NVIDIA Tegra: Zooming to Bang Bang Racing & Next-Gen Mobile Gaming

Lars M. Bishop, NVIDIA

Mike Clarke, Playbox Ltd.

NVIDIA®

Agenda

- Bringing high-end games to Android
- The common stages of a port
- Target APIs and Android OS versions
- Application lifecycle
- Graphics topics
- Development tools and aids
- Real-world case-study: Bang Bang Racing
  - Presented by Mike Clarke

Targeting Android

- What screen sizes and form factors do you intend to target?
  - Tablet? Phone? Both?
  - Widescreen? Portrait? Both?
- Input devices?
  - Touch? Multi-touch?
  - Accelerometer/gyros?
- What versions of Android do you intend to support?
  - Include Éclair (2.1) and Froyo (2.2) support?
  - Support only Gingerbread (2.3) and newer?
- Rendering HW requirements?
- Can require workload / TAM tradeoffs

Bringing High-end Gaming to Android

- Common sources today:
  - Ported from existing console/PC titles
  - Ported and expanded/improved from existing mobile titles
  - Co-developed on multiple mobile platforms
- Common phases:
  - “Bring-up”
  - “Tuning”
  - “Productization”
- Plan your port up-front…

Stages of the Port: Bringup

- Common work items:
  - Build system integration
  - System APIs (sockets, file I/O, input, etc)
  - Renderer creation (OpenGL ES)
  - Content re-export/packing
  - Initial debug
- Do not ignore productization planning even at this stage

Bringing up the Renderer

- Pivotal part of any port
- OpenGL ES is the 3D API for Android
  - Mobile titles likely have one
  - PC and console titles probably don’t
- OpenGL ES can be “brought up” on desktop!
  - Linux/Windows OpenGL ES emulators to the rescue
  - Can make a world of difference
- Must choose between OpenGL ES 1.x and 2.0
  - Next-gen titles require 2.0
  - All titles should consider 2.0

Stages of the Port: Tuning

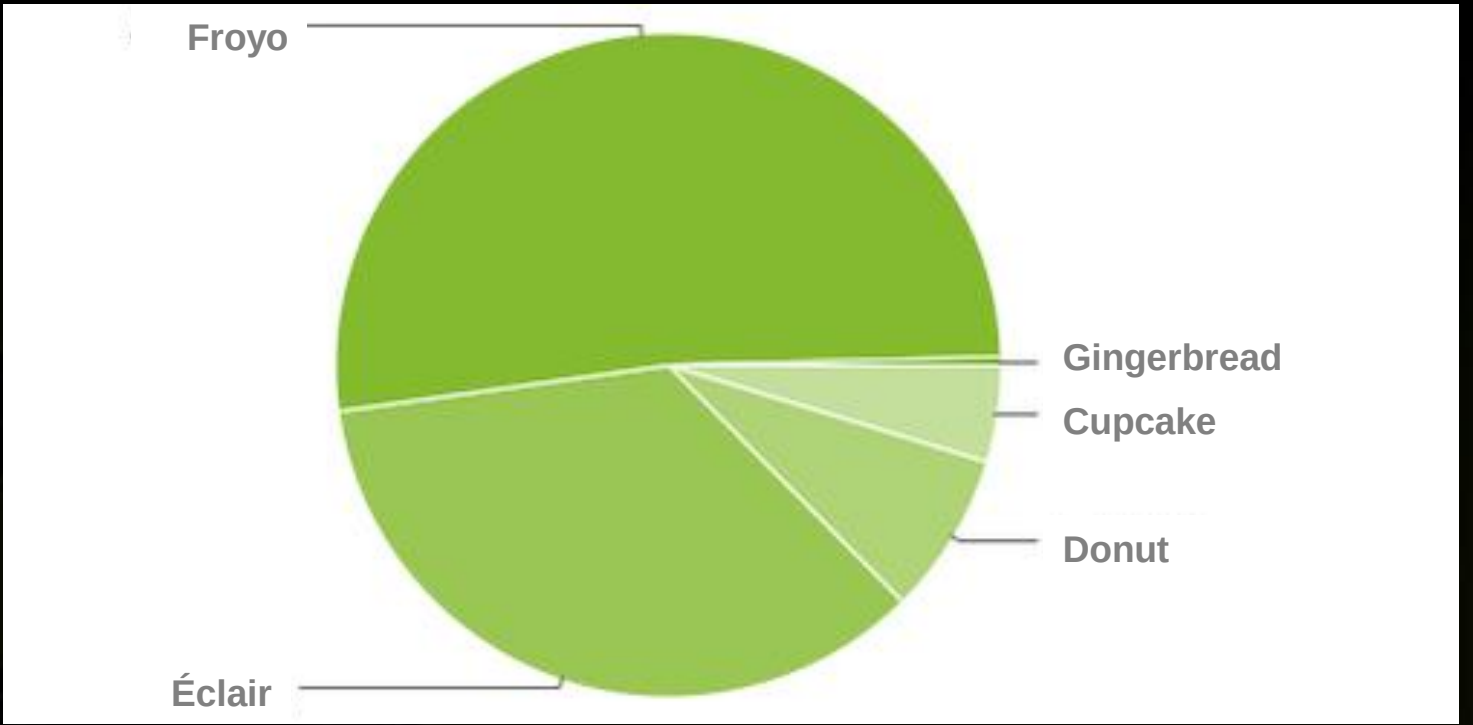
- Performance tuning
  - Determine the main bottlenecks (CPUs, GPU, memory bandwidth)
  - Circle down on them with the available tools
- Gameplay tuning
  - Sensitivity/variance of input devices and sensors
  - Adjusting for screen size and form-factor (e.g. on-screen “gamepads”)
- Important to have multiple devices no later than the tuning phase

Stages of the Port: Productization

- Key to good reviews and happy users
- More than “doesn’t crash”
- Includes handling:
  - Android lifecycle
  - 3D platform differences
  - OS versions
  - Power management

Android Market Share (2 Feb 2010)

Android Version	Active Market Share
Gingerbread (2.3)	0.8%
Froyo (2.2)	57.6%
Eclair (2.0/2.1)	31.4%
Donut (1.6)	6.3%
Cupcake (1.5)	3.9%



^ "Platform Versions". *Android Developers*. http://developer.android.com/resources/dashboard/platform-versions.html. Retrieved 2011-02-02. "based on the number of Android devices that have accessed Android Market within a 14-day period ending on the data collection date noted below"

Native Code, Java and the NDK

- High-end game content is C/C++
- Android apps are Java-focused
- The Android Native Development Kit adds support for C/C++
- Choosing your minimum Android SDK level determines:
  - The features you can use (newer means more)
  - The APIs that can be called from C/C++ (newer means more)
  - The oldest OS revision your game can support (older means larger addressable market)
- Decisions, decisions…

Android 2.1 (Éclair)

SDK API level 7+

- Available from native code (C/C++):
  - POSIX-like threads, file I/O, math, sockets
  - OpenGL ES 2.0
- Not available from native code (C/C++):
  - EGL
  - User input
  - Audio
  - Video
  - Android UI rendering
- In addition, the entire installable app must be installed to (limited) internal storage
  - If you want to put large data on external storage, you MUST sideload

Android 2.2 (Froyo)

SDK API level 8+

- Now available from native code C/C++:
  - The ability to lock a surface and CPU-render to it
- In addition, applications can now request to have most of their data (other than code) be installed to external storage
  - This still may not remove the need to side-load data…
  - The Android Market places size limits on APKs (currently about 50MB)

Android 2.3 (Gingerbread)

SDK API level 9+

- Now available from native code C/C++:
  - OpenSL ES Audio
  - Pure-native application lifecycle
  - EGL!
  - Window management
  - User input (touch, buttons)
  - Sensors (accelerometer, gyro, compass)
  - Installed asset/resource loading
- Basically, most games can do all of their work common work (other than play videos) with no Java code
- But the result will not run on Éclair or Froyo

Android 3.0 (Honeycomb)

SDK API level 11+

- No new NDK for Honeycomb yet
- Some app lifecycle changes
- Java-level additions
  - Code to assist async data loading
  - More Video/Camera-to-3D integration
  - Touch events can span multiple activities

NativeActivity

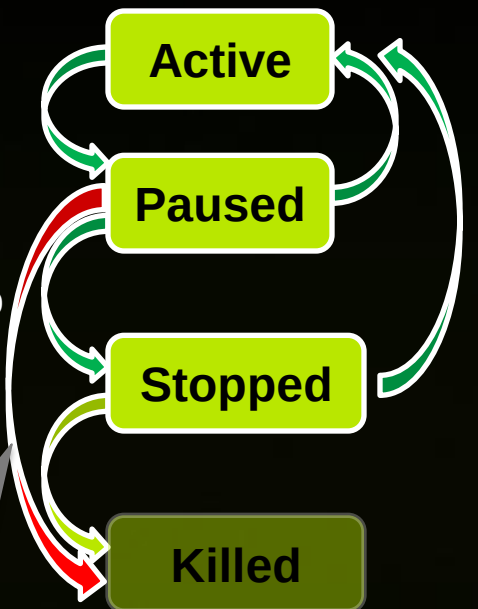
- API level 9 (Gingerbread) adds NativeActivity
- Supports
  - The entire app lifecycle
  - User input
- All in native C/C++ code
- This is key to minimizing or avoiding Java app code
- Also, note the NDK's native_app_glue

Dalvik (Java)-only Interfaces

- Still Java-only:
  - Video playback
  - Camera
  - Android UI rendering
  - Lots of system-integration APIs
- Research your “native only” decision carefully

The Circle of Life

- **Active**
  - On top of the stack of visible activities
  - Does not mean the user is actively interacting with it…
- **Paused**
  - Invisible; user has moved to home screen, another app in front, etc
  - (Rare) partially visible, but covered by another transparent or part-screen app
- **Stopped**
  - App being closed
  - Completely invisible (likely no rendering surface)
  - But may transition to Active again without the process dying
- **Shut Down/Killed**
  - Process no longer running



Pre-Honeycomb only

High-level Concepts

- Consider supporting:
- Fast-path loading for OpenGL ES resources
  - Textures
  - Vertex/Index buffers
  - Makes it easy/faster to handle EGL_CONTEXT_LOST
- Fine-scale “game save”
  - And do so incrementally
  - Makes it easy to support onPause/onResume
- Only explicitly-initialized static data in native code
  - Makes it easy to handle being kept resident in memory after application exit

Threading

- Multi-core SoC's like NVIDIA's Tegra are the accepted norm
- Threading is key to maximizing application performance
- Thread your:
  - Physics
  - Particles
  - Game logic
  - Rendering
  - Networking
- Prepare for the future as well
  - Don't assume just 2 CPU cores…

Graphics Topics

- Features, formats, flexibility
- Textures
- Geometry

Backbreaker THD:
Natural Motion Games

OpenGL ES Features: Be Flexible

- EGL is the configuration, buffer and rendering context API that sets up OpenGL ES
- Write your own EGL configuration sorting and filtering code
- Be ready to fall back if your preferred settings are not available
- Minimize the number of “absolute requirements”
- Log all configurations to debugging log for remote failure triage
- Different platforms have different support for
  - Texture compression
  - Anti-aliasing
  - Color and depth buffer formats

Texture Compression

- Texture compression is pivotal
- ETC1 is almost universal
  - But has no alpha channel
- Applications must use compressed textures for RGBA, too
- So they must handle vendor-specific or non-universal formats:
  - DXT3/5 (S3TC)
  - PVRTC
  - ATITC
- Generally, this means side-loading data per major platform
  - But the new <supports-gl-texture> tag allows the market to filter based on available texture compression formats

Geometry

- "Compress" vertex data wherever possible
  - Indexed primitives
  - Efficiently organized vertex-attribute streams
  - Use half-float, short int and signed byte attributes
- Smaller vertices lead to lower memory traffic
- Minimize independent attribute count
  - Pack attributes into 4-component attribute streams as tightly as possible

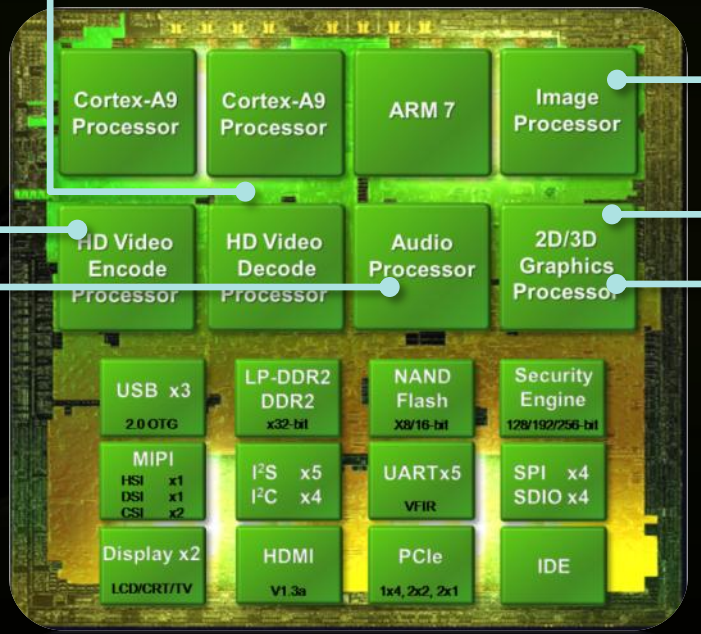
Vendetta Online, Guild Software

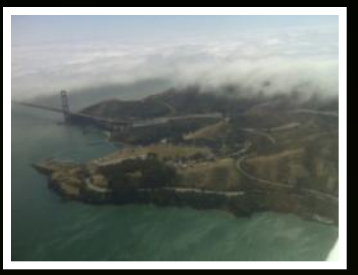
VBOs, Dynamic Geometry

- VBOs are GPU-friendly
- They
  - Minimize driver intervention (i.e. CPU work) in rendering
  - Maximize API responsiveness
  - Give the driver the maximum info for optimization
- Render static *and* dynamic geometry from VBOs
- Mark static VBOs as such
- Use round-robin sets of VBOs for dynamic geometry
  - Do not re-lock the same buffer you just used in a render call
- Use glBufferSubData to replace dynamic geometry

Tegra 2

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





Powered by Tegra 2

Development Kits

- NVIDIA supports a range of development devices
  - These are in addition to the current and soon-to-be-available consumer phones and tablets
- Tegra 250 devkit
  - Small, non-enclosed board
  - External HDMI/VGA and input

- Ventana devkit
  - Rough tablet form-factor
  - multitouch HD LCD, sensors
  - 3 cameras, wifi, battery power

- Next-gen devkit
  - Form-factor-accurate tablet with features similar to Ventana

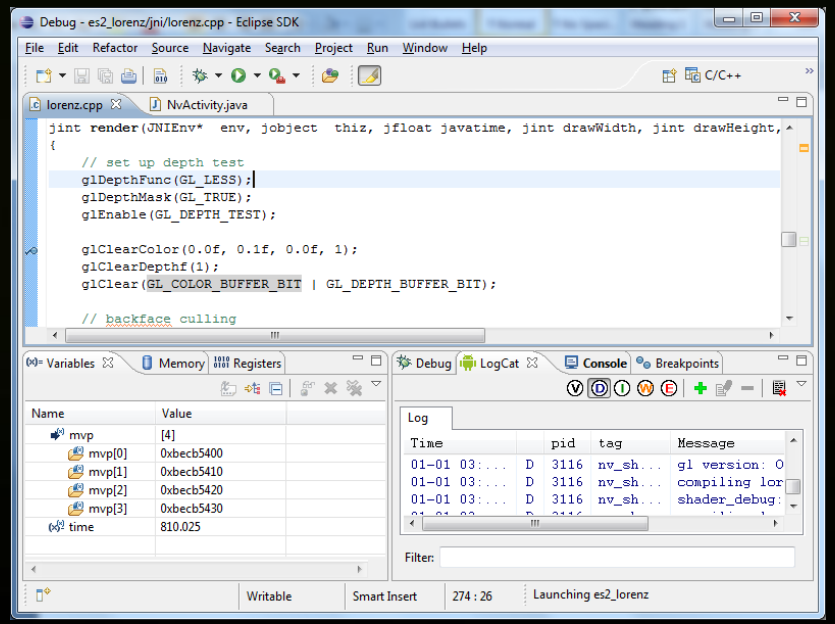
Desktop OpenGL “ES2 Profile”

- The Android emulator does not support GLES2 today
  - But even if it did, it assumes a partially-working Android app port!
- But many Android-bound apps run on Windows/Linux already
- `EXT_create_context_es2_profile` desktop GL extension to the rescue!
  - Port the renderer to ES2 on desktop
  - Using the tools you know
  - Parallelizes the port process

- Bringup on Android itself can include a known-good renderer

NVIDIA Debug Manager

- Is an Eclipse plug-in that simplifies debugging native C/C++ Android applications on Tegra
- Seamless Java and Native code debugging
- Supports the latest version of the NDK as soon as possible after each NDK update
- Supported Operating Systems include
  - Windows 7
  - Mac OS X
  - Linux



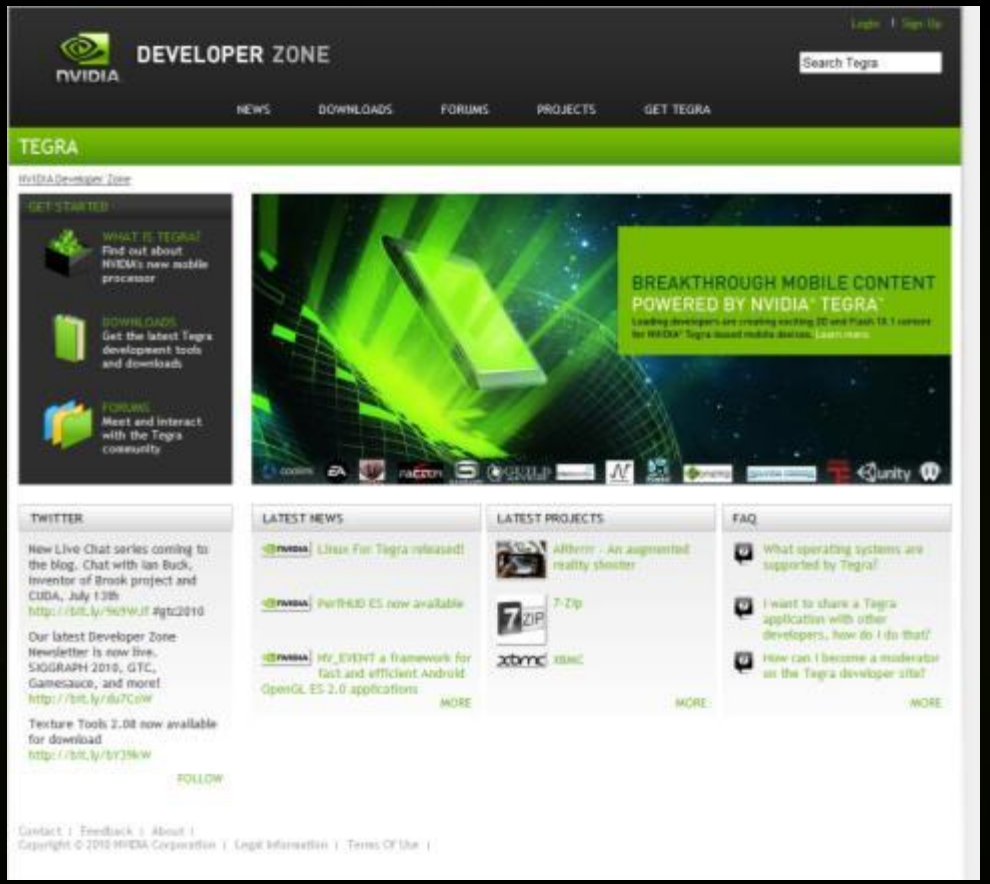
NVIDIA PerfHUD ES

- New and improved PerfHUD ES GPU profiling and debugging tool for Tegra-based development kits
- Client GUI supports
  - Windows
  - Mac OS X
  - Linux
- Improved GL State tree viewer
- Enhanced Draw Call state viewer with delta comparisons

NVIDIA Tegra Developer Zone

http://developer.nvidia.com/tegra

- OS Support packs
- SDK's, demos, apps
- Docs
- Development Tools
- Public support forums/community
- Access to the Tegra board store



TegraZone

- TegraZone makes it easy for consumers to find the best apps for their Tegra-powered devices
- In-depth game coverage
  - Game reviews
  - HD trailers
  - Gameplay videos
- Connected to social networks

Bang Bang Racing

Mike Clarke, Playbox Ltd.

Bang Bang Racing

playbox™ NVIDIA

Top-down racing

Reminiscent of retro racing games

Bang Bang Racing

playbox™ NVIDIA

1920 x 1080p @ 60fps, limited technical effects

Ripe for porting to lesser platforms

About Playbox

- Small-company mentality
- Relatively small-sized projects
- Projects could:
  - Come from anywhere
  - Be on any platform
- PC, PS2, Wii, X360, PS3, DS

Playbox Cyan Engine

- Everything abstracted
- Always chose OpenGl variants where possible
- Shaders based on Cg
- Binary data built per-platform via reflection/meta-data
- Scons build system

Porting To Android

playbox™ NVIDIA

Build System

- Cygwin is always a pain
- We use Linux environment in a VM
  - Easy to get everyone up and running
  - No licensing issues
- Playbox engine runs on PS3 and Linux
  - gcc already supported in our scons scripts
- NDK didn't have STL support until r5
  - r5 just provides STLport
  - We used customised NDK from crystax.net

BangBangRacing

Physics

- Bang Bang Racing uses PhysX
- NVIDIA provided a Tegra port
- It just worked!

Mr. Dalvik, Why Do You Hate Me So?

- C++ code trapped behind the JNI
- Fundamentals need to bind with Java
- A nightmare to debug
  - Have to use DDD alongside Eclipse
  - Java kills the C++ callstack
  - Getting it back is a pain
- NVIDIA's debug manager helps

Fundamentals

playbox™ NVIDIA

Graphics

- Relatively easy once the OpenGl context was created
- No glu functions (gluLookat etc.)

File I/O

- Just worked via stdio/fstream
- But can't be used
- Files should be read through Android package system

Threading

- Linux kernel uses pthreads
- Seems to work

Fundamentals

playbox™ NVIDIA

Input

- Must pass events through JNI
- Touch resolution matches tablet resolution
- See NVIDIA samples

Audio

- Not so bad once we worked it out
- Used our software mixer (not super-efficient, but works)
- We still have synchronisation issues

Code Samples are best

- Read NVIDIA's samples
- Somebody has probably solved your issues before (SDL/ScummVM for Android)
- Don't forget the Android issues
  - Suspend/resume
  - File loading
  - Android marketplace
  - Package size limits
  - Accelerometers spam the event queue

Renderer Issues

ES1

- Deal with the geometry/textures before the shaders
- Fixed function
- Easy to port but limited

ES2

- No fixed function
- Not far off what we already had for OpenGl
- Showed up problems in our pipeline/renderer
- We still had some glMatrix calls that had to go

Renderer Issues

- Chose GLSL instead of binary Cg shaders
- Expected to have to rewrite them anyway
- Much less forgiving than Cg
- Made a GLSL renderer for Win32 and Linux
- Used gDebugger to highlight problems
- Problems on different devices
  - Fragment precision
  - Screen Resolution
  - Tegra has various extensions (e.g. S3TC support)
  - NVPerfHud requires support of NVIDIA's timer extension for full features

Tegra Optimisation

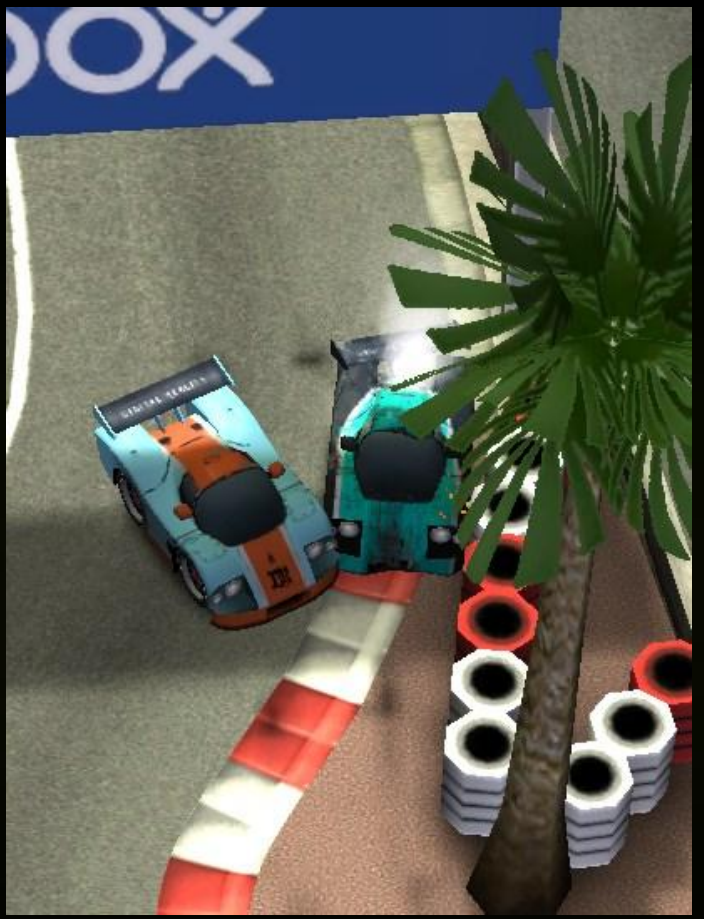
playbox™ NVIDIA

Lighting

- All per-pixel on PS3
- Put all lighting into Vertex Shaders
- Optimized redundancy
- Fragments now do very little

Shadows

- No change necessary
- Pre-rendered texture
- UV calculation
- Perfect for Tegra2



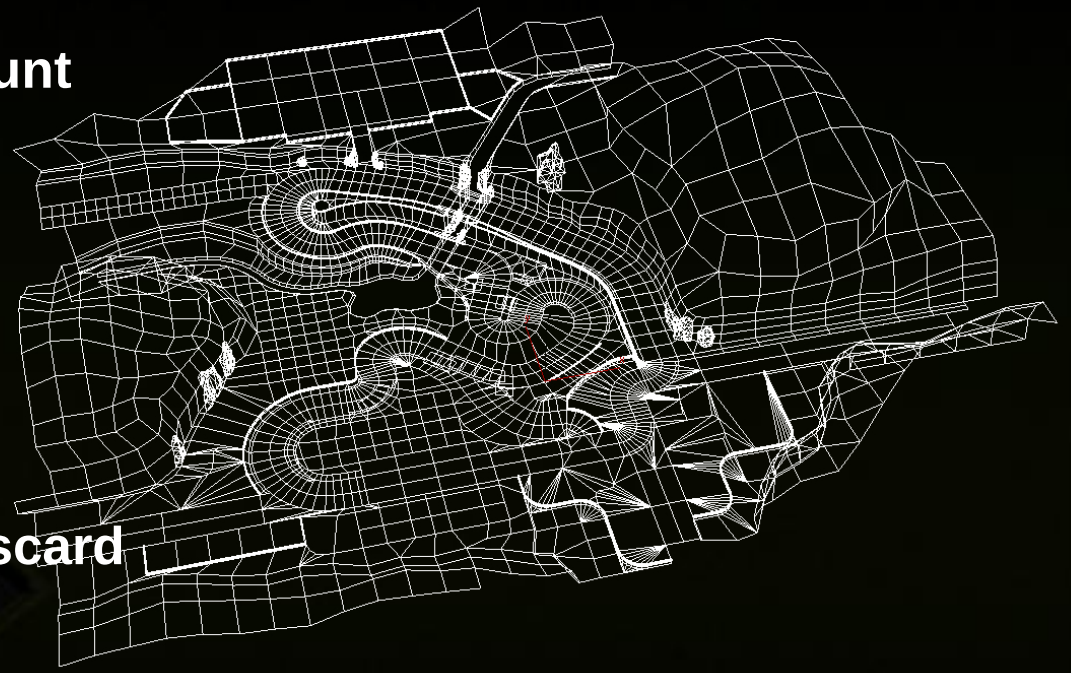
Tegra Optimisation

Geometry

- Setup for PS3
- Large pieces to reduce object count
- Chopped up for Tegra

Textures

- Alphas are very expensive
- Alpha-test is almost unusable
- Changed trees to geometry
- We use zero alphas instead of discard
- Removed HUD alphas



Touch Control

- Didn't want multi-touch if possible
  - Complex controls need to be explained to the player
- Bang Bang Racing has a directional mode
  - Mapped screen coordinates to joystick position
  - Point and go
- Touch resolution matches tablet resolution, not requested screen resolution
- Use Normalised values

Summary

playbox™

- Target 2.3+ if you can
- Deal with Suspend/Resume as early as possible
- Use PhysX
- Do as much as possible in vertex shaders
- Keep the fragment shaders very simple
- Keep alphas to the absolute minimum
- Don't use alpha test if possible
- Run on different tablets

NVIDIA @ GDC 2011

CAN'T GET ENOUGH? MORE WAYS TO LEARN:

NVIDIA GAME TECHNOLOGY THEATER

Fri, March 4th @ NVIDIA Booth

Open to all attendees. Featuring talks and demos from leading developers at game studios and more, covering a wide range of topics on the latest in GPU game technology.

MORE DEVELOPER TOOLS & RESOURCES

Available online 24/7 @ developer.nvidia.com

NVIDIA Booth
South Hall #1802
Details on schedule and to download copies of presentations visit
www.nvidia.com/gdc2011