Developing Java™ Platform, Micro Edition Graphical Applications to Take Advantage of Hardware Acceleration

Jerry Evans
Sun Microsystems

Ashmi Bhanushali
Nvidia Corporation

Nandini Ramani
Sun Microsystems

TS-3024

Goal of This Talk

Learn about hardware acceleration for 2D and 3D graphical APIs in the Java™ Platform, Micro Edition (Java ME) and how your applications can take advantage of it

Agenda

Java ME
Overview of Graphics APIs
Hardware Acceleration
Demos
Adding a GPU to the System
CPU Related Optimizations
System Bus Related Optimizations
Taking Advantage of the GPU
JSR 239 Demos
Q&A

Agenda

Java ME

Overview of Graphics APIs

Hardware Acceleration

Demos

Adding a GPU to the System

CPU Related Optimizations

System Bus Related Optimizations

Taking Advantage of the GPU

JSR 239 Demos

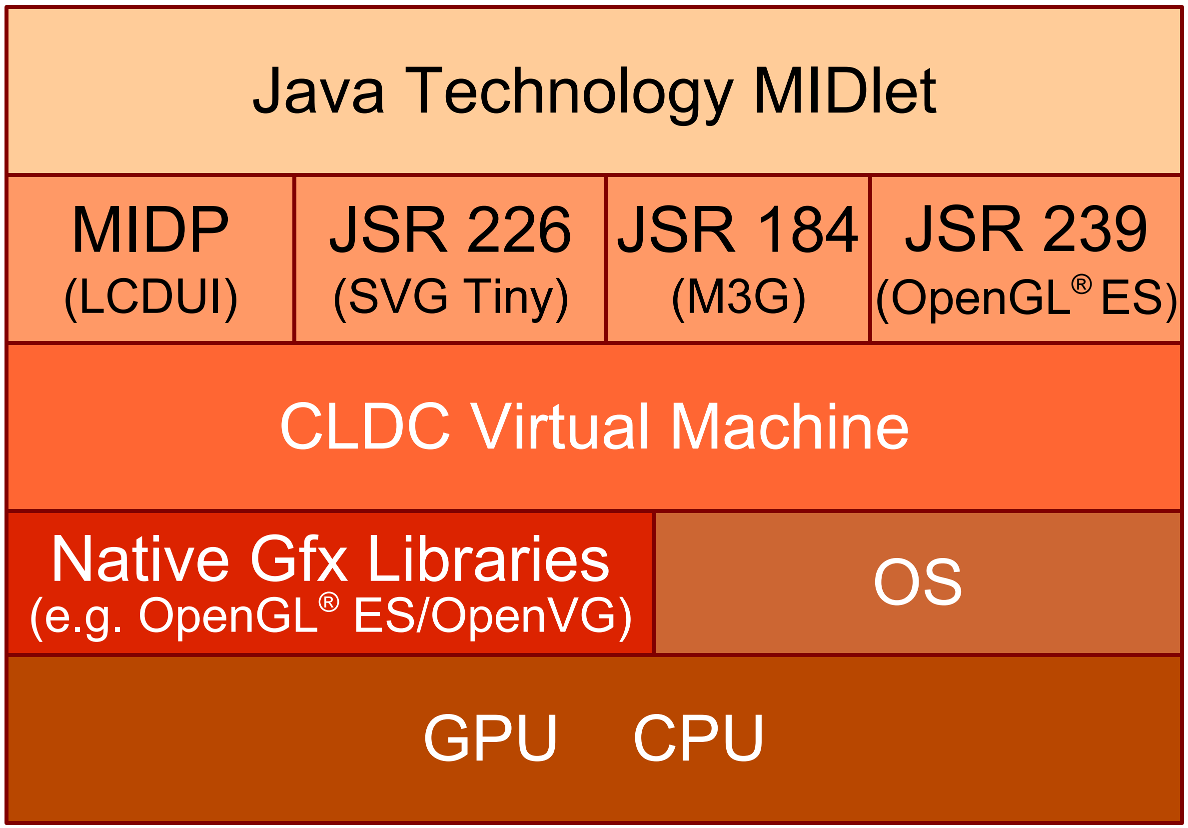
Q&A

Java ME Platform

Configuration and Profiles

- Volume phone
  - CLDC—Connected Limited Device Configuration Platform
  - MIDP—Mobile Information Device Profile
- High-end device
  - CDC—Connected Device Configuration Platform
  - Personal Basis Profile and Personal Profile
- This talk focuses on MIDP/CLDC

Java Platform, Micro Edition



Agenda

Java ME

Overview of Graphics APIs

Hardware Acceleration

Demos

Adding a GPU to the System

CPU Related Optimizations

System Bus Related Optimizations

Taking Advantage of the GPU

JSR 239 Demos

Q&A

Overview of Graphics APIs

- LCDUI for MIDP
- JSR 226—Scalable 2D Vector Graphics API for J2ME™
- JSR 184—Mobile 3D Graphics API for J2ME
- JSR 239—Java Bindings for OpenGL® ES

LCDUI for MIDP—Canvas

- Immediate mode API
- All drawing done within a paint() callback
- Can also draw to an off screen mutable image
- Suited for event based interaction
- High performance, low system overhead

LCDUI for MIDP—Game Canvas

- Immediate mode synchronous drawing
- Ideal for “platformer” games
- Sprite and tiled background
- Well suited for games
- Only on MIDP 2.0

JSR 226—Scalable 2D Vector Graphics API for J2ME

- Based on W3C SVG Tiny 1.1
- XML Grammar for rich interactive 2D graphics
- Java technology API to manipulate, animate and interact with SVG Tiny content
- No immediate mode rendering
- Display list model
- DOM can be edited using the JSR 226 API

JSR 226—Scalable 2D Vector Graphics API for J2ME (Cont.)

- Key features
  - Basic and complex shapes
  - Rich paint styles (gradients*, patterns*)
  - Rich text
  - Opacity*
  - Filter effects*
  - Scripting* and animation of dynamic content
  - Internationalization (i18n)

* SVG Tiny 1.2 features (JSR 287)

JSR 184—Mobile 3D Graphics API for J2ME

- 3D retained mode and immediate mode API
- Focus is retained mode (display list, scene graph)
- Defines standard file format—m3g
- Tools available to export graphics models in m3g format
- Easier to develop using 3D authoring tools
- Some animation support
- HW acceleration likely to be through OpenGL® ES

JSR 239—Java Bindings for OpenGL® ES

- OpenGL® ES is “embedded subset” of OpenGL®
- JSR 239 defines Java bindings to OpenGL® ES 1.0 and 1.1
- Immediate mode API
  - Highly flexible control of rendering
- Low level hardware oriented 3D API
  - Potentially higher performance
- Access to latest and greatest hardware features

JSR 239—Java Bindings for OpenGL® ES (Cont.)

- Key features
  - Full featured 3D API
  - 3D viewing pipeline
  - Lighting and shading
  - Texture mapping, cube maps
  - Fog
  - 2D sprites
  - Extensibility

Agenda

Java ME
Overview of Graphics APIs
Hardware Acceleration
Demos
Adding a GPU to the System
CPU Related Optimizations
System Bus Related Optimizations
Taking Advantage of the GPU
JSR 239 Demos
Q&A

Software vs. Hardware API Implementation

- Until recently, most 2D/3D APIs were implemented mostly in software
- 2D/3D hardware typically accessed through a native API
- With underlying hardware acceleration, Java based API performance improvement should be transparent
- However, this may be limited by:
  - Poor platform usage of the native graphics API
  - Application overhead exposed by higher graphics performance
  - Ultimate performance may require hardware-specific tweaks

Software vs. Hardware API Implementation (Cont.)

- 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 APIs are supported by a large number of hardware vendors for mobile devices
  - OpenVG (2D)
  - OpenGL® ES (3D)

Performance Issues

- Floating point vs. fixed point
  - API level vs. implementation level
    - SVG—Only float at API level, may be fixed in implementation
    - OpenGL® ES—Both float and fixed in API
  - Performance difference of float and fixed is highly variable
    - Depends on application, APIs, CPU, …
    - Probably an overall win to use fixed at API level, but maybe not enough to justify not using floats
    - Can use transforms to define user coordinates in a convenient fixed point space

Performance Issues (Cont.)

- Scene graph vs. immediate mode
  - Convenience and simplicity vs. control and flexibility
  - Scene graph API can preprocess graphical data
    - Store in optimized, possibly device dependent, format
    - Can provide features to off-load application,
  - Scene graph can offer improved performance if internal format is optimized for the device
  - However, scene graph formats typically do not evolve as quickly as graphics hardware and usually cannot be extended by application developers
  - Can combine immediate mode and scene graph
    - e.g. JSR 184, or mixing calls from different APIs

Performance Issues (Cont.)

- Mixing calls to different APIs
  - Unless the two API implementations have been designed to work together and share parts of the implementation, this involves two paths to the framebuffer
    - HW needs a private buffer (not in main memory)
    - Switching is expensive
  - Avoid mixing APIs as much as possible (for now)
    - Pick one API as the main API and minimize dynamic content from other API
  - OpenGL® ES and OpenVG have been designed to enable efficient hardware sharing
    - Future devices

Demos

Agenda

Java ME

Overview of Graphics APIs

Hardware Acceleration

Demos

Adding a GPU to the System

CPU Related Optimizations

System Bus Related Optimizations

Taking Advantage of the GPU

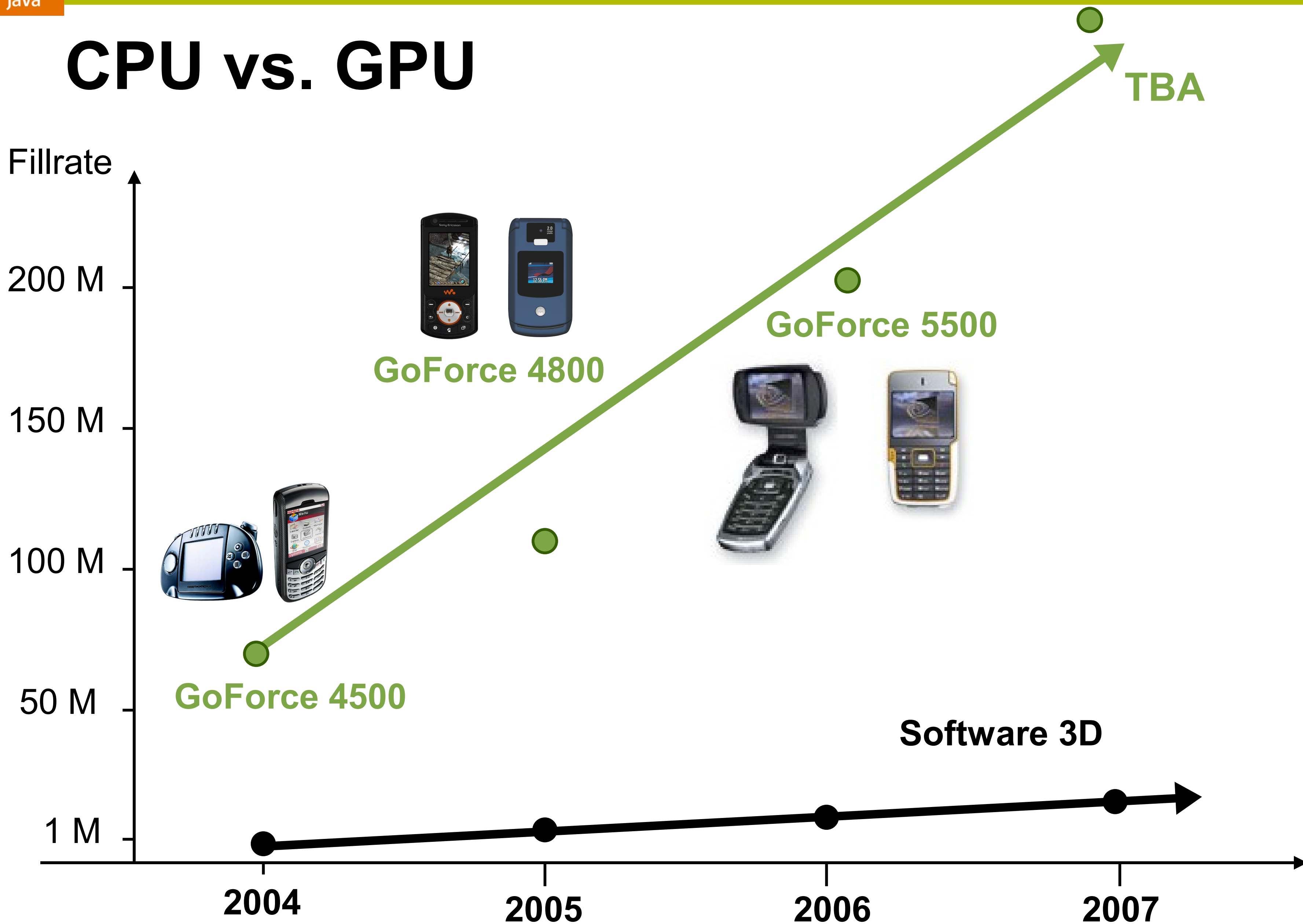
JSR 239 Demos

Q&A

Why Add a GPU to a Handheld Device?

- Graphics
  - Games
  - Snappy UI
- Video
- Audio
- Power
  - Graphics quality and performance per watt

CPU vs. GPU


TBA
Fillrate
200 M
150 M
100 M
50 M
1 M
GoForce 4800
GoForce 5500
GoForce 4500
Software 3D
2004
2005
2006
2007

Holistic System

- Now you’ve got a GPU in there
- Challenge is to keep all of the hardware resources busy
- Balancing act
  - CPU
  - GPU
  - System bus

Agenda

Java ME
Overview of Graphics APIs
Hardware Acceleration
Demos
Adding a GPU to the System
CPU Related Optimizations
System Bus Related Optimizations
Taking Advantage of the GPU
JSR 239 Demos
Q&A

Optimize CPU Work

- Simplify CPU load
  - Collision detection
  - Physics
  - Skinning/Animation
- Mind your memory
- Floating point emulation overhead
  - JSR 239 supports fixed point math

Optimize CPU Work

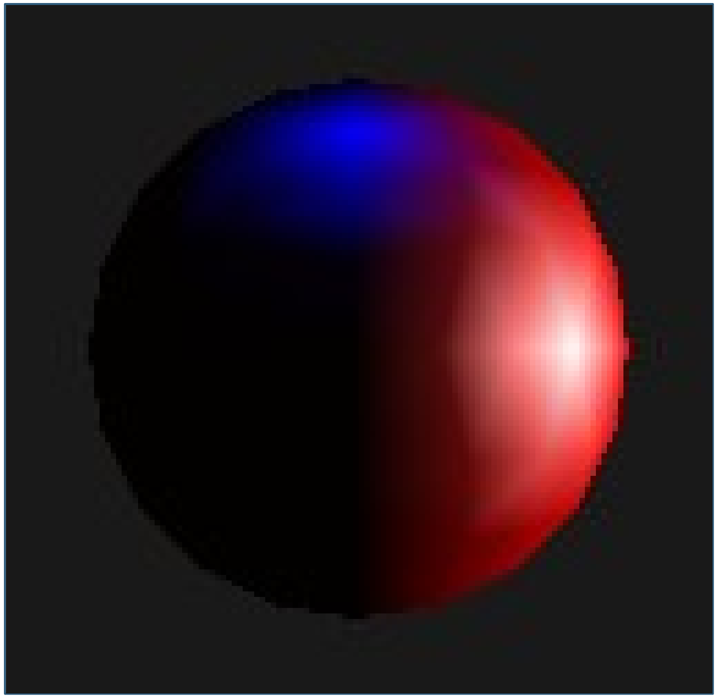
- Cull geometry at object level
  - Per triangle culling at app level means additional CPU work and cache strain
  - Will also benefit System Bus bandwidth
- Avoid multi-pass rendering
  - Use multitexturing
  - With JSR 239, you can use shader programs

Optimize CPU Work

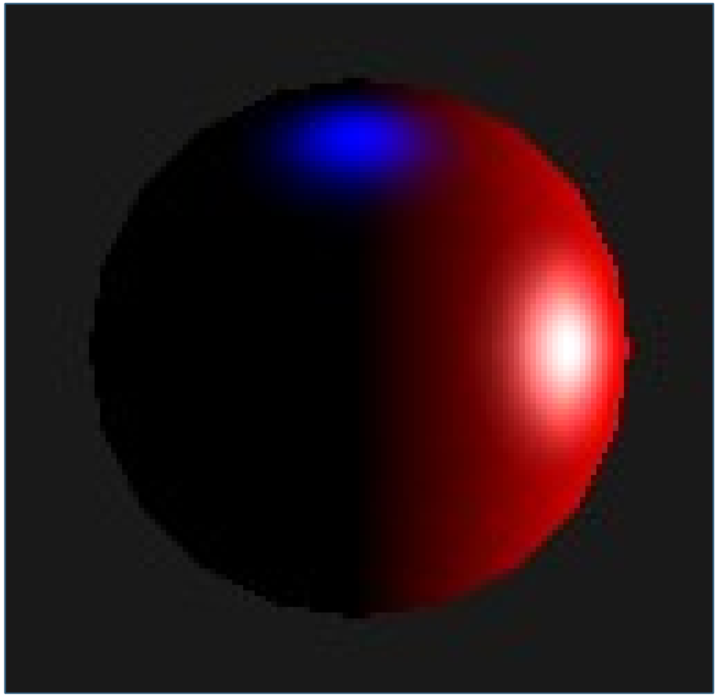
- Batch geometry
  - Into buckets of common states
  - Reduce number of state changes
- Avoid OpenGL® ES lighting
  - With JSR 239, use per pixel lighting

Specular Lighting Screenshots

Per Vertex



Per Pixel



Alternative Lighting Strategies

- Fake Phong highlights using multi-texture
- Pre-computed vertex lighting

Stuntcar Extreme
(Images Courtesy of Fathammer)

Agenda

Java ME
Overview of Graphics APIs
Hardware Acceleration
Demos
Adding a GPU to the System
CPU Related Optimizations
System Bus Related Optimizations
Taking Advantage of the GPU
JSR 239 Demos
Q&A

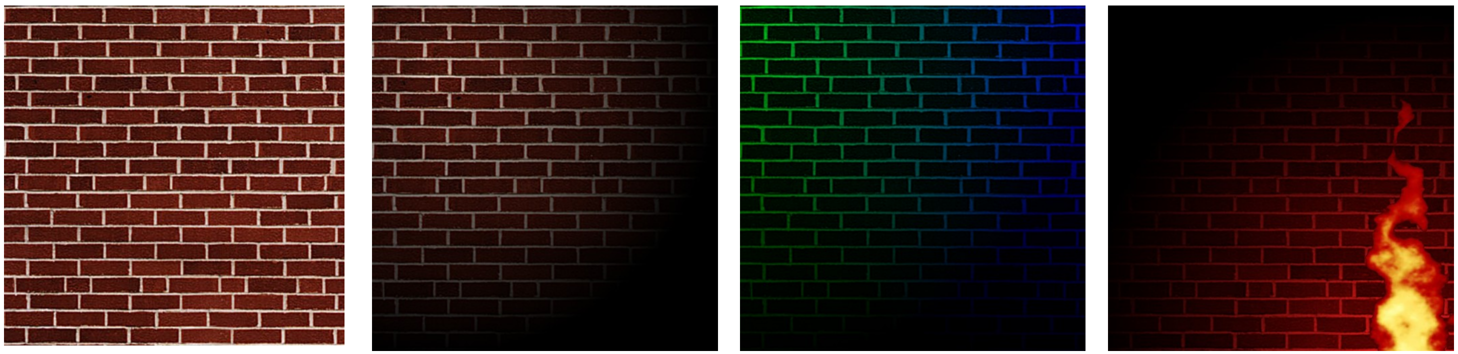
System Bus Related Optimizations

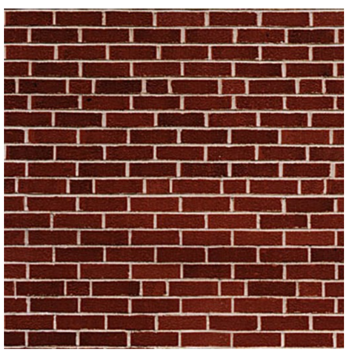
- Maximize the value of vertices rendered
  - Use LOD (Level of Detail)
  - Remodel objects for the small screen size
- Avoid multi-pass when possible
- With JSR 239, utilize VBO's (Vertex Buffer Objects)

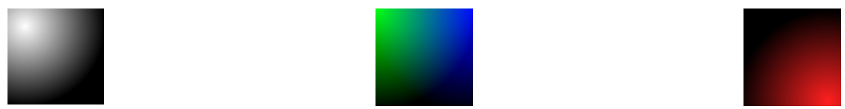
System Bus Related Optimizations

- Batch the geometry by texture
- Don't overflow GPU video memory
  - Video memory contains frame buffer, pbuffers and textures
  - GoForce 4800: 1280K SRAM
- Avoid allocating alpha if unused
- With JSR 239, use compressed texture formats

Leveraging Multi-Texture







Four Full Res (256 x 256) 4-bit = 128kb

Full Res Base + four 1/4 res lightmaps = 40kb

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

Agenda

Java ME
Overview of Graphics APIs
Hardware Acceleration
Demos
Adding a GPU to the System
CPU Related Optimizations
System Bus Related Optimizations
Taking Advantage of the GPU
JSR 239 Demos
Q&A

Taking Advantage of the GPU

- Dedicated 3D HW allows you to add more geometry, textures and shaders
- Freebies
  - Bilinear filtering
    - Perspective correction
  - Per-pixel fog, alpha blending, alpha test
- Use trilinear filtering, full scene anti-aliasing

A Few JSR 184 Observations…

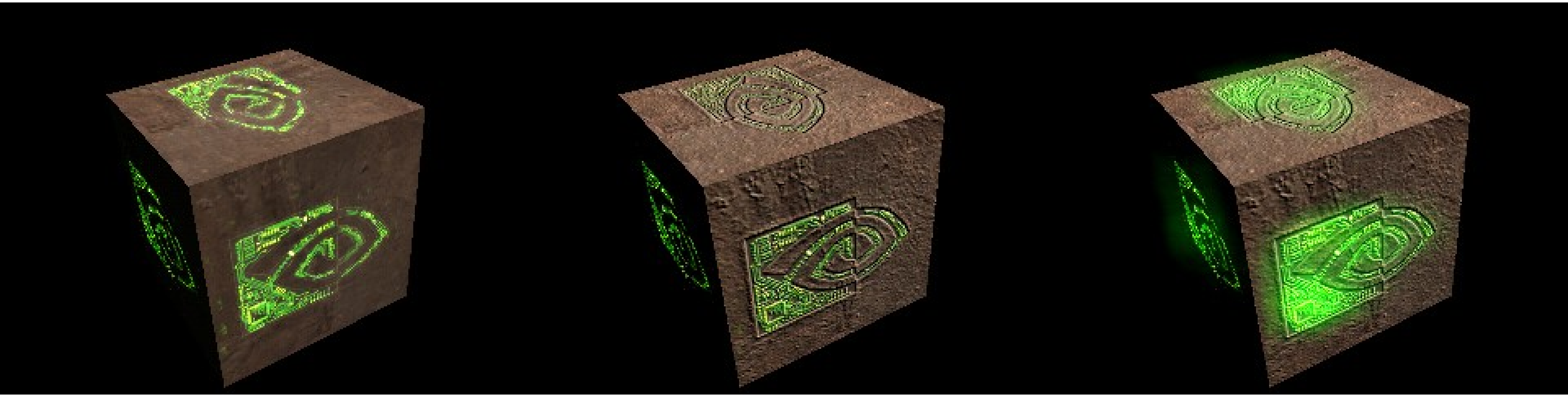
- From my experience…
  - Do not mix 2D with 3D
  - Do not use paint()/repaint() [MIDP 1.0]. Instead use GameCanvas.flushGraphics() [MIDP 2.0]
  - Object creation is expensive
    - Try to allocate them outside the main rendering loop and reuse
    - Be careful with skinning, animation and lighting

JSR 239 Demo

Fragment Shading

- JSR 239 is extensible
- NVIDIA GoForce 4800/5500
  - DOT3 bump/normal mapping
  - Environment-mapped bump mapping
  - Image processing
    - Motion blur, edge detect, blooms

Fragment Program SDK Sample



Vertex Lighting

DOT3 Bump Mapping

Glow + DOT3 Bump mapping

GoForce 5500 Demos

Environment-Mapped Bump Mapping + Bloom + reflection

Agenda

Java ME
Overview of Graphics APIs
Hardware Acceleration
Demos
Adding a GPU to the System
CPU Related Optimizations
System Bus Related Optimizations
Taking Advantage of the GPU
JSR 239 Demos
Q&A

JSR 239 Fragment Shading Demos

Agenda

Java ME
Overview of Graphics APIs
Hardware Acceleration
Demos
Adding a GPU to the System
CPU Related Optimizations
System Bus Related Optimizations
Taking Advantage of the GPU
JSR 239 Demos
Q&A

For More Information

- http://www.w3.org/TR/SVGMobile/
- http://www.w3.org/TR/SVGMobile12/
- http://jcp.org
- Register for Nvidia Handheld Developer Program
  - http://developer.nvidia.com
- Email: handset-dev@nvidia.com

Q&A

Developing Java™ Platform, Micro Edition Graphical Applications to Take Advantage of Hardware Acceleration

Jerry Evans
Sun Microsystems

Ashmi Bhanushali
Nvidia Corporation

Nandini Ramani
Sun Microsystems

TS-3024