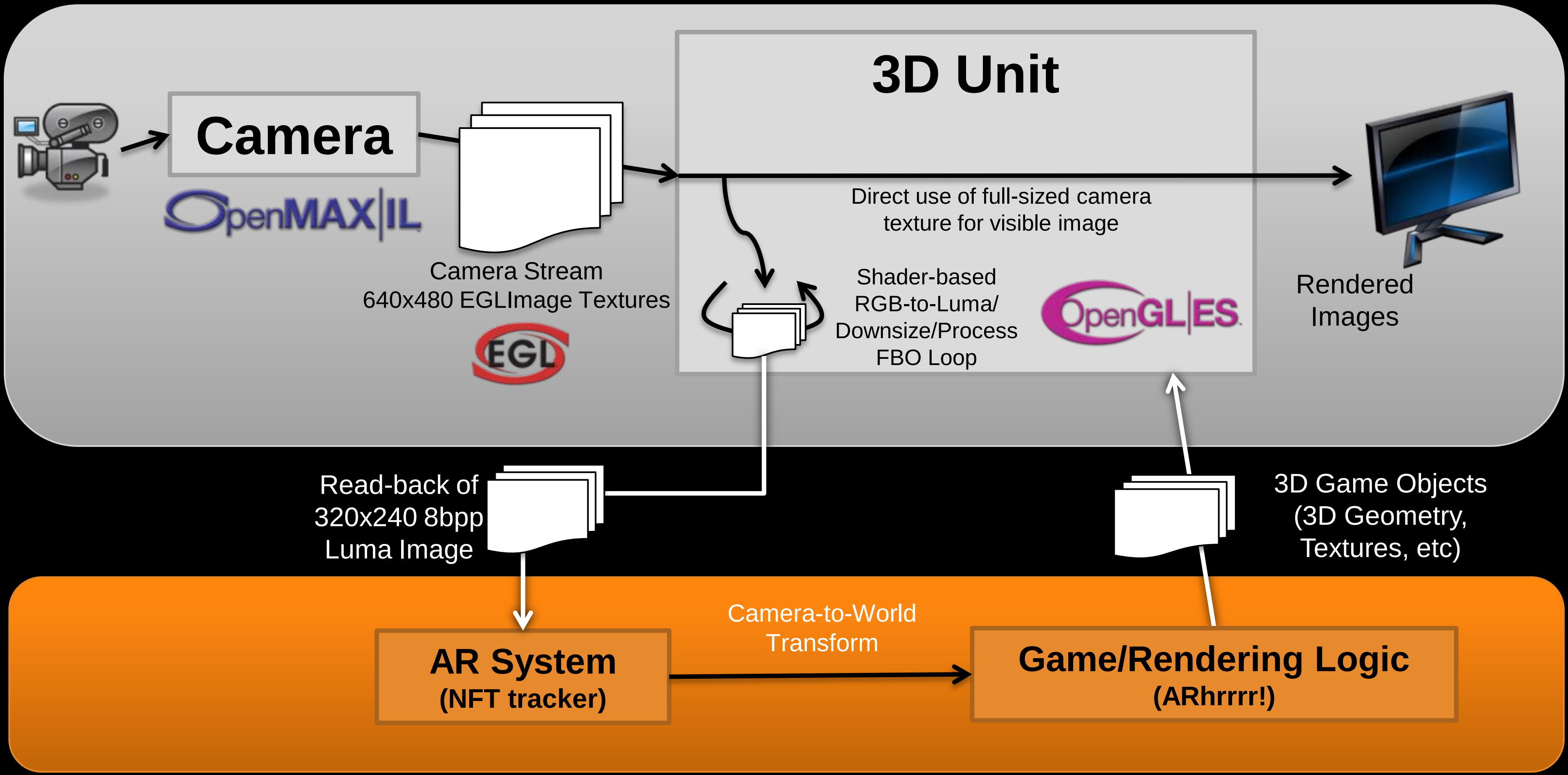
GPU Image Processing Pipeline



Application: Augmented Reality

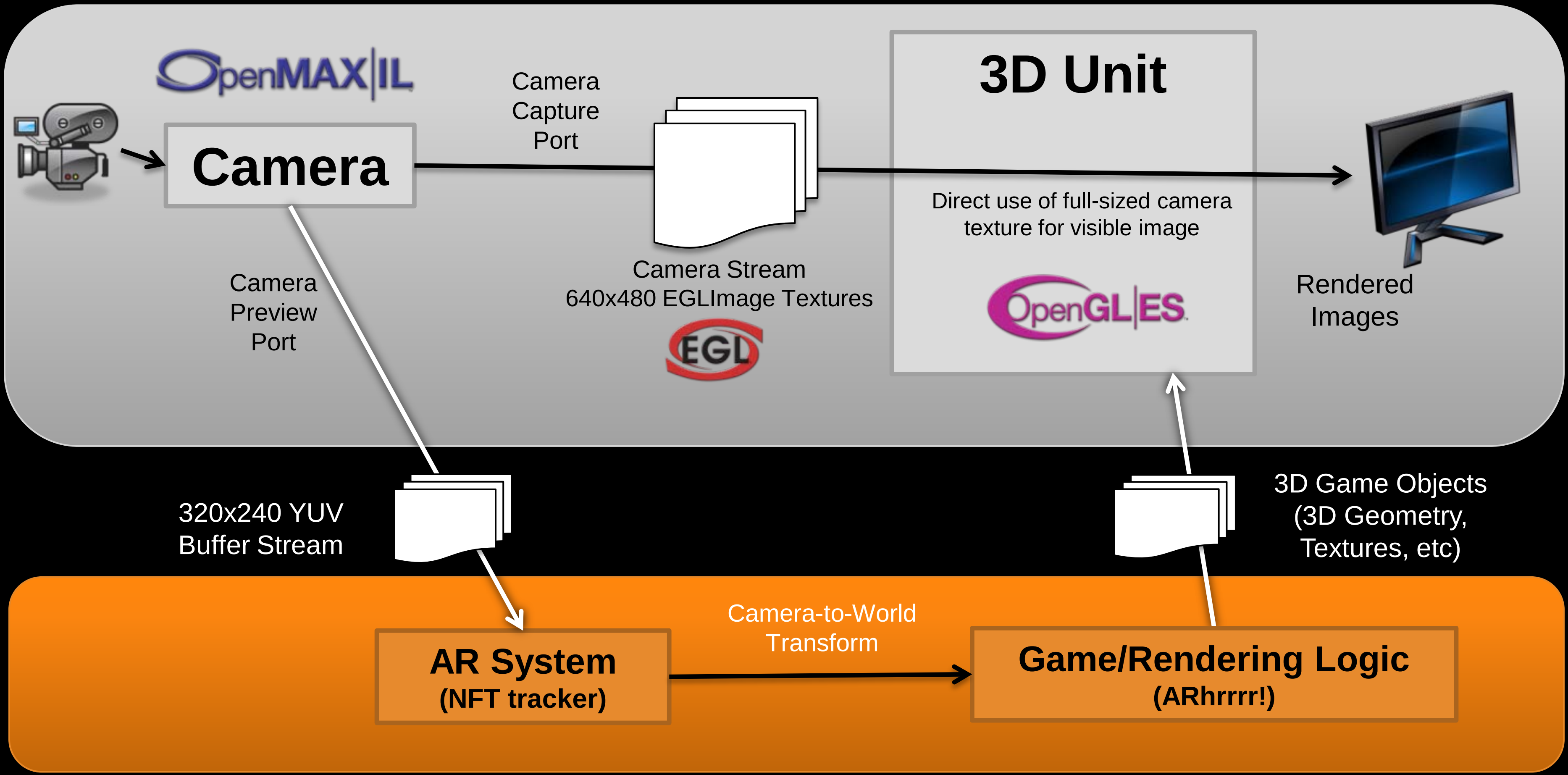
- Natural Feature Tracking
  - Camera position inferred from recognizing the game board in the real world
  - No barcodes, etc
- Camera image used
  - On CPU for tracking
  - On GPU for rendering

Augmented Reality Imaging Pipeline



- Note that this is merely one possible pipeline…

Alternate AR Imaging Pipeline



- No need to stall the GPU for a read-back
- But synchronization of the streams a challenge

Augmented Reality Video Demo

- Zombies Augmented Reality game, “ARhrrrr!”
  - Camera-based AR with CPU-based tracking
  - Natural feature tracking of a physical map
- Created by
  - Augmented Environments Lab, GA Tech (Blair MacIntyre)
  - Savannah College of Art and Design (Tony Tseng)
  - T U Graz (Daniel Wagner)
- Details at NVIDIA’s 2009 GTC site and the Augmented Environment Lab’s site

PerfHUD ES

- Mobile-centric NV PerfHUD
- Renders stats and graphs on a separate host PC
  - Minimizes overhead on mobile device
  - Allows for more screen real estate for feedback
  - Most mobile dev is done with a host PC, anyway
- Works on Android, Linux and WinCE targets
- Includes/Supports
  - Stats graphs (memory, frame time, driver time, draw calls)
  - Directed tests (2x2 textures, ignore draw/all calls, etc)
  - Frame profiling
  - Frame debugger

Handy PerfHUD ES Features (2)

- Call Trace / Frame Debugger Mode
- Full list of state calls in frame (redundancy checking)
- Frame “scrubbing”
- Partial-frame (frame-to-call) views including FBOs
  - Color buffer
  - Depth buffer

Handy PerfHUD ES Features

- Performance Dashboard Mode
- Ultra-fast top-level performance triage
  - Hit the common, top-level issues quickly
  1. Ignore all calls (are we just app-limited?)
  2. Null fragment shader (are we shader-heavy?)
  3. 2x2 textures (are we memory-bound?)
  4. Disable primitive batches by histogram
- Break-on-GL-error
  - For those rare (ahem) cases where your code isn’t checking each call

Tegra Developers' Site

http://developer.nvidia.com/tegra

- OS Support packs
  - Android
  - Windows CE
  - Linux (coming soon)
- SDK's, demos, apps
- Docs
- Development Tools
- Public support forums/community
- Access to the Tegra board store