



Whiz-Bang Graphics and Media Performance for Java Platform, Micro Edition (JavaME)

Pavel Petrosenko, Sun Microsystems, Inc.

Ashmi Bhanushali, NVIDIA Corporation

Jerry Evans, Sun Microsystems, Inc.

Nandini Ramani, Sun Microsystems, Inc.

Session TS-5585

Goal of this Session

Learn what's happening in the Java™ Platform, Micro Edition (Java Platform ME) to enable high performance graphics and how you can leverage hardware acceleration capabilities to create compelling graphical applications

Agenda

CLDC Platform and Graphics Java Specification Requests (JSRs)

SW versus HW Performance

Khronos Media Acceleration APIs

JSR 226—Scalable Vector Graphics

JSR 239—Java Binding for OpenGL[®] ES API

Performance Optimizations

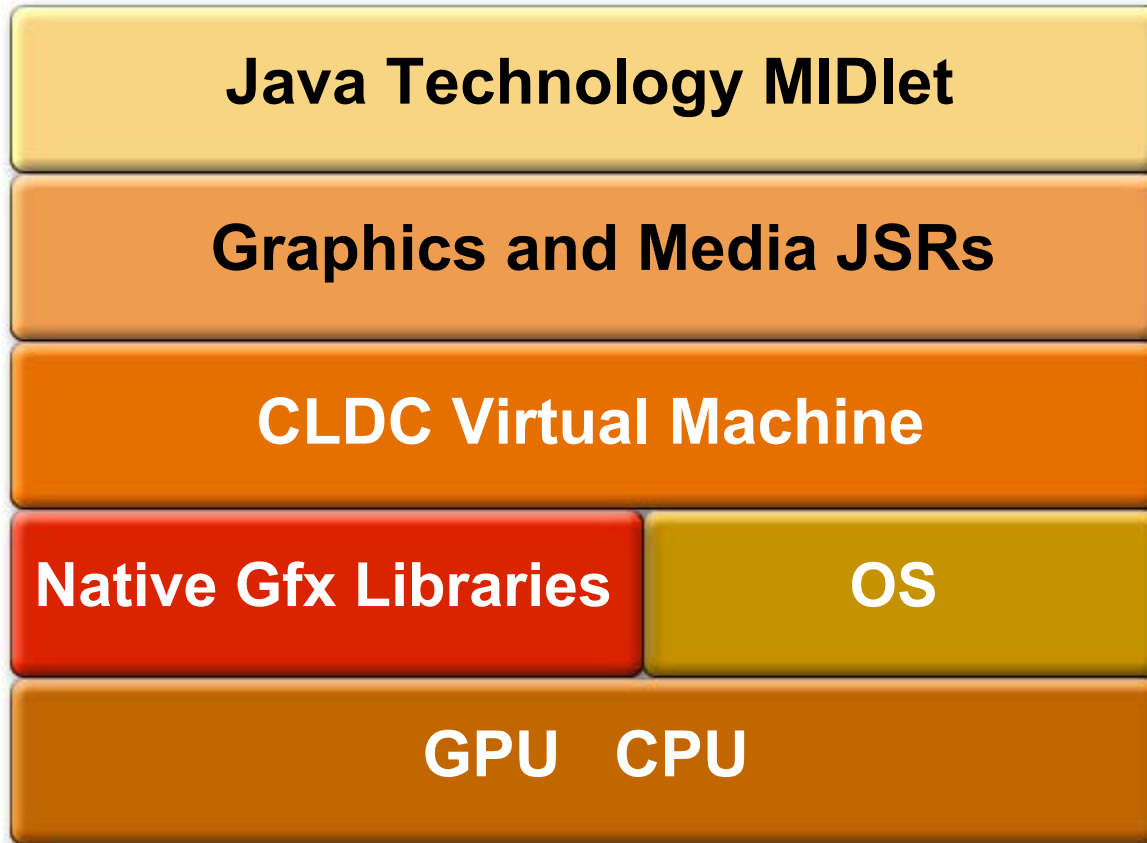
VM, Library, CPU, System Bus, and GPU

JSR 239 Advanced Effects

Q&A

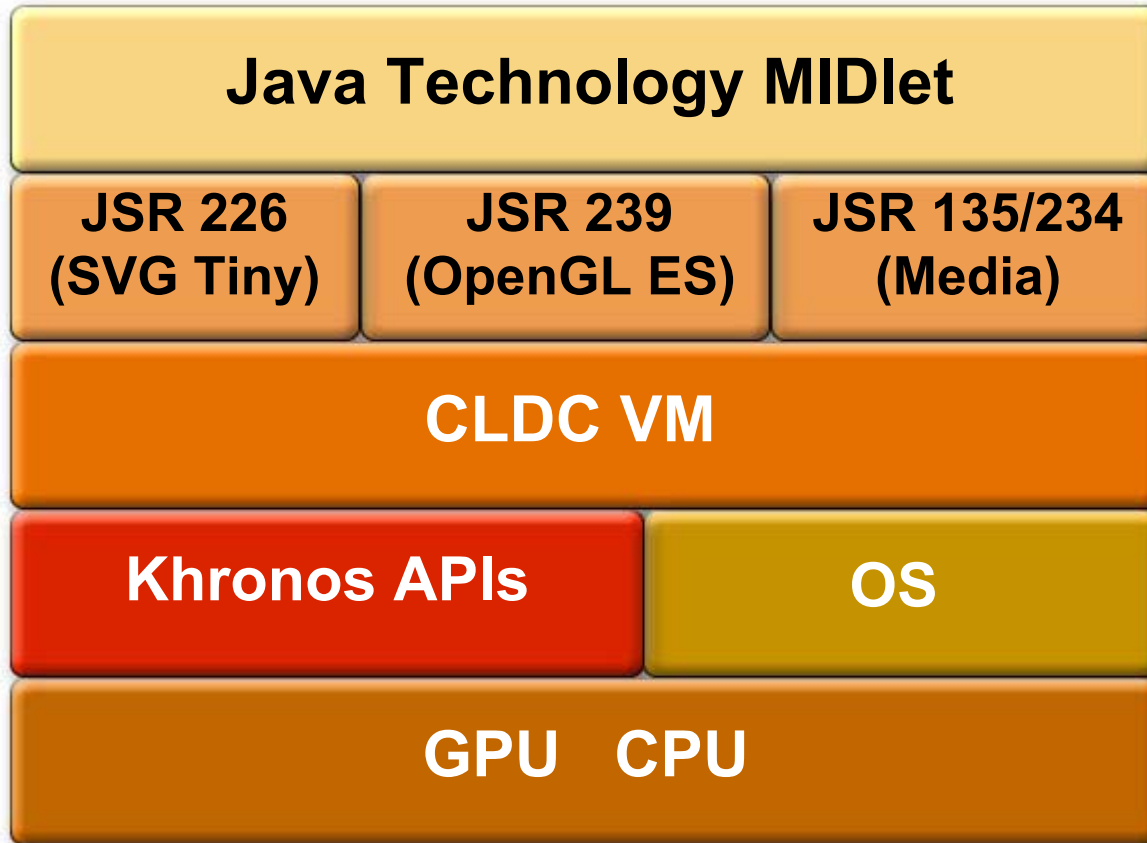


Platform Overview



Platform Implementation

Focus for this talk



Native Graphics Performance

Software versus hardware

- Until recently, most mobile 2D/3D APIs were implemented in software
- 2D/3D hardware, when available, was typically accessed through a proprietary API
- Sun is working to enable Java technology APIs to leverage standard native graphics APIs when available
 - Provide good application performance transparently
 - Allow for hardware-specific tuning for ultimate performance

Khronos Media Acceleration APIs

Applications (Java Application or Native)



Media Silicon—CPUs, DSP, Hardware Accelerators

Khronos Media Acceleration APIs (Continued)

- Khronos APIs are supported by a large number of hardware vendors for mobile devices
 - Royalty-free, cross-platform
- OpenVG (2D)
 - Low-level API for 2D vector graphics
- OpenGL ES (3D)
 - Full function 3D graphics on embedded systems
- OpenMAX (Media)
 - Comprehensive streaming media acceleration

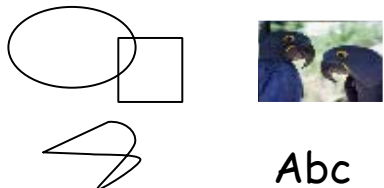


Graphics API for Java Platform, Micro Edition (Java ME)


- Java API to draw, manipulate and animate SVG Tiny 1.1 content
- Part of Mobile Service Architecture (JSR-248)—wide deployment
- Rich UI through SVG
- Complex behavior through Java platform
- DOM API connects SVG and Java platform
- Richer UI than MIDP alone

JSR 226 Features

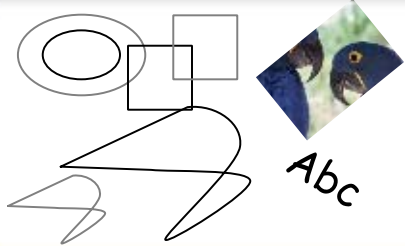
Graphical Objects



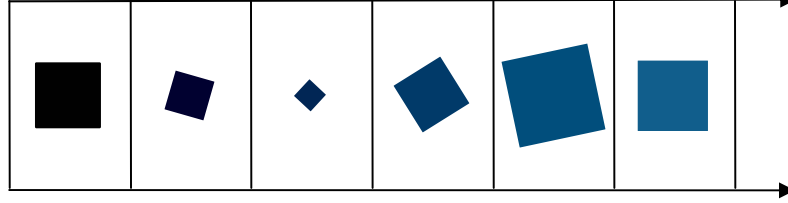
Rendering



Transform



Animation



JSR 226 and OpenVG

JSR 226



Vector Graphics Hardware



Demo

JSR 226 Demo



JSR 239—Java Binding for OpenGL ES API

- OpenGL ES is “embedded subset” of OpenGL
- JSR 239 defines Java binding to OpenGL ES API 1.0 and 1.1
- Immediate mode API
 - Highly flexible control of rendering
- Low level hardware oriented 3D API
 - 3D viewing pipeline, lighting and shading, texture mapping, cubemaps, fog...
- Access to latest and greatest hardware features

JSR 239 and OpenGL ES

JSR 239



3D Graphics Hardware



JSR 239 Game Demo





Demo

JSR 239-Based Game



VM Development Goals for Graphics Performance

- CLDC HI VM
- Optimize for interactive user experience
- Maximize performance of real-world applications
- Minimize and manage pauses
- Faster startup
- Make use of coprocessors and accelerators if available

High Performance in Limited Memory

- Optimizing compilers
 - Dynamic adaptive compiler
 - Ahead-of-time compiler
 - Optimize fast path for speed, slow path for size
- Manually optimized assembly interpreter and runtime
- Generational mark and compact garbage collector
 - Memory is scarce, so heap occupancy is high
 - Sliding young generation
 - Separate areas for large objects and compiled code

Most Effective Compiler Optimizations

- Constant folding
- Common-subexpression elimination
- Null-pointer check elimination
- Method in-lining
- Speculative devirtualization
- Register allocation
- Optimized arraycopy

Pause Management

- Incremental schedulable compilation
 - Compiler runs as a co-routine
 - Can be suspended after every bytecode
 - Can suspend itself after a given time interval
- Improvements in Garbage Collector
 - Young generation size is managed to limit a minor collection pause
 - Only user program allocations can cause collections
 - Compiler allocates in a separately managed area
- Prevention of pause clustering

Coprocessors and Hardware Accelerators

- Support of video accelerators is essential for gaming experience
 - Compiler needs to be aware of JSR implementation
- Floating-point coprocessors
 - ARM VFP, 3x3 matrix multiplication
 - Float: 36 times faster than software FP
 - Double: 54 times faster
- ARM Jazelle makes cold startup faster
- Instruction set extensions (XScale, Thumb2EE)

Future VM and Compiler Optimization Plans

- Improve general-purpose compiler optimizations
 - Array index bounds check elimination
 - Better common-subexpression elimination for loops
- Quick native functions and compiler intrinsics
 - Matrix and vector operations in a few ARM VFP instructions
- More support for JSR implementation
- Partial compilation of large methods
- Pause logging and analysis tools

Creating Compelling JSR 239 Applications

- 3D application performance tips
- Advanced rendering effects
- Demos

Handheld Device Systems

- Handheld System
 - CPU
 - GPU
 - System Bus
- Performance problems can occur at any of these levels
- Feed each part carefully to avoid performance problems

CPU Level 3D Optimizations

- Simplify CPU work (collision detecting, Skinning...)
- Avoid per triangle culling
- Batch geometry
- Avoid redundant state changes
- Avoid CPU assisted driver paths

CPU Level 3D Optimizations (Cont.)

- CPU cache is small
 - Use coherent algorithms
- Use fixed point if hardware does not support floating point
- OpenGL ES 1.0 lighting
 - Directional lights are fastest
 - Point lights are more expensive
 - Spot lights are the most expensive
 - Cost of additional light scales linearly

System Bus Level 3D Optimizations

- Remodel objects for smaller screen size
- Use VBO's (Vertex Buffer Objects) for static data
- Use compressed texture formats
- Avoid bus transfer by not overflowing GPU video memory
- Batch by texture

What's in Nvidia's GoForce 5500 GPU?

Overview

- HW geometry pipeline
- High performance texturing support
- Programmable Fragment Shader support
- Pixel Buffer support
- Vertex Buffer Objects support
- QCIF, QVGA, VGA and XGA screen size support
- Support for per-pixel fogging, alpha blending, alpha testing, bilinear filtering, perspective correction

High Performance Texturing

- Multitexturing (four simultaneously active textures)
- Texture filtering
- Texture mipmapping (using Trilinear Interpolation)
- Compressed texture support (DXT1, DXT3, DXT5)
- High quality texture support
 - 16/32 bpp
 - Supports texture size up to 1024X1024 texels
- Render to texture support (PBuffers)

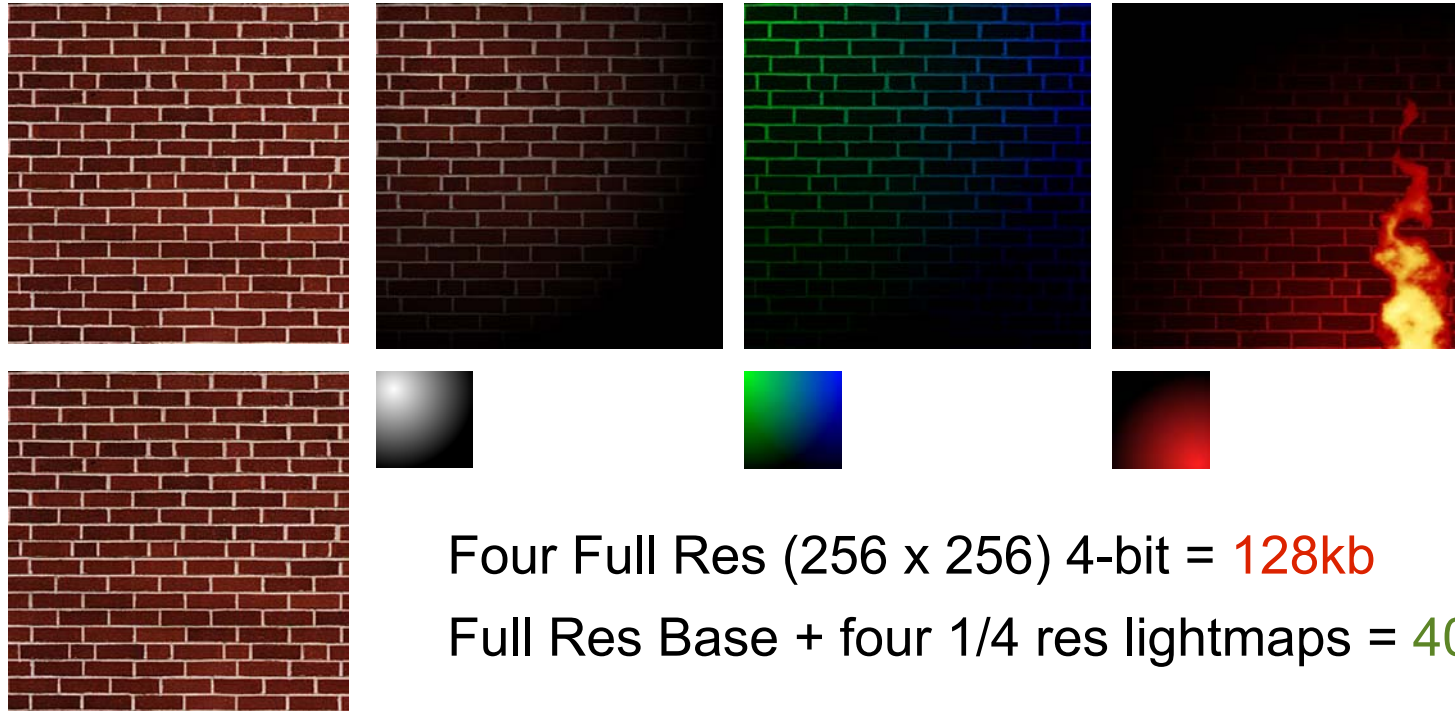
Multitexturing for Lighting

Fake Phong Highlights Using Multi-Texture



Stuntcar Extreme
(Images Courtesy of Fathammer)

Leveraging Multi-Texture



- Store diffuse maps in lower resolution and use multi-texture to save memory

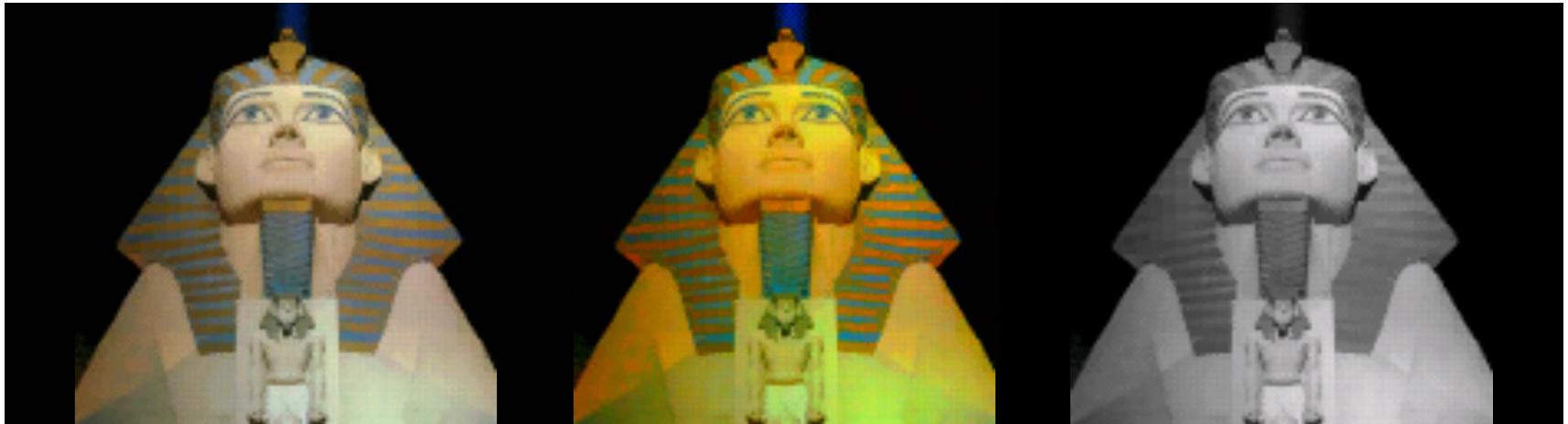
VBOs

- Support for Vertex Buffer Objects
 - Store vertex data in video memory
 - Greatly reduces system bus bandwidth requirements
 - Available via extension to OpenGL ES 1.0
- Support for Both Fixed-Point and Floating-Point Vertex Formats
 - Good for embedded CPUs that don't provide hardware support for floating-point math
- If the program has large amount of static data performance can be improved by 50–100%

Rendering to Texture and Pixel Buffers

- PBuffers are the method of rendering to texture
- For extremely powerful effects
 - Live motion blur
 - High-speed texture blurring and downsizing
 - Light blooming
 - Environment mapped bump mapping
 - Fluid flow or “plasma” effects
- All of these are possible on the GPU with very few polygons and passes

Example Effects: Image Processing



Original Image

Increasing Saturation

Decreasing Saturation

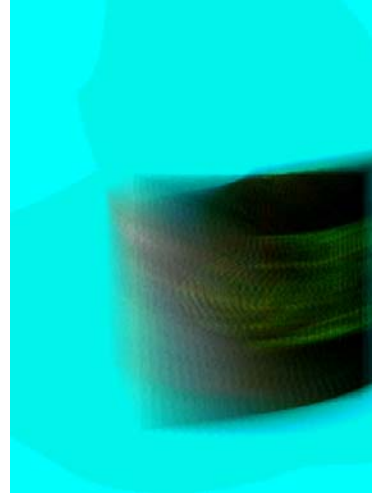


Sharpening

Blurring

Edge Detection

Example Effects

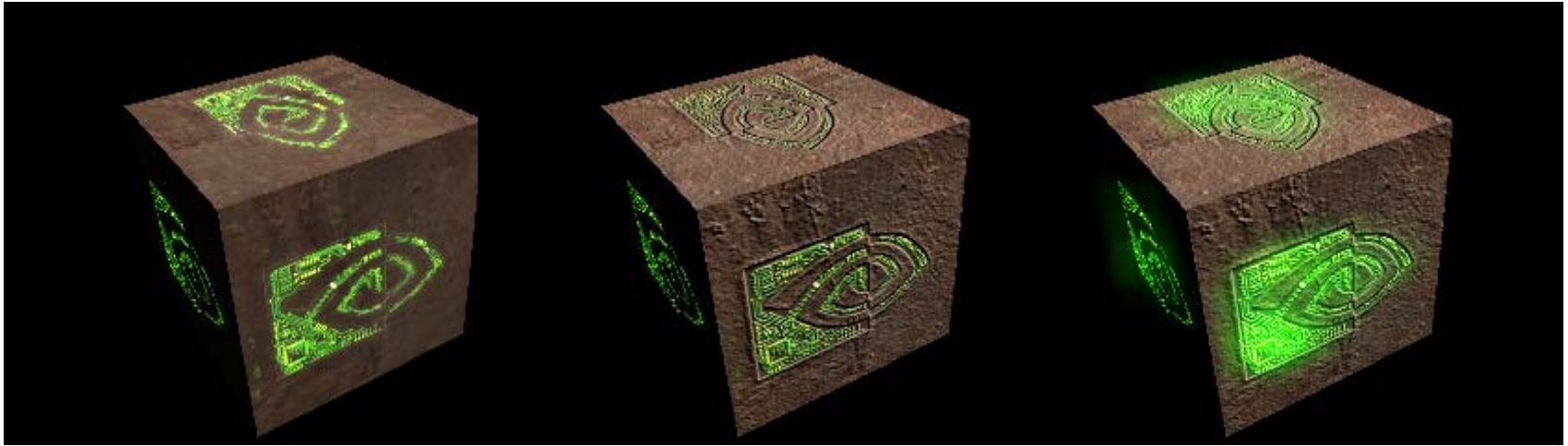


Motion Blur

Fragment Shaders

- JSR 239 is extensible
- NVIDIA GoForce 4800/5500
 - DOT3 bump/normal mapping
 - Environment-mapped bump mapping
 - Image processing
 - Motion Blur
 - Edge Detect
 - Bloom

Example Effects



Glow

Bump Mapping

HDR Blooming

Example Effects



Environment-Mapped Bump Mapping

Fragment Shaders

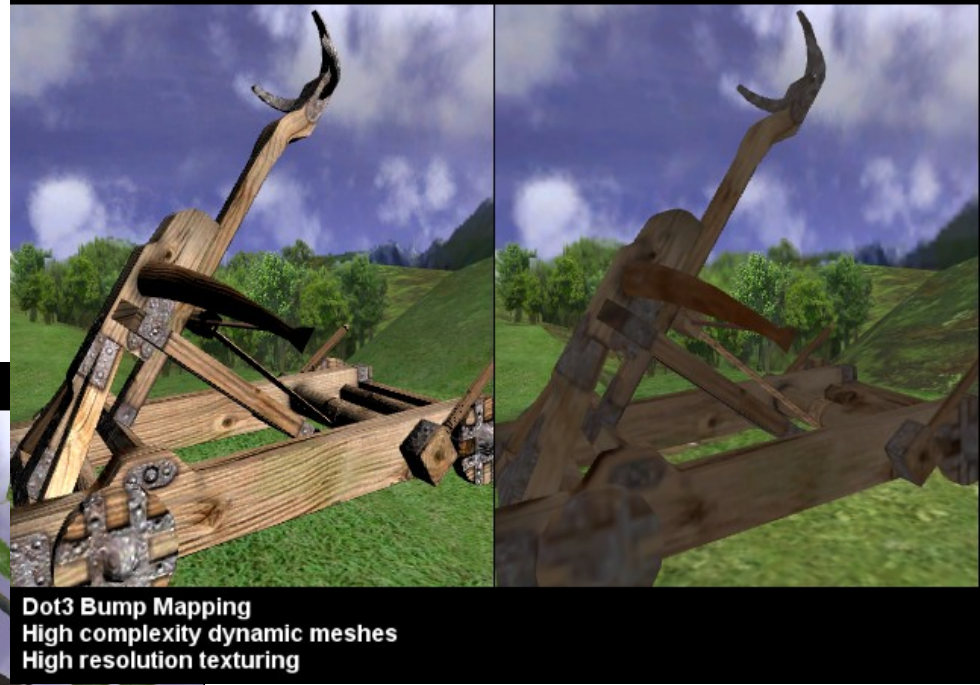
- Short programs that generate a fragment color to be passed to the final pixel processing stages
- Replaces the OpenGL ES Texturing Pipeline
- Operates on the Current Fragment
 - Four 4D constant registers (set by the application)
 - Current interpolated 4D color
 - Eight interpolated 2D texture coordinates
 - Up to 12 lookups into up to four textures

See the Difference

NVIDIA GoForce shader technology



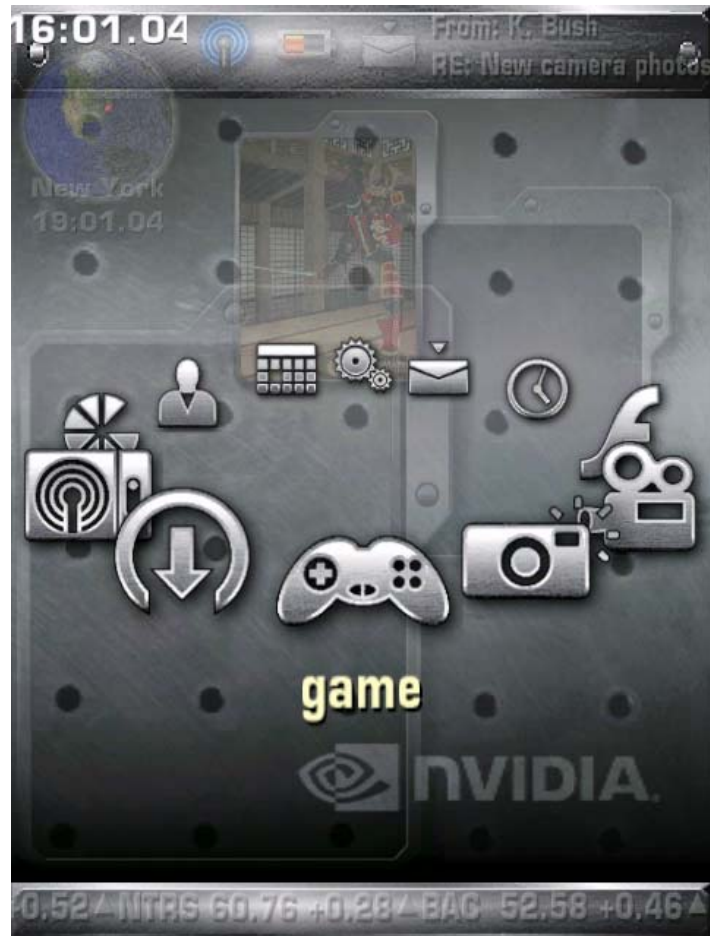
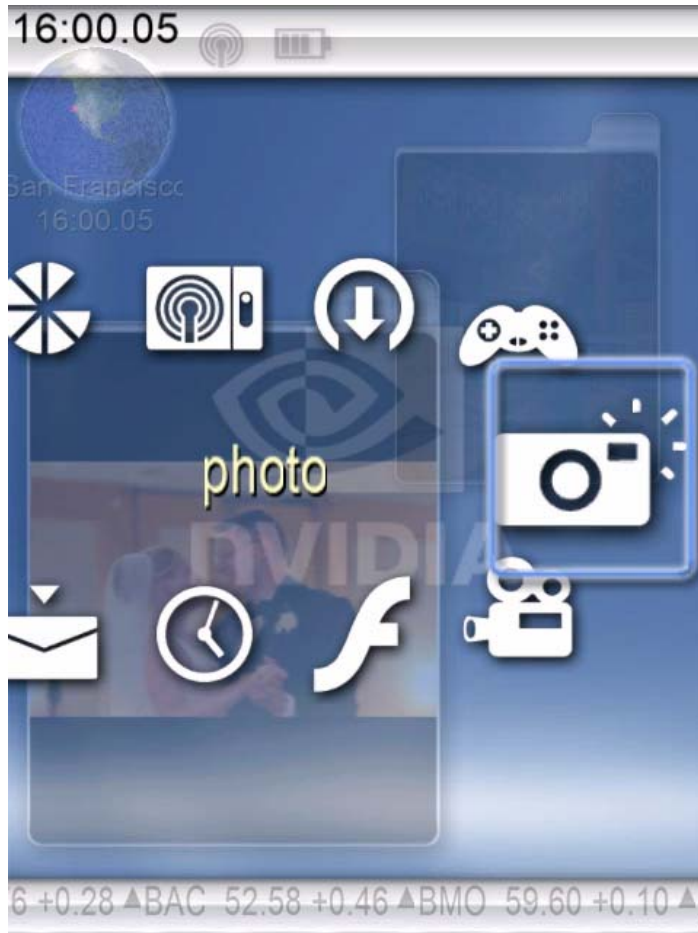
Combined power



Example: User Interface



Example: User Interface



Example: User Interface



- GPU: GoForce 5500
- Each application will render into a separate pbuffer
- Demo heavily uses
 - Fragment Shaders
 - PBuffers
 - HW Blending



DEMO

JSR-239 Demo



For More Information

- SVG-related sessions at JavaOneSM conference
 - TS-5525, TS-5628, TS-5743
- JSR 226 and JSR 239
 - <http://jcp.org/en/jsr/detail?id=226>
 - <http://jcp.org/en/jsr/detail?id=239>
- Khronos APIs
 - <http://www.khronos.org>
- Nvidia Handheld Developer Program
 - <http://developer.nvidia.com>
 - Email: mobile-dev@nvidia.com



Q&A





Whiz-Bang Graphics and Media Performance for Java Platform, Micro Edition (JavaME)

Pavel Petrosenko, Sun Microsystems, Inc.

Ashmi Bhanushali, NVIDIA Corporation

Jerry Evans, Sun Microsystems, Inc.

Nandini Ramani, Sun Microsystems, Inc.

Session: TS-5585