Mobile 3D Graphics

Kari Pulli
Jani Vaarala
Ville Miettinen
Robert Simpson
Tomi Aarnio
Mark Callow

Nokia Research Center
Nokia
AMD
Nokia Research Center
HI Corporation
Today’s program: Morning

Start at 9:00

Intro & OpenGL ES overview
40 min, Kari Pulli

Using OpenGL ES 1.x
45 min, Jani Vaarala

OpenGL ES on PyS60
5 min, Kari Pulli

Break 10:30 – 11:00

OpenGL ES performance considerations
40 min, Ville Miettinen

OpenGL ES 2.0
50 min, Robert Simpson

Break 12:30
Today’s program: Afternoon

Start at 14:00

M3G Intro
5 min, Kari Pulli

M3G API overview
60 min, Tomi Aarnio

M3G in the Real World 1
25 min, Mark Callow

Break 15:30 – 16:00

M3G in the Real World 2
55 min, Mark Callow

M3G 2.0
25 min, Tomi Aarnio

Closing & Q&A
10 min, Kari Pulli

Finish at 17:30
Evolution of the Computer

Mainframe computer → Mini computer → Personal computer

Laptop computer → Multimedia Computer
Pervasive Mobile Computing

Mobile phones are the largest and fastest growing market - ever

The largest ever market opportunity for the graphics industry

Handsets are becoming personal computing platform

Not “just” phones: A real computer in your hand

Sophisticated media processing is a key

Just like it has been on the PC

Games are one of the first handheld media applications
Current expectation:

3 billion mobile subscribers by 2007.

Over 1 billion wireless broadband subscribers by 2009.

Up to 90% of the 6 billion will have mobile coverage by 2010.

Sources: Nokia 2005 & 2006, GSM Association 2006
Towards the 3 Billion Milestone

Mobile phone subscriptions globally, millions

Source: Nokia

Current global penetration 33%

3 billion in 2007

Source: Nokia
Challenge? Power!

Power is the ultimate bottleneck

Usually not plugged to wall, just batteries

Batteries don’t follow Moore’s law

Only 5-10% per year
Gene’s law

"power use of integrated circuits decreases exponentially" over time => batteries will last longer

Since 1994, the power required to run an IC has declined 10x every 2 years

But the performance of 2 years ago is not enough

Pump up the speed

Use up the power savings
Challenge? Thermal mgt!

But ridiculously good batteries still won’t be the miracle cure

The devices are small
Generated power must get out
No room for fans
Challenge?  Thermal mgt!

Thermal management must be considered early in the design.

- Hot spot would fry electronics
- Or at least inconvenience the user…

Conduct the heat through the walls, and finally release to the ambient.
Changed? Displays!

Resolution

- S60: 320 x 240
- Communicators: 640 x 200
- Internet tablets like N800: 800 x 480

Color depth

- Not many new B/W phones
- 12 / 16 / 18 / … bit RGB
Future? Displays!

Physical size remains limited

- TV-out connection
- Near-eye displays?
- Projectors?
- Roll-up flexible displays?

allaboutssymbian.com
Moore’s law in action

3410: ARM 7 @ 26MHz
  Not much caching, narrow bus

6600: ARM 9 @ 104MHz
  Decent caching, better bus

6630: ARM 9 @ 220MHz
  Faster memories

N93: ARM 11 @ 330MHz
  HW floating-point unit
  3D HW
State-of-the-art in 2001: GSM world

The world’s most played electronic game?

According to The Guardian (May 2001)

Communicator demo 2001

Remake of a 1994 Amiga demo

<10 year from PC to mobile
State-of-the-art in 2001: Japan

High-level API with skinning, flat shading / texturing, orthographic view
State-of-the-art in 2002: GSM world

3410 shipped in May 2002

A SW engine: a subset of OpenGL including full perspective (even textures)

3D screensavers (artist created content)

FlyText screensaver (end-user content)

a 3D game
State-of-the-art in 2002: Japan

Gouraud shading, semi-transparency, environment maps

I-3D PolyGame Boxing
@ Hi Vanguard•REZO, BNW

Ulala Channel J
©SEGA/UGA,2001 ©SEGA/UGA,2002

KDDI Au 3D Launcher
©SAN-X•GREEN CAMEL

3d menu
State-of-the-art in 2003: GSM world

N-Gage ships

Lots of proprietary 3D engines on various Series 60 phones
State-of-the-art in 2003: Japan

Perspective view, low-level API

Ridge Racer
@ Namco

Mission Commander
Multi player Fps Game
© IT Telecom
Mobile 3D in 2004

6630 shipped late 2004
First device to have both
OpenGL ES 1.0 (for C++) and
M3G (a.k.a JSR-184, for Java) APIs

Sharp V602SH in May 2004
OpenGL ES 1.0 capable HW
but API not exposed
Java / MascotCapsule API
2005 and beyond: HW
Mobile graphics evolution snapshot

2D

Spider-Man 2
Activision

Spider-Man 2 3D: NY Subway
Sony Pictures

Spider-Man 2: The Hero Returns
Sony Pictures

Software 3D

Accelerated 3D
Mobile 3D APIs

Native C/C++ Applications

Java Applications

M3G (JSR-184)

Java UI API

OpenGL ES

Java Virtual Machine

Graphics Hardware

Operating System (Symbian, Linux, …)
Overview: OpenGL ES

Background: OpenGL & OpenGL ES

- OpenGL ES 1.0
- OpenGL ES 1.1
- EGL: the glue between OS and OpenGL ES

How can I get it and learn more?
What is OpenGL?

The most widely adopted graphics standard
most OS’s, thousands of applications
Map the graphics process into a pipeline
matches HW well

A foundation for higher level APIs
Open Inventor; VRML / X3D; Java3D; game engines
What is OpenGL ES?

OpenGL is just too big for Embedded Systems with limited resources
memory footprint, floating point HW

Create a new, compact API
mostly a subset of OpenGL
that can still do almost all OpenGL can
OpenGL ES 1.0 design targets

- Preserve OpenGL structure
- Eliminate un-needed functionality
  - redundant / expensive / unused
- Keep it compact and efficient
  - <= 50KB footprint possible, without HW FPU
- Enable innovation
  - allow extensions, harmonize them
- Align with other mobile 3D APIs (M3G / JSR-184)
Adoption

Symbian OS, S60
Brew
PS3 / Cell architecture

Sony’s arguments: Why ES over OpenGL
  • OpenGL drivers contain many features not needed by game developers
  • ES designed primarily for interactive 3D app devs
  • Smaller memory footprint
Outline

Background: OpenGL & OpenGL ES

OpenGL ES 1.0

OpenGL ES 1.1

EGL: the glue between OS and OpenGL ES

How can I get it and learn more?
OpenGL ES Pipe

Here's the OpenGL ES pipeline stages:
- vertices
- primitives
- fragments
Vertex pipeline

- Vertex buffer
  - Current color
  - Current normal
  - Current vertex
  - Current texcoord 0
  - Current texcoord n-1

- Vertex array

- Matrix control
  - M
  - T_0
  - ... T_{n-1}
  - P
  - M^{-T}

- Primitive assembly

- Material control
- Lighting

- User clip
Primitive processing
Fragment pipeline

- Rasterization & interpolation
  - Texture memory
    - Texture 0 application
      - Texel 0 fetch
    - ... (number of textures)
    - Texture n-1 application
      - Texel n-1 fetch
  - Frame Buffer (Color, Depth, Stencil)
    - Copy pixels
      - Alpha test
      - Multi-sample
      - Scissor test
      - Coverage generation
      - Fog
      - Depth test
      - Blending
      - Dithering
      - Logic Op
      - Masking
    - Read pixels
Functionality: in / out? (1/7)

Convenience functionality is **OUT**

- **GLU** (utility library)
- evaluators (for splines)
- feedback mode (tell what would draw without drawing)
- selection mode (for picking, easily emulated)
- display lists (collecting and preprocessing commands)

```c
gluOrtho2D(0,1,0,1)
vs.
glOrtho(0,1,0,1,-1,1)
```

```c
glNewList(1, GL_COMPILE)
myFuncThatCallsOpenGL()
glEndList()
...
glCallList(1)
```
Functionality: in / out? (2/7)

Remove old complex functionality

glBegin – glEnd (OUT); vertex arrays (IN)

new: coordinates can be given as bytes

```c
static const GLbyte verts[4 * 3] =
{   -1, 1, 1, 1, 1, 1,
    1, -1, 1, -1, -1, 1 };  
static const GLubyte colors[4 * 3] =
{ 255, 0, 0, 255, 0, 0,
    0,255, 0, 0,255, 0 };  
glVertexPointer( 3,GL_BYTE,0, verts );
glColorPointerf( 3,GL_UNSIGNED_BYTE,
                0, colors );
glDrawArrays( GL_TRIANGLE_STRIP,
                0, 4 );
```
Functionality: in / out? (3/7)

Simplify rendering modes
  double buffering, RGBA, no front buffer access

Emulating back-end missing functionality is expensive or impossible
  full fragment processing is **IN**
    alpha / depth / scissor / stencil tests,
    multisampling,
    dithering, blending, logic ops)
Functionality: in / out? (4/7)

Raster processing

ReadPixels **IN**, DrawPixels and Bitmap **OUT**

Rasterization

**OUT**: PolygonMode, PolygonSmooth, Stipple
Functionality: in / out? (5/7)

2D texture maps **IN**

1D, 3D, cube maps **OUT**

borders, proxies, priorities, LOD clamps **OUT**

multitexturing, texture compression **IN** (optional)

texture filtering (incl. mipmaps) **IN**

new: paletted textures **IN**
Functionality: in / out? (6/7)

Almost full OpenGL light model **IN**
back materials, local viewer,
separate specular **OUT**

Primitives

**IN:** points, lines, triangles

**OUT:** quads & polygons
Functionality: in / out? (7/7)

Vertex processing

**IN:** transformations

**OUT:** user clip planes, texcoord generation

Support only static queries

**OUT:** dynamic queries, attribute stacks

application can usually keep track of its own state
Floats vs. fixed-point

Accommodate both

- integers / fixed-point numbers for efficiency
- floats for ease-of-use and being future-proof

Details

16.16 fixed-point: add a decimal point inside an int

```
    glRotatef( 0.5f, 0.f, 1.f, 0.f );
```

vs.

```
    glRotatex( 1 << 15, 0, 1 << 16, 0 );
```

get rid of doubles
Outline

Background: OpenGL & OpenGL ES

OpenGL ES 1.0

OpenGL ES 1.1

EGL: the glue between OS and OpenGL ES

How can I get it and learn more?
OpenGL ES 1.1: core

Buffer Objects
allow caching vertex data

Better Textures
>= 2 tex units, combine (+,-,interp), dot3 bumps, auto mipmap gen.

User Clip Planes
portal culling (>= 1)

Point Sprites
particles as points not quads, attenuate size with distance

State Queries
enables state save / restore for middleware
Bump maps

Double win

increase realism

reduce internal bandwidth -> increase performance
OpenGL ES 1.1: optional

Draw Texture
fast drawing of pixel rectangles
using texturing units
(data can be cached),
constant Z, scaling

Matrix Palette
vertex skinning
(>= 3 matrices / vertex, palette >= 9)
Outline

Background: OpenGL & OpenGL ES
OpenGL ES 1.0
OpenGL ES 1.1
EGL: the glue between OS and OpenGL ES
How can I get it and learn more?
EGL glues OpenGL ES to OS

EGL is the interface between OpenGL ES and the native platform window system

similar to GLX on X-windows, WGL on Windows

facilitates portability across OS’s (Symbian, Linux, …)

Division of labor

EGL gets the resources (windows, etc.) and displays the images created by OpenGL ES

OpenGL ES uses resources for 3D graphics
EGL surfaces

Various drawing surfaces, rendering targets

*windows* – on-screen rendering
  (“graphics” memory)

*pbuffers* – off-screen rendering
  (user memory)

*pixmaps* – off-screen rendering
  (OS native images)
EGL context

A rendering context is an abstract OpenGL ES state machine

- stores the state of the graphics engine
- can be (re)bound to any matching surface
- different contexts can share data
  - texture objects
  - vertex buffer objects
- even across APIs (OpenGL ES, OpenVG, later others too)
Main EGL 1.0 functions

Getting started

eglInitialize() / eglTerminate(), eglGetDisplay(), eglGetConfigs() / eglChooseConfig(), eglCreate\texttt{X}Surface() (X = Window | Pbuffer | Pixmap), eglCreateContext()

eglMakeCurrent( display, drawsurf, readsurf, context )

binds context to current thread, surfaces, display
Main EGL 1.0 functions

eglSwapBuffer(display, surface)
posts the color buffer to a window

eglWaitGL(), eglWaitNative(engine)
provides synchronization between OpenGL ES and native (2D) graphics libraries

eglCopyBuffer(display, surface, target)
copy color buffer to a native color pixmap
EGL 1.1 enhancements

Swap interval control
  specify # of video frames between buffer swaps
  default 1; 0 = unlocked swaps, >1 save power

Power management events
  PowerMgmnt event => all Context lost
  Display & Surf remain, Surf contents unspecified

Render-to-texture [optional]
  flexible use of texture memory
Outline

Background: OpenGL & OpenGL ES
OpenGL ES 1.0 functionality
OpenGL ES beyond 1.0
EGL: the glue between OS and OpenGL ES
How can I get it and learn more?
SW Implementations

Vincent

Open-source OpenGL ES library

http://www.vincent3d.com/
http://sourceforge.net/projects/ogl-es

Reference implementation

Wraps on top of OpenGL

http://www.khronos.org/opengles/documentation/gles-1.0c.tgz
HW implementations

There are many designs

The following slides gives some idea

rough rules of thumb

from a couple to dozens of MTri / sec (peak)

1 pixel / clock

clock speeds 50MHz – 200+MHz

power consumption should be ~ 10’s of mW
Bitboys

Graphics processors

G12: OpenVG 1.0
G34: OpenGL ES 1.1
G40: OpenGL ES 2.0, GLSL, OpenVG 1.0 vertex and pixel shader, Flipquad antialiasing, Max clock 200MHz

Partners / Customers

NEC Electronics
Hybrid Graphics (drivers)
ATI

- **Imageon 2300**
  - OpenGL ES 1.0
  - Vertex and raster HW
    - 32-bit internal pipe
    - 16-bit color and Z buffers
- **Imageon 3D**
  - OpenGL ES 1.1
  - 3M Tri/s, 100M Pix/s @ 100 MHz
  - 2nd gen. Imageon 3D adds
    - OpenGL ES 1.1 extension pack
    - Vertex shader
    - HyperZ
    - Audio codecs, 3D audio
- **Partners, customers**
  - Qualcomm
  - LG SV360, KV3600
  - Zodiac

AMD bought ATI
AMD Graphics IP

3D Processors
- AMD Z430 & Z460
  - Unified Shader architecture derived from the Xbox 360 Xenos core
  - OpenGL ES 2.0
  - OpenGL ES 1.1 backwards compatible
  - OpenVG 1.x

Vector Graphics Processors
- AMD Z160 & Z180
  - Native, high-performance OpenVG acceleration
  - OpenVG 1.x
  - 16 x antialiasing

All processors are designed to be combined to achieve native HW acceleration of both OpenGL ES 2.0 and OpenVG 1.x for unrivalled performance and image quality.
Falanx

» Mali 110
  » OpenGL ES 1.1 + extensions
  » 4x / 16x full screen anti-aliasing
  » Video codecs (e.g., MPEG-4)
  » 170-400k logic gates + SRAM
  » 2.8M Tri / s, 100M Pix / s with 4xAA

» Mali 200
  » OpenGL ES 2.0, OpenVG, D3D
  » 5M Tri / s, 100M Pix / s, 11 instr. / cycle

» Partners / Customer
  » Zoran

<table>
<thead>
<tr>
<th>Core Function</th>
<th>MALI95</th>
<th>MALI110</th>
<th>MALI200</th>
<th>MALIISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Function</td>
<td>Pixel Shader</td>
<td>Pixel Shader</td>
<td>Programmable Pixel Shader</td>
<td>Programmable Vertex Shader</td>
</tr>
<tr>
<td>Gate Count</td>
<td>100K</td>
<td>235K</td>
<td>400K-500K</td>
<td>150K</td>
</tr>
<tr>
<td>Max Freq</td>
<td>200MHz</td>
<td>200MHz</td>
<td>200MHz</td>
<td>150MHz</td>
</tr>
<tr>
<td>Anti-Aliasing</td>
<td>4X / 16X</td>
<td>4X / 16X</td>
<td>4X / 16X</td>
<td>4X / 16X</td>
</tr>
<tr>
<td>OpenGL ES 1.1</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>OpenGL ES 2.0</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>OpenVG 1.0</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DirectX w/Vista Extensions</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Deferred Vertex Shading</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MPEG-4/AVC 264</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>H.264 Decode</td>
<td>15fps</td>
<td>15fps</td>
<td>30fps</td>
<td>30fps</td>
</tr>
<tr>
<td>OpenGL MAX</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
ARM® Mali™ Architecture

- Compared to traditional immediate mode renderer
  - 80% lower per pixel bandwidth usage, even with 4X FSAA enabled
  - Efficient memory access patterns and data locality: enables performance even in high latency systems
- Compared to traditional tile-based renderer
  - Significantly lower per-vertex bandwidth
  - Impact of scene complexity increases is substantially reduced
- Other architectural advantages
  - Per frame autonomous rendering
  - No renderer state change performance penalty
  - On-chip z / stencil / color buffers
    - minimizes working memory footprint
  - Acceleration beyond 3D graphics (OpenVG etc.)

<table>
<thead>
<tr>
<th>Mali200</th>
<th>MaliGP2</th>
<th>Mali55</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anti-Aliasing</td>
<td>4X / 16X</td>
<td>4X / 16X</td>
</tr>
<tr>
<td>OpenGL®ES 1.x</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>OpenGL®ES 2.x</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>OpenGL®ES 2.x</td>
<td>YES</td>
<td>NA</td>
</tr>
<tr>
<td>Max CLK</td>
<td>275MHz</td>
<td>275MHz</td>
</tr>
<tr>
<td>Fill rate Mpix / s</td>
<td>275</td>
<td>NA</td>
</tr>
<tr>
<td>Triangles / s</td>
<td>9M</td>
<td>9M</td>
</tr>
</tbody>
</table>
PICA graphics core

3D Features
- OpenGLES 1.1
- DMP’s proprietary “Maestro” shader extensions
  - Very high quality graphics with easier programming interface
  - Per-fragment lighting,
  - Shadow-mapping,
  - Procedural texture,
  - Polygon subdivision (Geo shader), and
  - Gaseous object rendering.

Hardware Features
- Performance: 40Mtri/s,
  400Mpixel/s@100MHz
- Power consumption: 0.5-1mW/MHz
- Max. clock freq. 400MHz (65nm)
Fujitsu Graphics Controllers

- Optimized for automotive environment
  - Extended temp range (-40...+85degC or -40...+105degC)
  - No external active or passive cooling required
  - Long term availability (devices from 1998 still in full mass production!)
  - Fulfills the latest qualification requirements from automotive industry
  - Automotive network interfaces included on-chip
  - Dedicated competence center in Munich for automotive graphics

- Used in many major car brands for:
  - Onboard navigation systems (2D and 3D)
  - Cluster Instrumentation (incl. virtual dashboards)
  - Rear seat entertainment systems
  - Head-up displays
  - Night vision systems

- Also used today in:
  - Flight instrumentation
  - Marine displays
  - Medical, etc...

<table>
<thead>
<tr>
<th>Feature</th>
<th>This generation (in MP)</th>
<th>Next generation (tba)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>~2 GB/s</td>
<td>~6 GB/s</td>
</tr>
<tr>
<td>Performance</td>
<td>~5MT/s ; 200Mpix/s</td>
<td>~10MT/s ; 500Mpix/s</td>
</tr>
<tr>
<td>Graphic processing</td>
<td>OpenGL ES 1.1</td>
<td>OpenGL ES 2.0 ; OpenVG</td>
</tr>
<tr>
<td># of video inputs</td>
<td>2 video inputs</td>
<td>4 video inputs (up to HD)</td>
</tr>
<tr>
<td># of display outputs</td>
<td>2 display outputs</td>
<td>2 display outputs with dual view option</td>
</tr>
</tbody>
</table>
Imagination Technologies
POWERVR MBX & SGX 2D/3D Acceleration

- **5th Generation Tile Based Deferred Rendering**
  - Market Proven Advanced Tiling Algorithms
  - Order-independent Hidden Surface Removal
  - Lowest silicon area, bandwidth and power
  - Excellent system latency tolerance

- **POWERVR SGX: OpenGL ES 2.0 in Silicon Now**
  - Scalable from 1 to 8 pipelines and beyond
  - Programmable multi-threaded multimedia GPU
  - Optimal load balancing scheduling hardware
  - Vertex, Pixel, Geometry shaders + image processing

- **Partners/Customer**
  - TI, Intel, Renesas, Samsung, NXP, NEC, Freescale, Sunplus, Centrality & others unannounced

www.powervrinsider.com
Market-leading Ecosystem with more than 1650 members

---

**POWERVR MBX: The de-facto standard for mobile graphics acceleration, with >50 PowerVR 3D-enabled phones shipping worldwide**

<table>
<thead>
<tr>
<th></th>
<th>PowerVR MBX Family</th>
<th>PowerVR SGX Family</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenGL</td>
<td>ES1.1</td>
<td>2.0, ES1.1 and ES2.0</td>
</tr>
<tr>
<td>Direct3D</td>
<td>Mobile</td>
<td>Mobile, 9L and 10.1</td>
</tr>
<tr>
<td>OpenVG</td>
<td>1.0</td>
<td>1.0.1 and 1.1</td>
</tr>
<tr>
<td>Triangles/Sec</td>
<td>1.7M ... 3.7M</td>
<td>1M ... 15.5M</td>
</tr>
<tr>
<td>Pixels/Sec</td>
<td>135M ... 300M</td>
<td>50M ... 500M</td>
</tr>
</tbody>
</table>

Performance quoted at 100MHz for MBX, MBX Lite and for SGX510 to SGX545. Peak SoC achievable performance not quoted, e.g. <50% Shader load for Tri/Sec. Performance scales with clock speeds up to 200MHz and beyond. Planned future cores will offer higher performance levels.
Mitsubishi

• Z3D family
  – Z3D and Z3D2 out in 2002, 2003
    • Pre-OpenGL ES 1.0
    • Embedded SRAM architecture
  – Z3D3 in 2004
    • OpenGL ES 1.0, raster and vertex HW
    • Cache architecture
    • @ 100 MHz: 1.5M vtx / s, 50-60 mW, ~250 kGates
  – Z3D4 in 2005
    • OpenGL ES 1.1

• Partners / Customers
  Several Japanese manufacturers
**GiPump™ Series**

**GiPump™ NX1005**
; Mobile 3D graphics acc. with camera control functions
- OpenGL ES 1.1 / GIGA / JSR184
- 5M poly/s, 80M pix/s @ 80MHz, JPEG codec (3M pixel), ~QVGA display
- Cellular phone, smart phone, etc.

**GiPump™ NX1007**
; High end 3D graphics acc. for mobile
- OpenGL ES 1.1 + Ext. / GIGA / JSR184
- 12.5M poly/s, 200M pix/s @ 100MHz, ~SVGA display, PIP supports
- PND, PMP, game device, mobile device, etc.

**GiPump™ NX1008**
; Mobile 3D graphics acc. with stereoscopic display
- OpenGL ES 1.1 / GIGA / JSR184
- 5M poly/s, 80M pix/s @ 80MHz, ~QVGA display, stereoscopic display
- Cellular phone, smart phone, etc.

**GiPump™ NX1009**
; Economical mobile 3D graphics accelerator
- OpenGL ES 1.1 + Ext. / GIGA / JSR184
- 12.5M poly/s, 200M pix/s @ 100MHz, ~SVGA display, boost mode
- Cellular phone, Smart phone, etc.

**GiPump™ NX2001**
; 3D Graphics enhanced multimedia processor
- OpenGL ES 2.0 / 1.1 Ext. / JSR184 / D3DM
- 10M poly/s, 200M pix/s @ 200MHz, ~SVGA display
- PND, PMP, game device, mobile device, etc.

**GiPump™ Partners**: Samsung, SKT, Other Device Manufactures

* GIGA (Giga Instruction Giga Acceleration) - SK Telecom’s mobile 3D graphics platform.

**GiPump™ SDK**
- NXsdk with Emulator
- NXsdk Shader+
- NXm3g Engine
- NX3D Engine & Tools

**Service Solutions**

**Nexus Mobile Platform™**
Gaming Device Platform (OS: WinCE, Linux, RTOS, etc.)
To: Game Device Maker

**NX1008TK™**
3D Reference B/D
GiPump™ Integration Platform
To: Device Developer

**GiPump™ SDK**
- NXsdk with Emulator
- NXsdk Shader+
- NXm3g Engine
- NX3D Engine & Tools

**New Wave Digital Paradigm**
NVidia

GoForce 5500 handheld GPU

- 3D geometry and rasterization HW
- OpenGL ES 1.1, D3D Mobile, OpenVG 1.0
- 1.3M tri / s, 100M pix / s (@ 100 MHz)
- Programmable pixel micro shaders
- 40 bit signed non-int (overbright) color pipeline
- Dedicated 2D engine (bitblt, lines, alpha blend)
- Supersampled anti-aliasing, up to 6 textures
- <50mW avg. dynamic power cons. for graphics
- 10MPx1 camera support, XGA LCD, MPEG-4 video, audio

Partners / Customers

- Motorola, Sony Ericsson, Samsung, LG, Kyocera, O2, HTC, Marvell, Freescale, …
Sony PSP

- **Game processing unit**
  - **Surface engine**
    - tessellation of Bezier and splines
    - skinning (<= 8 matrices), morphing (<= 8 vtx)
    - HW T&L
    - 21 MTri / s (@ 100 MHz)
  - **Rendering engine**
    - basic OpenGL-style fixed pipeline
    - 400M pix / s (@ 100 MHz)
  - 2MB eDRAM

- **Media processing engine**
  - 2MB eDRAM
  - H.264 (AVC) video up to 720x480 @ 30fps
TAKUMI

- **GSHARK-TAKUMI Family**
  - **GP**
    - OpenGL ES 1.0
    - 0.5M tri/s @100MHz, 170Kgate
  - **GT**
    - OpenGL ES 1.1
    - 1.4M tri/s @100MHz, < 30mW
  - **G2**
    - OpenGL ES 1.1
    - 5M tri/s @100MHz

- **Concepts & Architecture**
  - Small Gate Counts
  - Low Power Consumption
  - Vertex Processor (T&L)
  - Dedicated 2D Sprite Engine
  - Target Application
    - Mobile Phone and Digital AV Equipments such as DTV, STB, DSC, PMP, etc.

- **Partners / Customers**
  - NEC Electronics
TC35711XBG

Programmable shader

Plan to support OpenGL ES2.0

Large embedded memory for
  Color and Z buffer
  Caches for vertex arrays, textures
  Display lists (command buffer)

50M vtx / sec, 400M pix / sec (@ 100 MHz)
  clocks up to 200MHz

WVGA LCD controller

13mm x 13mm x 1.2mm 449Ball BGA
Vivante GPU for Handheld

- OpenGL ES 1.1 & 2.0 and D3D 9.0
- Unified vertex & pixel shader
- Anti-Aliasing
- AXI/AHB interface
- GC500
  - 3 mm² die area in 65nm (1.8mm x 1.2mm)
  - 10 MPolygons/s and 100 MPixel/s at 200 MHz
  - 50mW GPU core power
- Scalable solution to 50 MPolygons/s and 1 GPixels/s (GC1000, GC4000)
- Silicon proven solution
- Designed into multiple 65nm SoCs
SDKs

Nokia S60 SDK (Symbian OS)
http://www.forum.nokia.com

Imagination SDK
http://www.pvrdev.com/Pub/MBX

NVIDIA handheld SDK
http://www.nvidia.com/object/hhsdk_home.html

Brew SDK & documentation
http://brew.qualcomm.com

see http://people.csail.mit.edu/kapu/EG_08
Mobile 3D Graphics with OpenGL ES and M3G

Kari Pulli, Tomi Aarnio, Ville Miettinen, Kimmo Roimela, Jani Vaarala

http://www.graphicsformasses.com/
Questions?
Using OpenGL ES

Jani Vaarala
Nokia
Using OpenGL ES

Simple OpenGL ES example
EGL configuration selection
Texture matrix example
Fixed point programming
Converting existing code
“Hello OpenGL ES”
Hello OpenGL ES, EGL initialization

/* ==============================================================
 * "Hello OpenGL ES" OpenGL ES code.
 * 
 * Eurographics 2008 tutorial.
 * 
 * Copyright: Jani Vaarala
 * ==============================================================
 */

#include <GLES/gl.h>
#include <GLES/egl.h>

EGLDisplay display;
EGLContext context;
EGLSurface surface;
EGLConfig config;
Hello OpenGL ES, EGL initialization

EGLint attrib_list[ ] =
{
    EGL_BUFFER_SIZE,  16,
    EGL_DEPTH_SIZE,   15,
    EGL_SURFACE_TYPE, EGL_WINDOW_BIT,
    EGL_NONE
};

void init_egl(void)
{
    EGLint numOfConfigs;

    display = eglGetDisplay( EGL_DEFAULT_DISPLAY );
    eglInitialize( display, NULL, NULL );
    eglChooseConfig( display, attrib_list, &config, 1 , &numOfConfigs );
    surface = eglCreateWindowSurface( display, config, WINDOW( ), NULL );
    context = eglCreateContext( display, config, EGL_NO_CONTEXT, NULL );
    eglMakeCurrent( display, surface, surface, context );
}
Hello OpenGL ES, OpenGL ES part

#include <GLES/gl.h>

static const GLbyte vertices[3 * 3] = {
    -1, 1, 0,
    1, -1, 0,
    1, 1, 0
};

static const GLubyte colors[3 * 4] = {
    255, 0, 0, 255,
    0, 255, 0, 255,
    0, 0, 255, 255
};

v0 (-1,1)  v1 (1, -1)  v2 (1, 1)
Hello OpenGL ES, OpenGL ES part

```c
void init( )
{
    glColorClearColor ( 0.f, 0.f, 0.1f, 1.f );
    glMatrixMode ( GL_PROJECTION );
    glFrustumf          ( -1.f, 1.f, -1.f, 1.f, 3.f, 1000.f );
    glMatrixMode        ( GL_MODELVIEW );
    glShadeModel        ( GL_SMOOTH );
    glDisable ( GL_DEPTH_TEST );
    glVertexPointer     ( 3, GL_BYTE, 0, vertices );
    glColorPointer      ( 4, GL_UNSIGNED_BYTE, 0, colors );
    glEnableClientState ( GL_VERTEX_ARRAY );
    glEnableClientState ( GL_COLOR_ARRAY );
    glViewport ( 0, 0, GET_WIDTH(), GET_HEIGHT() );

    INIT_RENDER_CALLBACK(drawcallback);
}
```
Hello OpenGL ES, OpenGL ES part

```c
void drawcallback(void)
{
    glClear        ( GL_COLOR_BUFFER_BIT );
    glLoadIdentity ( );
    glTranslatef   ( 0.f, 0.f, -5.f );
    glDrawArrays   ( GL_TRIANGLES, 0, 3 );

    eglSwapBuffers(display, surface);
}
```
EGL config sorting

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>DEFAULT VALUE</th>
<th>SELECTION RULE</th>
<th>SORT PRIORITY</th>
<th>SORT ORDER</th>
</tr>
</thead>
<tbody>
<tr>
<td>EGL_BUFFER_SIZE [16]</td>
<td>0</td>
<td>AtLeast</td>
<td>3</td>
<td>Smaller</td>
</tr>
<tr>
<td>EGL_DEPTH_SIZE [15]</td>
<td>0</td>
<td>AtLeast</td>
<td>6</td>
<td>Smaller</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Selection rule: minimum requirement
Sort priority: which attrib is sorted first
Sort order: how attrib is sorted
One way of sorting
Not optimal for all applications
### Example of sorted list of configs

<table>
<thead>
<tr>
<th>EGL_CONFIG_ID</th>
<th>EGL_BUFFER_SIZE (Sort priority = 3)</th>
<th>EGL_DEPTH_SIZE (Sort priority = 6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>40</td>
<td>24</td>
<td>15</td>
</tr>
<tr>
<td>11</td>
<td>32</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>30</td>
<td>32</td>
<td>32</td>
</tr>
</tbody>
</table>

- Sorted first, smaller comes first
- Sorted next, smaller comes first
- Sorted last (if otherwise no unique order exists), smaller comes first
Example EGL config selection

```c
EGLConfig select_config(int type, int color_bits, int depth_bits, int stencil_bits)
{
    EGLBoolean err;
    EGLInt amount, attrib_list[5*2]; /* fits 5 attrs */
    EGLConfig best_config, configs[64]; /* max 64 configs considered */
    EGLInt *ptr;

    ptr = &attrib_list[0];

    /* Make sure that the config supports target surface type */
    *ptr++ = EGL_SURFACE_TYPE;
    *ptr++ = type;

    /* For color, we require minimum of <color_bits> bits */
    *ptr++ = EGL_BUFFER_SIZE;
    *ptr++ = color_bits;

    /* For depth, we require minimum of <depth_bits> bits */
    if(depth_bits)
    {
        *ptr++ = EGL_DEPTH_SIZE;
        *ptr++ = depth_bits;
    }
}```
if(stencil_bits)
{
    ptr[0] = EGL_STENCIL_SIZE;
    ptr[1] = stencil_bits;
    ptr[2] = EGL_NONE;
}
else
{
    ptr[0] = EGL_NONE;
}

err = eglChooseConfig(display, &attrib_list[0], &configs[0], 64, &amount);

if(amount == 0)
{
    /* If we didn't have get any configs, try without stencil */
    ptr[0] = EGL_NONE;
    err = eglChooseConfig(display, &attrib_list[0], &configs[0], 64, &amount);
}
if(amount > 0)
{
    /* We have either configs w/ or w/o stencil, not both. Find one with best AA */
    int i,best_samples;
    best_samples = 0;
    best_config = configs[0];

    for(i=0 ; i<amount ; i++)
    {
        int samp;
        eglGetConfigAttrib( display, configs[i], EGL_SAMPLES, &samp );
        if(samp > best_samples)
        {
            best_config = configs[i];
            best_samples = samp;
        }
    }
}
else best_config = (EGLConfig)0; /* no suitable configs found */

return best_config;
Texture matrix example

```c
void appinit_glass(void)
{
    GLint texture_handle;

    /* View parameters */
    glMatrixMode ( GL_PROJECTION );
    glFrustumf                 ( -1.f, 1.f, -1.f, 1.f, 3.f, 1000.f );
    glMatrixMode            ( GL_MODELVIEW );

    /* Reset state */
    glEnable                   ( GL_DEPTH_TEST );
    glClearColor              ( 0.f, 0.f, 0.1f, 1.f );

    /* Enable vertex arrays */
    glEnableClientState  ( GL_VERTEX_ARRAY );
    glEnableClientState ( GL_TEXTURE_COORD_ARRAY );
}```
/* Setup texture */

GLenum ( GL_TEXTURE_2D );

glGenTextures( 1, texture_handle );

GLuint ( GL_TEXTURE_2D, texture_handle );

GLenum ( GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0,
         GL_RGB, GL_UNSIGNED_BYTE, texture_data );

GLenum ( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
         GL_MODULATE );

GLenum ( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
         GL_LINEAR );

GLenum ( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
         GL_LINEAR );

GLenum ( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
         GL_CLAMP_TO_EDGE );

GLenum ( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
         GL_CLAMP_TO_EDGE );

}
Texture matrix example

```c
int render(float time)
{
    glClear ( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

    /* draw background with two textured triangles */
    glMatrixMode ( GL_TEXTURE );glLoadIdentity ( );
    glMatrixMode ( GL_PROJECTION);
    glLoadIdentity ( );
    glMatrixMode ( GL_MODELVIEW);
    glLoadIdentity ( );
    glColor4ub ( 255, 255, 255, 255 );
    glScalef ( 2.f, -2.f, 0.f );
    glTranslatef ( -0.5f, -0.5f, 0.f );
    glVertexPointer             ( 2, GL_BYTE, 0, back_coords );
    glTexCoordPointer ( 2, GL_BYTE, 0, back_coords );
    glDrawArrays ( GL_TRIANGLE_STRIP, 0, 4 );
```
Texture matrix example, coordinates

Texture "normals"

Vertex coordinates
Texture matrix example, coordinates

We just take the (x, y) of the texture coordinate output
Texture matrix example, coordinates
In this example we use the same data for vertex and texture "normals" as the object is cut away from roughly tessellated sphere (all coordinates unit length).

This is NOT possible for general objects. You should use separate normalized normals for other objects.
Texture matrix example

```c
void main()
{
    glMatrixMode ( GL_PROJECTION );
    glLoadIdentity ( );
    glFrustumf                ( -1.f, 1.f, -1.f, 1.f, 3.f, 1000.f );

    glMatrixMode ( GL_MODELVIEW );
    glLoadIdentity           ( );
    glTranslatef             ( 0, 0, -5.f );
    glRotatef ( time*25, 1.f, 1.f, 0.f ); /* (1) */
    glRotatef ( time*15, 1.f, 0.f, 1.f );

    glMatrixMode ( GL_TEXTURE );
    glLoadIdentity ( );
    glTranslatef ( 0.5f, 0.5f, 0.f );       /* [-0.5,0.5] -> [0,1] */
    glScalef ( 0.5f, -0.5f, 0.f );        /* [-1,1]  -> [-0.5,0.5] */
    glRotatef ( time*25, 1.f, 1.f, 0.f ); /* identical rotations! */
    glRotatef ( time*15, 1.f, 0.f, 1.f ); /* see (1) */
}```
Texture matrix example

/* use different color for the (glass) object vs. the background */
glColor4ub ( 255, 210, 240, 255 );
glVertexPointer ( 3, GL_FIXED, 0, vertices );
glTexCoordPointer ( 3, GL_FIXED, 0, vertices );
glDrawArrays ( GL_TRIANGLES, 0, 16*3 );
}
Texture matrix example
Fixed point programming

Why should you use it?
Most mobile handsets don’t have a FPU

Where does it make sense to use it?
Where it makes the most difference
For per-vertex processing: morphing, skinning, etc.
Per vertex data shouldn’t be floating point

OpenGL ES API supports 32-bit FP numbers
Fixed point programming

There are many variants of fixed point:

- Signed / Unsigned
- 2’s complement vs. Separate sign

OpenGL ES uses 2’s complement

Numbers in the range of $[-32768, 32768]$ [16 bits for decimal bits (precision of $1/65536$)]

All the examples here use 16.16 fixed point
Float to fixed and vice versa

Convert from floating point to fixed point

#define float_to_fixed(a)  (int)((a)*(1<<16)) or
#define float_to_fixed(a)  (int)((a)*(65536))

Convert from fixed point to floating point

#define fixed_to_float(a)  (((float)a)/(1<<16)) or
#define fixed_to_float(a)  (((float)a)/(65536))
Fixed point programming

Examples:

0x0001 0000 = 65536 = “1.0f”
0x0002 0000 = 2*65536 = “2.0f”
0x0010 0000 = 16*65536 = “16.0f”
0x0000 0001 = 1/65536 = “0.0000152587…”
0xffffffff ffff = -1/65536(-0x0000 0001)
Fixed point operations

Addition
#define add_fixed_fixed(a,b) ((a)+(b))

Multiply fixed point number with integer
#define mul_fixed_int(a,b) ((a)*(b))

MUL two FP numbers together
#define mul_fixed_fixed(a,b) \
(int)(((int64)a)*((int64)b)) >> 16)
Fixed point operations and scale

Addition:

\[ a & b = \text{original float values} \]
\[ S = \text{fixed point scale (e.g., 65536)} \]

\[ \text{result} = (a \times S) + (b \times S) = (a + b) \times S \]

Scaling term keeps the same
Range of the result is 33 bits
Possible overflow by 1 bit
Fixed point operations and scale

Multiplication:

\[ a \& b = \text{original float values} \]
\[ S = \text{fixed point scale (e.g., 65536)} \]

\[ \text{result} = (a \times S) \times (b \times S) = ((a \times b) \times S^2) \]
\[ \text{final} = ((a \times b) \times S^2) / S = (a \times b) \times S \]

Scaling term is squared \((S^2)\) and takes 32 bits
Also the integer part of the multiplication takes 32 bits

=> need 64 bits for full s16.16 * s16.16 multiply
Fixed point programming

Intermediate overflow
- Higher accuracy (64-bit)
- Downscale input
- Redo range analysis

Result overflow (48 bits)
- Redo range analysis
- Detect overflow, clamp

VALUE 1
32-bit

VALUE 2
32-bit

>> 16 = RESULT
48-bit
Fixed point programming

Division of integer by integer to a fixed point result

```c
#define div_int_int(a, b) \ 
   (int)(((int64)a)*(1<<16))/(b))

(a*S)/b = (a/b)*S
```

Division of fixed point by integer to a fixed point result

```c
#define div_fixed_int(a, b) ((a)/(b))
```

Division of fixed point by fixed point

```c
#define div_fixed_fixed(a, b) \ 
   (int)(((int64)a)*(1<<16))/(b))

(a*S*S)/(b*S) = (a/b)*S
```
Fixed point programming

Power of two MUL & DIV can be done with shifts
\[ a \times 65536 = a << 16, \quad a \div 256 = a >> 8 \]

Fixed point calculations overflow easily
Careful analysis of the range requirements is required

=>

Always add validation code to your fixed point code
#if defined(DEBUG)
int add_fix_fix_chk(int a, int b)
{
    int64 bigresult = ((int64)a) + ((int64)b);
    int smallresult = a + b;
    assert(smallresult == bigresult);
    return smallresult;
}
#endif

#if defined(DEBUG)
#  define add_fix_fix(a,b) add_fix_fix_chk(a,b)
#else
#  define add_fix_fix(a,b) ((a)+(b))
#endif
Fixed point math functions

Complex math functions
Pre-calculate for the range of interest

An example: Sin & Cos

Sin table between [0, 90°], fixed point angle (S = 2048)
Generate other angles and Cos from the table
Store in a short table (16-bit) as s0.16 (S = 32768)
Range for shorts is [-32768, 32767] ([-1.0, 1.0] for s0.16 FP)
Equals to [-1.0, +1.0] for s0.16 FP (+1.0 not included !)
Negative values stored in the table (can represent -1.0)
Example: Simple morphing (LERP)

Simple fixed point morphing loop (16-bit data, 16-bit coeff)

#define DOLERP_16(a,b,t) ((short)(((b)-(a))*(t))>>16)+(a))

void lerpgeometry(short *out, const short *inA, const short *inB, int count, int scale)
{
    int i;

    for(i=0; i<count; i++)
    {
        out[i*3+0] = DOLERP_16(inB[i*3+0], inA[i*3+0], scale);
        out[i*3+1] = DOLERP_16(inB[i*3+1], inA[i*3+1], scale);
        out[i*3+2] = DOLERP_16(inB[i*3+2], inA[i*3+2], scale);
    }
}
Converting existing code

OS/device conversions
  Programming model, C/C++, compiler, CPU

Windowing API conversion
  EGL API is mostly cross platform
  EGL Native types are platform specific

OpenGL -> OpenGL ES conversion
Example: Symbian porting

Programming model

- C++ with some changes (e.g., exceptions)
- Event based programming (MVC), no main / main loop
- Three level multitasking: Process, Thread, Active Objects

ARM CPU

- Unaligned memory accesses will cause exception (unlike x86)

OpenC (http://www.forum.nokia.com/openc)
Example: EGL porting

Native types are OS specific

- EGLNativeWindowType (RWindow *)
- EGLNativePixmapType (CFbsBitmap *)

Pbuffers are portable

Config selection

- Select the color depth to be same as in the display

Windowing system issues

- What if render window is clipped by a system dialog?
- Only full screen windows may be supported
OpenGL porting

• **glBegin/gEnd wrappers**
  - `_glBegin` stores the primitive type
  - `_glColor` changes the current per-vertex data
  - `_glVertex` stores the current data behind arrays and increments
  - `_glEnd` calls `glDrawArrays` with primitive type and length

```c
_glBegin(GL_TRIANGLES);
    _glColor4f(1.0,0.0,0.0,1.0);
    _glVertex3f(1.0,0.0,0.0);
    _glVertex3f(0.0,1.0,0.0);
    _glColor4f(0.0,1.0,0.0,1.0);
    _glVertex3f(0.0,0.0,1.0);
_glEnd();
```
OpenGL porting

Display list wrapper

- Add the display list functions as wrappers
- Add all relevant GL functions as wrappers
- When drawing a list, go through the collected list
void _glEnable( par1, par2 )
{
    if( GLOBAL()->iSubmittingDisplayList )
    {
        *(GLOBAL()->dlist)++ = DLIST_CMD_GLENABLE;
        *(GLOBAL()->dlist)++ = (GLuint)par1;
        *(GLOBAL()->dlist)++ = (GLuint)par2;
    }
    else
    {
        glEnable(par1,par2);
    }
}
OpenGL porting

Vertex arrays

- OpenGL ES supports only vertex arrays
- SW implementations get penalty from float data
- Use as small types as possible (byte, short)
- For HW it shouldn’t make a difference, mem BW

- With OpenGL ES 1.1 always use VBOs
OpenGL porting

No quads

Convert a quad into 2 triangles

No real two-sided materials in lighting

If you really need it, submit front and back triangles

OpenGL ES and querying state

OpenGL ES 1.0 only supports static getters

OpenGL ES 1.1 supports dynamic getters

For OpenGL ES 1.0, create own state tracking if needed
Demo

Sequel to game One (Nokia)
Questions?
Python: Great for rapid prototyping

Python

designed to be as small, practical, and open as possible

easy and fun OO programming

sourceforge.net/projects/pyS60

Python 2.2.2 on Symbian S60

wrappers for phone SDK libraries

can extend in Symbian C++
Python bindings to OpenGL ES

Almost direct bindings

OpenGL ES functions that take in pointers typically take in a Python list

Next we’ll show a full S60 GUI program with OpenGL ES
import appuifw  # S60 ui framework
import sys

from glcanvas import *
from gles import *
from key_codes import *
class Hello:

    vertices = array( GL_BYTE, 3,
                      [-1, 1, 0,
                       1,-1, 0,
                       1, 1, 0] )

    colors = array( GL_UNSIGNED_BYTE, 4,
                    [255, 0, 0, 255,
                     0, 255, 0, 255,
                     0, 0, 255, 255] )
def __init__(self):                 # class constructor
    self.exiting = False          # while !exit, run
    self.frame, self.angle = 0, 0  # set variables
    self.old_body = appuifw.app.body
    try:                          # create surface
        c = GLCanvas( redraw_callback = self.redraw,
                        resize_callback = self.resize )
        appuifw.app.body = c
        self.canvas = c
    except Exception, e:
        appuifw.note( u"Exception: %s" % (e) )
        self.start_exit()
    return

    appuifw.app.menu = [(u"Exit", self.start_exit)]
    c.bind( EKeyLeftArrow, lambda:self.left() )
    c.bind( EKeyRightArrow, lambda:self.right() )
    self.initgl()
Keyboard and resize callbacks

def left(self):
    self.angle -= 10

def right(self):
    self.angle += 10

def resize(self):
    if self.canvas:
        glViewport( 0, 0,
                    self.canvas.size[0],
                    self.canvas.size[1] )
def start_exit(self):
    self.exiting = True

def run(self):
    app = appuifw.app
    app.exit_key_handler = self.start_exit_exit
    while not self.exiting:
        self.canvas.drawNow()
        e32.ao_sleep(0.01)
    app.body = self.old_body
    self.canvas = None
    app.exit_key_handler = None
def initgl(self):
    glMatrixMode( GL_PROJECTION )
    glFrustumf  ( -1.0, 1.0, -1.0, 1.0, 3.0, 1000.0 )
    glMatrixMode( GL_MODELVIEW )
    glEnable   ( GL_DEPTH_TEST )
    glShadeModel( GL_SMOOTH )
    glClearColor( 0.0, 0.0, 0.1, 1.0 )
    glVertexPointerb( self.vertices )
    glColorPointerub( self.colors )
    glEnableClientState( GL_VERTEX_ARRAY )
    glEnableClientState( GL_COLOR_ARRAY )
def redraw(self, frame=None):
    if self.canvas:
        glClear( GL_COLOR_BUFFER_BIT )
        glLoadIdentity()
        glTranslatef( 0.0, 0.0, -5.0 )
        glRotatef   ( self.angle,
                       0.0, 0.0, 1.0 )
        glRotatef   ( self.frame,
                       0.0, 1.0, 0.0 )
        glDrawArrays( GL_TRIANGLES, 0,3 )
    self.frame += 1
Using the class

```python
appuifw.app.screen = 'full'
try:
    app = Hello()
except Exception, e:
    appuifw.note(u"Cannot start: %s" % e)
else:
    app.run()

del app
```
OpenGL ES Performance Considerations

Ville Miettinen
Targeting the "mobile platform"

CPU speed and available memory varies

- Current range ~30Mhz - 600+ MHz, ARM7 to ARM11
- From no FPUs to SIMD FPUs

Different resolutions

- QCIF (176x144) to VGA (640x480) and beyond, antialiasing on higher-end devices
- Color depths 4-8 bits per channel (12-32 bpp)

Portability issues

- Different CPUs, OSes, Java VMs, C compilers, ...
- OpenKODE from the Khronos Group will help to some extent
Graphics capabilities

General-purpose multimedia hardware
  Pure software renderers (all done using CPU & integer ALU)
  Software + DSP / WMMX / FPU / VFPU
  Multimedia accelerators

Dedicated 3D hardware
  Software T&L + HW tri setup / rasterization
  Full hardware acceleration

Performance: 50K – 2M tris, 1M – 100M pixels / sec
Next gen: 30M+ tris, 1000M pixels / sec
Standards help somewhat

Act as hardware abstraction layers
- Provide programming interface (API)
- Same feature set for different devices
- Unified rendering model

Performance cannot be guaranteed
Scalability

Successful application has to run on hundreds of different phone models

No single platform popular enough

Same gameplay but can scale video and audio

Design for lowest-end, add eye candy for high-end

Scalability has to be built into the design
3D content is easy to scale

Separate low and high poly count 3D models
Different texture resolutions & compressed formats
Rendering quality can be scaled
  Texture filtering, perspective correction, blend functions, multi-texturing, antialiasing
Special effects

Identify special effects

- Bullet holes, skid marks, clouds, ...
- Cannot have impact on game play
  - Fog both gameplay and visual element
  - Multiplayer games have to be fair

Users can alter performance by controlling effects
Tuning down the details

Particle systems

- Number of particles, complexity, visuals
- Shared rendering budget for all particle systems

Background elements

- Collapse into sky cubes, impostors

Detail objects

- Models to have “important” and “detail” parts
Profiling

Performance differences often system integration issues - not HW issues

Measuring is the only effective way to find out how changes in code affect performance

Profile on actual target device if possible

Public benchmark apps provide some idea of graphics performance

gDEBugger ES for gfx driver profiling
Identifying bottlenecks

Three groups: application code, vertex pipeline, pixel pipeline

- Further partitioned into pipeline stages
- Overall pipeline runs as fast as its slowest stage

Locate bottlenecks by going through each stage and reducing its workload

- If performance changes, you have a bottleneck

Apps typically have multiple bottlenecks
Pixel pipeline bottlenecks

Find out by changing rendering resolution
   If performance increases, you have a bottleneck
   Either texturing or frame buffer accesses

Remedies
   Smaller screen resolution, render fewer objects,
   use simpler data formats, smaller texture maps,
   less complex fragment and texture processing
Vertex pipeline bottlenecks

Vertex processing or submission bottlenecks

Find out by rendering every other triangle but using same vertex arrays

Remedies

Submission: smaller data formats, cache-friendly organization, fewer triangles

Vertex processing: simpler T&L (fewer light sources, avoid dynamic lighting and fog, avoid floating-point data formats)
Application code bottlenecks

Two ways to find out
- Turn off all application logic
- Turn off all rendering calls

Floating-point code #1 culprit

Use profiler
- HW profilers on real devices costly and hard to get
- Carbide IDE from Nokia (S60 and UIQ Symbian)
- Lauterbach boards
- Desktop profiling (indicative only)
Changing and querying the state

- Rendering pipes are one-way streets
- Apps should know their own state
  - Avoid dynamic getters if possible!
- Perform state changes in a group at the beginning of a frame
- Avoid API synchronization
  - Do not mix 2D and 3D libraries!
”Shaders”

Combine state changes into blocks (”shaders”)

Minimize number of shaders per frame

Typical application needs only 3-10 ”pixel shaders”

Different 3-10 shaders in every application

Enforce this in artists’ tool chain

Sort objects by shaders every frame

Split objects based on shaders
Complexity of shaders

Software rendering: everything costs!

- Important to keep shaders as simple as possible
  Even if introduces additional state changes
  Example: turn off fog & depth buffering when rendering overlays

Hardware rendering: Usually more important to keep number of changes small
Model data

- Keep vertex and triangle data short and simple!
  - Better cache coherence, less memory used
- Make as few rendering calls as possible
  - Combine strips with degenerate triangles
- Weld vertices using off-line tool
- Order triangle data coherently
- Use hardware-friendly data layouts
  - Buffer objects allow storing data on server
Transformation pipeline

Minimize matrix changes
  Changing a matrix may involve many hidden costs
  Combine simple objects with same transformation
  Flatten and cache transformation hierarchies

ES 1.1: Skinning using matrix palettes
  CPU doesn’t have to touch vertex data

ES 1.1: Point sprites for particle effects
Rendering pipeline

Rendering order is important
  Front-to-back improves depth buffering efficiency
  Also need to minimize number of state changes!

Use culling to speed up rendering pipeline
  Conservative: frustum culling & occlusion culling
    Portals and Potentially Visible Sets good for mobile
  Aggressive culling
    Bring back clipping plane in, drop detail & small objects
Lighting

Fixed-function lighting pipelines are so 1990s

Drivers implemented badly even in desktop space
In practice only single directional light fast
OpenGL’s attenuation model difficult to use
Spot cutoff and specular model cause aliasing
No secondary specular color
Flat shading sucks
Artifacts unless geometry heavily tessellated
Lighting (if you have to use it)

- Single directional light usually accelerated
- Pre-normalize vertex normals
- Avoid homogeneous vertex positions
- Turn off specular illumination
- Avoid distance attenuation
- Turn off distant non-contributing lights
Lighting: the fast way

While we’re waiting for OpenGL ES 2.0 drivers

- Pre-computed vertex illumination good if slow T&L
- Illumination using texturing
  - Light mapping
  - ES 1.1: dot3 bump mapping + texture combine
  - Less tessellation required
- Combining with dynamic lighting: color material tracking
Environment mapping
Textures

Mipmaps always a Good Thing™

- Improved cache coherence and visual quality
- ES 1.1 supports auto mipmap generation

Avoid modifying texture data

Keep textures "right size", use compressed textures

Different strategies for texture filtering & perspective correction

- SW implementations affected
Textures (cont’d)

Multitexturing

Always faster than doing multiple rendering passes
ES 1.1: support at least two texturing units
ES 1.1: TexEnvCombine neat toy

Use small & compressed texture formats

Texture atlases: combining multiple textures
    Reduces texture state changes
ES 2.0
Overview

Robert J. Simpson
AMD
Contents

GLSL Overview
Example Application
  Physics of reflections
  Creating the skybox
  Simulating water
Detailed walk-through
  Initializing EGL
  Compiling and linking shaders
  Setup: attributes, textures, uniforms, attribute buffers
  Drawing the frame
OpenGL ES Shading Language
  Differences versus desktop
  Embedded architectures
  Relative cost of operations
  Special features: Precision & Invariance
  Some tips for programming with ES
ES 2.0 Pipeline

Robert J. Simpson
AMD
Open GL Fixed Function pipeline

API

Primitive Processing → Transform and Lighting → Primitive Assembly → Rasterizer

Vertices

Triangles/Lines/Points

Texture Environment → Colour Sum → Fog

Alpha Test → Depth Stencil → Colour Buffer Blend → Dither

Frame Buffer
Open GL Programmable pipeline

API

Primitive Processing

Vertices

Vertex Shader

Primitive Assembly

Rasterizer

Triangles/Lines/Points

Fragment Shader

Depth Stencil

Colour Buffer Blend

Dither

Frame Buffer
Programmer’s model

- Attributes (8 * vec4)
  - Vertex Uniforms (128 * vec4) → Vertex Shader
  - Primitive Assembly & Rasterize
  - Varyings (8 * vec4)
  - Fragment Uniforms (16 * vec4) → Fragment Shader
  - Per-Sample Operations
Vertex Shader

Attribute 0
Attribute 1
Attribute 2
Attribute 3
Attribute 4
Attribute 5
Attribute 6
Attribute 7

Uniforms

Textures

Varying 0
Varying 1
Varying 2
Varying 3
Varying 4
Varying 5
Varying 6
Varying 7

Temporary variables

gl_Position
gl_PointSize
Fragment Shader

- **Uniforms**
- **Textures**

- **Varying**
  - Varying 0
  - Varying 1
  - Varying 2
  - Varying 3
  - Varying 4
  - Varying 5
  - Varying 6
  - Varying 7

- **Temporary variables**
  - gl_FragCoord
  - gl_FrontFace
  - gl_PointPosition
The Vertex Shader

The vertex shader can do:

- Transformation of position using model-view and projection matrices
- Transformation of normals, including renormalization
- Texture coordinate generation and transformation
- Per-vertex lighting
- Calculation of values for lighting per pixel
The vertex shader cannot do:

- Anything that requires information from more than one vertex
- Anything that depends on connectivity.
- Any triangle operations (e.g. clipping, culling)
- Access colour buffer
The Fragment Shader

The fragment shader can do:

- Texture blending
- Fog
- Alpha testing
- Dependent textures
- Pixel discard
- Bump and environment mapping
The Fragment Shader

The fragment shader cannot do:

- Blending with colour buffer
- ROP operations
- Depth or stencil tests
- Write depth
GLSL ES

Robert J. Simpson
AMD
GLSL ES Overview

Based on GLSL as used in OpenGL 2.0
  Open standard
Pure programmable model
  Most fixed functionality removed.
Not 100% backward compatible with ES1.x
  Embedded systems do not have the legacy requirements of the desktop
No Software Fallback
  Implementations (usually) hardware or nothing
  Running graphics routines in software doesn’t make sense on embedded platforms
Optimized for use in Embedded devices
  Aim is to reduce silicon cost
  Reduced shader program sizes
  Reduced register usage
  Reduced numeric precision
GLSL ES Overview

‘C’ – like language
Many simplifications
   No pointers
   Strongly typed. No implicit type conversion
   Simplified preprocessor
Some graphics-specific additions
   Built-in vector and matrix types
   Built-in functions
   Support for mixed precisions
   Invariance mechanism.
Differences from Desktop OpenGL
   Restrictions on shader complexity
   Fewer sampler modes
GLSL ES Overview

Comments
//
/*   */

Control
#if
#endif
#else
#error

Operators
defined

Macros
#
#define
#undef

Extensions
#pragma
#extension

Misc
#version
#line
# GLSL ES Overview

## Scalar
- `void`
- `float`
- `int`
- `bool`

## Vector
- **Boolean:**
  - `bvec2`
  - `bvec3`
  - `bvec4`
- **Integer:**
  - `ivec2`
  - `ivec3`
  - `ivec4`
- **Floating Point:**
  - `vec2`
  - `vec3`
  - `vec4`

## Matrix
- **Floating Point:**
  - `mat2`
  - `mat3`
  - `mat4`

## Sampler
- `sampler2D`

## Containers
- **Structures:**
  - `struct`
- **Arrays:**
  - `[]`
GLSL ES Storage Qualifiers

**const**
Local constants within a shader.

**uniform**
‘Constant shader parameters’ (light position/direction, texture units, …)
Do not change per vertex.

**attribute**
Per-vertex values (position, normal,…)

**varying**
Generated by vertex shader
Interpolated by the rasterizer to generate per pixel values
Used as inputs to Fragment Shader
e.g. texture coordinates
Function Parameter Qualifiers

Used to pass values in or out or both e.g.

```c
bool f(in vec2 in_v, out float ret_v)
{
    ...
}
```

Qualifiers:

- **in** Input parameter. Variable can be modified.
- **const in** Input parameter. Variable cannot be modified.
- **out** Output parameter.
- **inout** Input and output parameter.

Functions can still return a value
But need to use a parameter if returning an array.
Function Parameter Qualifiers

Call by value ‘copy in, copy out’ semantics.

Not quite the same as C++ references:

```c
bool f(inout float a, b)
{
    a++;
    b++;
}

void g()
{
    float x = 0.0;
    f(x,x);            // x = 1.0 not 2.0
}
```
GLSL ES Overview

Order of copy back is undefined

```c
bool f(inout float a, b)
{
    a = 1.0;
    b = 2.0;
}

void g()
{
    float x ;
    f(x,x);    // x = 1.0 or 2.0
    // (undefined)
}
```
Precision Qualifiers

**lowp float**
- Effectively sign + 1.8 fixed point.
- Range is $-2.0 < x < 2.0$
- Resolution $1/256$
- Use for simple colour blending

**mediump float**
- Typically implemented by sign + 5.10 floating point
- $-16384 < x < 16384$
- Resolution 1 part in 1024
- Use for HDR blending.
Precision Qualifiers

**highp float**
- Typically implemented by 24 bit float (16 bit mantissa)
- range $\pm 2^{62}$
- Resolution 1 part in $2^{16}$
- Use of texture coordinate calculation
  - e.g. environment mapping

**single precision (float32)**
- Not explicit in GLSL ES but usually available in the vertex shader (refer to device documentation)
Precision Qualifiers

Precision depends on the operands:

```c
lowp float x;
mediump float y;
highp float z = x * y;
```
(evaluated at medium precision)

Literals do not have any defined precision

```c
lowp float x;
highp float z = x * 2.0 + 1.2;
```
(evaluated at low precision)
Constructors

Replaces type casting
No implicit conversion: must use constructors
All named types have constructors available
  Includes built-in types, structures
  Excludes arrays

Integer to Float:
```c
int n = 1;
float x, y;
x = float(n);
y = float(2);
```
Constructors

Concatenation:

```c
float x = 1.0, y = 2.0;
vec2 v = vec2(x, y);
```

Structure initialization

```c
struct S {int a; float b;};
struct S s = S(2, 3.5);
```
Swizzle operators

Use to select a set of components from a vector
Can be used in L-values

```cpp
vec2 u, v;
v.x = 2.0; // Assignment to single component
float a = v.x; // Component selection
v.xy = u.yx; // swap components
v = v.xx; // replicate components
v.xx = u; // Error
```

Component sets: Use one of

xyzw OR rgba OR stpq
Indexing operator

vec4 u, v;
float x = u[0]; // equivalent to u.x

Must use indexing operator for matrices

mat4 m
vec4 v = m[0];
m.x; // error
GLSL ES Overview

Operators

++  --  +  -  !  ()  []
*   /   +   -
<   <=  >  >=
==  !=
&&  ^^  ||
?:
=   *=  /=  +=  -=

Flow control

x == y ? a : b
if else
for while do
return break continue
discard (fragment shader only)
Built-in Variables

Aim of ES is to reduce the amount of fixed functionality
Ideal would be a totally pure programmable model
But still need some

Vertex shader

```cpp
vec4 gl_Position;    // Write-only
float gl_PointSize;  // Write-only
```

Fragment shader

```cpp
vec4 gl_FragCoord;   // Read-only
bool gl_FrontFacing; // Read-only
vec2 gl_PointCoord;  // Read-only
float gl_FragColor;  // Write only
```
Built-in Functions

General

- pow, exp2, log2, sqrt, inversesqrt
- abs, sign, floor, ceil, fract, mod, min, max, clamp

Trig functions

- radians, degrees, sin, cos, tan, asin, acos, atan

Geometric

- length, distance, cross, dot, normalize, faceForward, reflect, refract
GLSL ES Overview

Interpolations

- **mix(x, y, alpha)**
  
  \[ x \times (1.0 - \alpha) + y \times \alpha \]

- **step(edge, x)**
  
  \[ x \leq edge \, ? \, 0.0 \, : \, 1.0 \]

- **smoothstep(edge0, edge1, x)**
  
  \[ t = \frac{x - edge0}{edge1 - edge0}; \]
  \[ t = \text{clamp}(t, 0.0, 1.0); \]
  \[ \text{return } t \times t \times (3.0 - 2.0 \times t); \]

Texture

- **texture1D, texture2D, texture3D, textureCube**
- **texture1DProj, texture2DProj, textureCubeProj**
GLSL ES Overview

Vector comparison ($\text{vecn}$, $\text{ivec}$)
- $\text{vecn}$ lessThan($\text{vecn}$, $\text{vecn}$)
- $\text{vecn}$ lessThanEqual($\text{vecn}$, $\text{vecn}$)
- $\text{vecn}$ greaterThan($\text{vecn}$, $\text{vecn}$)
- $\text{vecn}$ greaterThanEqual($\text{vecn}$, $\text{vecn}$)

Vector comparison ($\text{vecn}$, $\text{ivec}$, $\text{bvec}$)
- $\text{bvec}$ equal($\text{vecn}$, $\text{vecn}$)
- $\text{bvec}$ notEqual($\text{vecn}$, $\text{vecn}$)

Vector ($\text{bvec}$)
- $\text{bvec}$ any($\text{bvec}$)
- $\text{bvec}$ all($\text{bvec}$)
- $\text{bvec}$ not($\text{bvec}$)

Matrix
- matrixCompMult ($\text{matn}$, $\text{matn}$)
Invariance

Robert J. Simpson
AMD
Invariance

Definition:

“An invariant operation is an operation that, given the same set of inputs, always produces the same result.”

So why might this not be true?
Invariance: The Problem

Causes of variance:

Mathematical operations are not precisely defined.
- No IEEE arithmetic
- User has limited control over the driver/compiler
  Compilers ‘cheat’ a bit to get better performance e.g.
  \[ a + b + c + d \rightarrow (a+b) + (c+d) \]
  Mathematically correct but in floating point can give different a result

Consequence:

Same code may produce (slightly) different results
Invariance

Why do you care?

Multi-pass

Any algorithm relying on repeatable calculations

Any algorithm relying on a value remaining constant
Invariance

Consider a simple transform in the vertex shader:

\[
\begin{pmatrix}
  x' \\
  y' \\
  z' \\
  w'
\end{pmatrix} =
\begin{pmatrix}
  a & b & c & d \\
  e & f & g & h \\
  i & j & k & l \\
  m & n & o & p
\end{pmatrix}\begin{pmatrix}
  x \\
  y \\
  z \\
  w
\end{pmatrix}
\]

\[x' = ax + by + cz + dw\]

But how is this calculated in practice?

There may be several possible code sequences
Invariance

e.g.

```
MUL R1, a, x
MUL R2, b, y
MUL R3, c, z
MUL R4, d, w
ADD R1, R1, R2
ADD R3, R3, R4
ADD R1, R1, R3
```

or

```
MUL R1, a, x
MADD R1, b, y
MADD R1, c, z
MADD R1, d, w
```
Invariance

Three reasons the result may differ:

Use of different instructions
Instructions executed in a different order
Different precisions used for intermediate results (only minimum precisions are defined)

But it gets worse...
Invariance

Modern compilers may rearrange your code
Values may lose precision when written to a register
Sometimes cheaper to recalculate a value
But will it be calculated the same way?

```cpp
const vec2 pos = a + b * c;  // Done once
// or twice?

vec4 col1 = texture2D(tex1, pos);
...
vec4 col2 = texture2D(tex2, pos);  // does pos have
// the same value
// as before?

gl_FragColor = col1 - col2;  //
```
Invariance

Solution is in two parts:

1. invariant keyword specifies that specific variables are invariant e.g.

   invariant varying vec3 LightPosition;

2. Currently can only be used on certain variables

3. Global switch to make all variables invariant

   #pragma STDGL invariant(all)
Invariance

Invariance flag controls:
- Invariance within shaders
- Invariance between shaders.

Usage
- Turn on invariance to make programs ‘safe’ and easier to debug
- Turn off invariance to get the maximum optimization from the compiler.
ES2.0 Example:
The Application Framework

Robert J. Simpson
AMD
Writing an application – Basic Steps

Set up EGL

Setup shader, pipeline state

Create vertex buffers, textures

Main loop
  Update state (transforms etc.)
  Bind
  Draw
Writing an App – Initialization

**EGL**
- Get EGL display
- EGL Initialization
- Choose EGL config options using an attribute list
- Create window surface
- Create EGL context and attach to surface

**GL ES**
- Compile and Link shaders
- Create and bind Textures
- Bind (or get) attributes
- Set up uniforms
- Create Vertex Buffers
- Map buffer data
EGLDisplay egl_display =
eglGetDisplay(EGL_DEFAULT_DISPLAY);

int ok = eglInitialize(egl_display,
            &majorVersion,
            &minorVersion)
EGL Initialization

Set up attributes for EGL context

```c
EGLint attr[MAX_EGL_ATTRIBUTES];

attr[nAttrib++] = EGL_RED_SIZE;
attrib[nAttrib++] = 5;
...
attrib[nAttrib++] = EGL_DEPTH_SIZE;
attrib[nAttrib++] = 16;
attrib[nAttrib++] = EGL_STENCIL_SIZE;
attrib[nAttrib++] = 0;
...
```
EGL Initialization (cont)

eglChooseConfig(egl_display, 
    attrib_list, 
    &egl_config, // returned configs 
    1, // max no. of configs 
    &num_configs)

eglCreateWindowSurface(egl_display, 
    egl_config, 
    NativeWindowType (hWnd), 
    NULL)
EGL Initialization: Creating a context

custom = eglCreateContext(egl_display, 
    egl_config, 
    EGL_NO_CONTEXT, 
    NULL);

eglMakeCurrent(egl_display, 
    egl_surface, // for draw 
    egl_surface, // for read 
    egl_context);
Compiling and using shaders

- glCreateShaderObject
- glShaderSource
- glCompileShader
- glAttachObject
- glLinkProgram
- glUseProgramObject
- glDeleteObject

Vertex Shader

Fragment Shader
Compiling and Linking Shaders

Create objects

```cpp
program_handle = glCreateProgram();

// Create one shader of object of each type.

GLuint vertex_shader_handle
    = glCreateShader (GL_VERTEX_SHADER);

GLuint fragment_shader_handle
    = glCreateShader (GL_FRAGMENT_SHADER);
```
Compiling Shaders

Compile vertex shader (and fragment shader)

```c
char* vert_source = ... 
const char* vert_gls[1] = {vert_source};

glShaderSource(vertex_shader_handle, 
  1,                                           // no. of strings 
  vert_gls, 
  NULL );

glCompileShader(vertex_shader_handle);

GLint vertCompilationResult = 0;

glGetShaderiv(vertex_shader_handle, 
  GL_COMPILE_STATUS, 
  &vertCompilationResult);
```
Linking Shaders

Attach shaders to program object and link

```c
glAttachShader(program_handle,
               vertex_shader_handle);

glAttachShader(program_handle,
               fragment_shader_handle);

glLinkProgram (program_handle);
```

Note that many compilers will only report errors at link time.
Setting up Attributes

Can bind attributes before linking e.g.

```c
glBindAttribLocation (prog_handle, 0, "pos");
```

Or get attribute location after linking:

```c
GLint p;

p = glGetAttribLocation (prog_handle, "pos");
```

Can do a combination.
Setting up Textures

Texture samplers are *Uniforms* in GLSL ES

First Generate ID and specify type (cube map)

```c
uint32 Id;

glGenTextures(1, &Id);

glActiveTexture (GL_TEXTURE0);

glBindTexture(GL_TEXTURE_CUBE_MAP, Id);
```
Setting up Textures (cont)

```c
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, image[0].pixels);

glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X, ...

glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y, ...

glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, ...

glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Z, ...

glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Z, ...
```
Setting up Uniforms

Must do this after `glUseProgram`:

```c
glUseProgram(prog_handle);
```

Use `glGetUniformLocation` e.g.

```c
GLint loc_sky_box =
    glGetUniformLocation (prog_handle,"skyBox");
```

Can then set value e.g.

```c
GLint texture_unit = 0;
    glUniform1i (loc_sky_box,texture_unit);
```
Setting up Attribute Buffers

Create buffer names

```c
GLuint bufs[1];
glGenBuffers (1, bufs);
```

Create and initialize buffer

```c
glBindBuffer (GL_ARRAY_BUFFER, bufs[0]);

glBufferData (GL_ARRAY_BUFFER,
    size_bytes, p_data, GL_STATIC_DRAW);
```
Specify format:

```c
glBindBuffer(GL_ARRAY_BUFFER, bufs[0]);

glVertexAttribPointer(0,  // index
        4,       // size
        GL_FLOAT, // type
        GL_FALSE, // normalize?
        0,       // (stride)
        NULL );   // (attribs)
```
Drawing the frame

- Clear frame buffer
- Set render state
- Enable array
- DrawArray
Enable array and Draw

```c
glEnableVertexAttribArray( 0 );

glBindBuffer (GL_ARRAY_BUFFER,0);

glDrawArrays (GL_TRIANGLE_STRIP,0, n_vertices);
```
ES2.0 Example
- The shaders
Example: Water demo
Skybox

Geometry is a sphere
Use position to access a cube map
Cube Map
Skybox

Access cube map using the normals of the vertices
Skybox

But...

Normal = Position

So no need to generate and store separate normals
uniform mat4 view_proj_matrix;
uniform vec4 view_position;
attribute vec4 rm_Vertex;
varying vec3 vTexCoord;

void main(void)
{
    vec4 newPos = vec4(1.0);
    newPos.xyz = rm_Vertex.xyz + view_position.xyz;
    gl_Position = view_proj_matrix * vec4(newPos.xyz,1.0);
    vTexCoord = rm_Vertex.xyz;
}
Sky box: Fragment Shader

```cpp
precision highp float;
uniform samplerCube skyBox;
varying vec3 vTexCoord;
void main(void)
{
    gl_FragColor = textureCube(skyBox,vTexCoord);
}
```
Water: Reflection Mapping

- Perturbed normal
- Original normal
- Actual geometry
- Geometry we are trying to emulate
Approximating Fresnel Reflection

Greater angle of incidence = more reflection

Smaller angle of incidence = less reflection
Fresnel Reflection (cont.)

- Perturbed normal, high angle of incidence: less reflection
- Unperturbed normal, low angle of incidence: more reflection
- View vector
Water

Geometry is a simple grid
Uses the same cubemap as the skybox
Water Ripples

Use noise texture for bump map.
Exact texture not important
   Try experimenting
uniform vec4 view_position;
uniform vec4 scale;
uniform mat4 view_proj_matrix;
attribute vec4 rm_Vertex;
attribute vec3 rm_Normal;
varying vec2 vTexCoord;
varying vec3 vNormal;
varying vec3 view_vec;
void main(void) {
    vec4 Position = rm_Vertex.xyzw;
    Position.xz *= 1000.0;
    vTexCoord = Position.xz * scale.xz;
    view_vec = Position.xyz - view_position.xyz;
    vNormal = rm_Normal;
    gl_Position = view_proj_matrix * Position;
}
Water: Fragment Shader

uniform sampler2D noise;
uniform samplerCube skyBox;
uniform float time_0_X;
uniform vec4 waterColor;
uniform float fadeExp;
uniform float fadeBias;
uniform vec4 scale;
uniform float waveSpeed;
varying vec2 vTexCoord;
varying vec3 vNormal;
varying vec3 vViewVec;
void main(void)
{
    vec2 tcoord = vTexCoord;
    tcoord.x += waveSpeed * time_0_X;
    vec4 noisy = texture2D(noise, tcoord.xy);
    // Signed noise
    vec3 bump = 2.0 * noisy.xyz - 1.0;
    bump.xy *= 0.15;

    // Make sure the normal always points upwards
    bump.z = 0.8 * abs(bump.z) + 0.2;
Water Fragment Shader (cont)

```cpp
// Offset the surface normal with the bump
bump = normalize(vNormal + bump);

// Find the reflection vector
vec3 reflVec = reflect(vViewVec, bump);
vec4 refl = textureCube(skyBox, reflVec.yzx);
```
float lrp = 1.0 - dot(-normalize(vViewVec), bump);

// Interpolate between the water color and
// reflection
float blend = fadeBias + pow(lrp, fadeExp);
blend = clamp(blend, 0.0, 1.0);
gl_FragColor = mix(waterColor, refl, blend);
Programming Tips
Check for errors regularly

Use e.g.

```c
assert(!glError());
```

But remember `glError()` gets the last error:

```c
... // error here
Glint error = glError();
...
assert(!glError()); // No error
```
The coordinate system

Coordinate system is:

Right handed before projection
Increasing z is towards the viewer.

Left handed after projection
Increasing z is away from the viewer.
Matrix Convention

Matrices are column-major
  column index varies more slowly
Vectors are columns
But this is purely convention
Only the position in memory is important
  Translation specified in elements 12,13,14
You need to provide a projection matrix e.g.

\[
\begin{bmatrix}
\frac{2.0 \times \text{near}}{\text{right} - \text{left}} & 0.0 & \frac{\text{right} + \text{left}}{\text{right} - \text{left}} & 0.0 \\
0.0 & \frac{2.0 \times \text{near}}{\text{top} - \text{bottom}} & \frac{\text{top} + \text{bottom}}{\text{top} - \text{bottom}} & 0.0 \\
0.0 & 0.0 & -(\frac{\text{far} + \text{near}}{\text{far} - \text{near}}) & -2.0 \times \frac{\text{far} \times \text{near}}{\text{far} - \text{near}} \\
0.0 & 0.0 & -1.0 & 0.0 \\
\end{bmatrix}
\]

near and far are both positive
Performance Tips

Keep fragment shaders simple
- Fragment shader hardware is expensive.
- Early implementations will not have good performance with complex shaders.

Try to avoid using textures for function lookups.
- Calculation is quite cheap, accessing textures is expensive.
- This is more important with embedded devices.
Performance Tips (cont)

Minimize register usage
  Embedded devices do not support the same number of registers compared with desktop devices. Spilling registers to memory is expensive.

Minimize the number of shader changes
  Shaders contain a lot of state
  May require the pipeline to be flushed
  Use uniforms to change behaviour in preference to loading a new shader.
ES vs. Desktop
GLSL ES vs. GLSL desktop

Based on GLSL as used in OpenGL 2.0
  Open standard
Pure programmable model
  Most fixed functionality removed.
Not 100% backward compatible with ES1.x
  Embedded systems do not have the legacy requirements of the desktop
No Software Fallback
  Implementations (usually) hardware or nothing
  Running graphics routines in software doesn’t make sense on embedded platforms
Optimized for use in Embedded devices
  Aim is to reduce silicon cost
  Reduced shader program sizes
  Reduced register usage
  Reduced numeric precision
Mobile Architecture

- CPU, graphics on the same chip.
- Unified memory architecture
- Cache for CPU
- Only limited cache for graphics device
- No internal framebuffer
- No internal z buffer
- Limited texture cache
Mobile vs. PC Architecture

Mobile

- SOC
  - CPU
  - GPU
  - BUS
  - Memory

PC

- CPU
- GPU
- Chipset
- Memory

Memory
Mobile Architecture – What this means

Limited memory bandwidth
- Hardware number crunching is relatively cheap
- Moving data around is the main problem
- Frame buffer access (or geometry buffer access for tiled architectures) is a heavy user
- Texture bandwidth is an issue

CPU cannot perform floating point
- Rather different from the PC world
- Means more rapid move to hardware vertex shaders

Any advantages?
- Lower resolution means less data to move around
- Easier to access frame/depth buffer because of unified memory
Performance Tips
Tips for Improving Performance

Golden rules:

- Don’t do things you don’t need to do
- Let the hardware do as much as possible
- Don’t interrupt the pipeline
- Minimize data transfer to and from the GPU
Don’t do what you don’t need to do

What frame rate do you actually need?

- Achieving 100fps on a PC may be a good thing
- LCD displays have slower response
- Eats power.

Not using depth testing? Turn off the z buffer

- Saves at least one read per pixel

Not using alpha blending? Turn it off.

- Saves reading the colour buffer.

Not using textures? Don’t send texture coordinates to the GPU

- Reduces geometry bandwidth
Let the hardware do it

Hardware is very good at doing computation

Transform and lighting is fast

Will become even more useful with vertex shaders

Texture blending is ‘free’. Don’t use multi-pass.

Anti-aliasing is very cheap.

Hardware is good when streaming data

But works best when data is aligned with memory

Most efficient when large blocks are transferred

BUT: Memory bandwidth is still in short supply
Minimize Data Transfer

Use Vertex Buffers
  Driver can optimize storage (alignment, format)

Reduce Vertex Size
  Use byte coordinates e.g. for normals

Reduce size of textures
  Textures cached but limited on-chip memory
  Use mip-mapping. Better quality and lower data transfer
  Use texture compression where available.
Memory Alignment

Always best to align data

Alignment to bus width very important
Alignment to DRAM pages gives some benefit

Embedded devices have different types of memory

Bus width may be 32 bit, 64 bit or 128 bit

Penalty for crossing a DRAM page (typically 1-4KB)

Can be several cycles.
Batch Data

Why?

- Big overhead for interpreting the command and setting up a new transfer
- Memory latency large

How?

- Combine geometry into large arrays.
- Use Vertex Buffer Objects
Batch Data

Don’t:
- Multiple API calls
- Lots of overhead
- Slow

Do:
- Fewer API calls
- Less overhead
- Much faster

Vertices → DrawArrays → GPU
Vertex Buffer Objects

Even better:

Step 1: Create

Client Side

Vertices

Server Side

VBO

Step 2: Draw

VBO

GPU

Driver can optimize storage
Vertex Buffer Objects

Ultimately:

Vertices -> VBO -> Local Memory -> GPU

Draw
Don’t interrupt the pipeline

GPUs are streaming architectures. They have high bandwidth but also long latencies.

Minimize state changes.
- Set up state and then render as much as possible.
- E.g. Don’t mix 2d and 3d operations

Don’t lock buffers unnecessarily
- OK at end of scene

Don’t use GetState unnecessarily
- Main usage is for 3rd party libraries. App should know what the state is.

All these can cause interruptions to data streaming.
- May even cause the entire pipeline to be flushed.
More on State Changes

Major changes of state will always be expensive
- Changing textures (consider using a texture atlas)
- Texture environment modes
- Light types (e.g. turning on specular)

Some are quite cheap
- Turning individual lights on and off
- Enabling/disabling depth buffer

BUT
- Varies with the hardware (and driver) implementation
Be nice to the Vertex Cache

Hardware typically holds 8-32 vertices.

- Smaller than in software implementations
- Likely to vary between implementations
- Possible to trade no. of vertices against vertex size in some implementations

Hardware cannot detect when two vertices are the same.

To make efficient use of the cache use:

- Vertex Strips
- Vertex fans
- Indexed triangles
Vertex Cache

Individual triangles: 3 vertices/triangle

Vertex Strip: 1 vertex triangle (ideal case)

Indexed triangles: 0.5 vertices/triangle (ideal case)
Depth Complexity

Drawing hidden triangles is expensive.

Costs:

- Transferring the geometry
- Transform & lighting
- Fetching texels for hidden pixels
- Reading and writing the colour and depth buffer
- Storing and reading the geometry (tiled or delay stream architectures)

Deferred rendering is not a magic solution

Can even make matters worse in some situations
Depth Complexity - solutions

Render scene front to back

- Eliminates many writes to colour and depth buffer
- Can reduce texture fetches (early z kill)

Use standard occlusion culling techniques

- View frustum culling (don’t draw objects that are out of view)
- DPVS
- Portals
Future Trends
Future Trends

- More local caches for graphics hardware
- Display resolution increasing
  - More pixel pipelines
- Better power management
- Integration of other functionality
  - Video
Questions?
M3G Intro

Kari Pulli
Nokia Research Center
Mobile 3D Graphics APIs

Native C/C++ Applications

Java applications

M3G (JSR-184)

OpenGL ES

Graphics Hardware
Why Should You Use Java?

Largest and fastest growing installed base
1200M phones running Java by June 2006
The majority of phones sold today support Java

Better productivity compared to C/C++
Much fewer opportunities to introduce bugs
Comprehensive, standardized class libraries
Java Will Remain Slower

Benchmarked on an ARM926EJ-S processor with hand-optimized Java and assembly code
M3G Design Principles

#1 No Java code along critical paths

Move all graphics processing to native code
Not only rasterization and transformations
Also morphing, skinning, and keyframe animation
All data on native side to avoid Java-native traffic
M3G Design Principles

#2 Cater for both software and hardware

Do not mandate hardware-only features
Such as per-pixel mipmapping or per-pixel fog

Do not try to expand the OpenGL pipeline
Such as with hardcoded transparency shaders
**M3G Design Principles**

- Address content creation and tool chain issues
  - Export art assets into a compressed file (.m3g)
  - Load and manipulate the content at run time
  - Need scene graph and animation support for that
- Minimize the amount of “boilerplate code”

**#3 Maximize developer productivity**
#4 Minimize engine complexity

#5 Minimize fragmentation

#6 Plan for future expansion
Why a New Standard?

OpenGL ES is too low-level
- Lots of Java code and function calls needed
- No support for animation and scene management

Java 3D is too bloated
- A hundred times larger (!) than M3G
- Still lacks a file format, skinning, etc.
M3G API
Overview

Tomi Aarnio
Nokia Research Center
Objectives

- Get an idea of the API structure and features
- Learn practical tricks not found in the spec
Prerequisites

Fundamentals of 3D graphics
Some knowledge of OpenGL ES
Some knowledge of scene graphs
M3G API Overview

Getting started
Rendering
Scene graph
Performance tips
Deformable meshes
Keyframe animation
Demos
Programming Model

M3G is not an “extensible scene graph”

Rather a black box – much like OpenGL

No interfaces, events, or render callbacks

No threads; all methods return only when done
Programming Model

Scene update is decoupled from rendering

- **render** ➔ Draw the scene, no side-effects
- **animate** ➔ Update the scene to the given time
- **align** ➔ Re-orient target cameras, billboards
Key Classes

- **Graphics3D**: 3D graphics context
  - Performs all rendering
- **Loader**: Loads individual objects and entire scene graphs
- **Mesh**: Encapsulates triangles, vertices and appearance
- **World**: Scene graph root node
Graphics3D: How to Use

Bind a target to it, render, release the target

```java
void paint(Graphics g) {
    myGraphics3D.bindTarget(g);
    myGraphics3D.render(world);
    myGraphics3D.releaseTarget();
}
```
Rendering State

Graphics3D contains global state
  Frame buffer, depth buffer
  Viewport, depth range

Most rendering state is in the scene graph
  Vertex buffers, textures, matrices, materials, …
  Packaged into Java objects, referenced by meshes
  Minimizes Java-native data traffic, enables caching
M3G API Overview

Getting started
Rendering
Scene graph
Performance tips
Deformable meshes
Keyframe animation
Demos
Renderable Objects

Sprite3D
- 2D image placed in 3D space
- Always facing the camera

Mesh
- Made of triangles
- Base class for meshes
Sprite3D

2D image with a position in 3D space

- Scaled mode for billboards, trees, etc.
- Unscaled mode for text labels, icons, etc.
- Too much overhead for particle effects
Mesh

One VertexBuffer, containing VertexArrays
1..N submeshes (IndexBuffer + Appearance)
## IndexBuffer Types

<table>
<thead>
<tr>
<th></th>
<th>Byte</th>
<th>Short</th>
<th>Implicit</th>
<th>Strip</th>
<th>Fan</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangles</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Lines</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Points</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Point sprites</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Relative to OpenGL ES 1.1
## VertexBuffer Types

<table>
<thead>
<tr>
<th></th>
<th>Byte</th>
<th>Short</th>
<th>Fixed</th>
<th>Float</th>
<th>2D</th>
<th>3D</th>
<th>4D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertices</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>TexCoords</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Normals</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Colors</td>
<td>✓</td>
<td></td>
<td>✗</td>
<td>✗</td>
<td>*</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>PointSizes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* M3G supports RGB color arrays, although OpenGL ES only supports RGBA
Vertex and Index Buffer Objects

Vertices and indices are stored on server side

Similar to OpenGL Buffer Objects

- Reduces data traffic from Java to native
- Allows caching, bounding box computation, etc.
Appearance Components

- **Material**: Colors for lighting, can track per-vertex colors.
- **Compositing Mode**: Blending, depth buffering, alpha testing, color masking.
- **Polygon Mode**: Winding, culling, shading, perspective correction hint.
- **Fog**: Fades colors based on distance, linear and exponential mode.
- **Texture2D**: Texture matrix, blending, filtering, one Texture2D for each unit.
Fragment (Pixel) Pipeline

[Diagram of a fragment pipeline with nodes labeled: Texture Blend, Texel Fetch, Texture Blend, Texel Fetch, Depth Buffer, Frame Buffer, Fog, Alpha Test, Depth Test, Blend & Mask, and CompositingMode. The diagram also includes a circle labeled Texture2D.]
M3G API Overview

Getting started
Rendering

Scene graph
Performance tips
Deformable meshes
Keyframe animation
Demos
Scene Graph

```
World
|-- Group
|-- Camera
|-- SkinnedMesh
|-- Light
  |-- Group
  |   |-- MorphingMesh
  |   |-- Group
  |     |-- Mesh
  |     |   |-- Group
  |     |     |-- Sprite
```

Not allowed!
Shared Node Components

- World
  - Mesh
  - Camera
  - SkinnedMesh
  - Appearance
  - Texture2D
  - Appearance
  - Mesh
Node Transformation

From this node to the parent node

Composed of four parts

Translation T
Orientation R
Non-uniform scale S
Generic 3x4 matrix M

$$C = T R S M$$
Node Alignment

Reorients a node towards a target node
Recomputes the orientation component (R)

For target cameras & lights, billboards, etc.
Other Node Features

Inherited properties

- Alpha factor (for fading in/out)
- Rendering enable (on/off)
- Picking enable (on/off)

Scope mask
Content Production

DCC tool
Exporter

Intermediate Format (e.g. COLLADA)

Optimizer & Converter

M3G Loader

Delivery Formats (.m3g, .png)

Runtime Scene Graph
M3G File Format

- Small size, easy to decode
- Matches API features 1:1
- Stores individual objects, entire scenes
- ZLIB compression of selected sections
- Can reference external files – e.g. textures
- Highly portable – no extensions
M3G API Overview

Getting started
Rendering
Scene graph
Performance tips
Deformable meshes
Keyframe animation
Demos
Use the Retained Mode

Render a World instead of separate objects

– Minimizes Java code and method calls
– Allows view frustum culling, etc.

Put co-located objects into Groups

– Speeds up hierarchic view frustum culling
Simplify Node Properties

Transformations

Favor the $T R S$ components over $M$

Avoid non-uniform scales in $S$

Use auto-alignment sparingly

Keep the alpha factor at 1.0
Optimize Rendering Order

`Appearance.setLayer(int layer)`

- Defines a **global** ordering for submeshes
- Within each layer, opaque objects come first

**Use layers for…**

- Making sure that overlays are drawn **first**
- Making sure that distant objects are drawn **last**
- Multipass effects (e.g. for lighting)
Optimize Texturing

Multitexturing is faster than multipass
   Transformation and setup costs cut by half

Use mipmaps to save memory bandwidth
   Tradeoff: 33% extra memory consumption

Combine small textures into a texture atlas
Use Perspective Correction

Much faster than increasing triangle count

Nokia: 2% fixed overhead, 20% in the worst case
No overhead at all on hardware implementations

Pitfall: Quality varies by implementation

Refer to quality scores at www.jbenchmark.com
Reduce Object Count

Per-Mesh processing overhead is high
Per-submesh overhead is also fairly high

Merge

Meshes that are close to each other
submeshes that have a common Appearance
Avoid Dynamic Geometry

`VertexArray.set(...)` can be slow

Java array contents must be copied in
May also trigger bounding box updates, etc.
Replace with morphing or skinning where possible

`IndexBuffers` have no `set(...)` method at all

`new IndexBuffer(...) per frame` is not a good idea
Switch between predefined `IndexBuffers` instead
Beware of Exporters

Exported content is often suboptimal

- Lighting enabled, but overwritten by texture
- Lighting disabled, normal vectors still included
- Alpha blending enabled, but alpha always 1.0
- 16-bit vertices when 8 bits would be enough
- Perspective correction always enabled

…

Always review the exported scene tree!
Hardware vs. Software

Shading state

SW: Minimize per-pixel operations
HW: Minimize shading state changes

Mixing 2D and 3D rendering

SW: No performance penalty
HW: Substantial penalty (up to 3x)
Layering 2D and 3D

2D backdrop
3D background
2D spectators
3D field
2D players
2D overlays

~7 layers of 2D and 3D!

Playman Beach Volley © RealNetworks, Inc.
Use Picking with Caution

myWorld.pick(...) can be very slow

Restrict the pick ray to
- meshes in a specific Group
- meshes with a specific scope mask

Use simplified geometry for picking
- setPickingEnable(true)
- setRenderingEnable(false)
Particle Effects

Point sprites – not available
Sprite3D – much too slow

Put all particles in one Mesh
One particle == two triangles
Animate by `VertexArray.set(...)`

Particles glued into a tri-strip
Easy Terrain Rendering

Split the terrain into tiles (Meshes)
Put the meshes into a scene graph
The engine will do view frustum culling
Terrain Rendering with LOD

Preprocess into a quadtree

leaf node == Mesh
inner node == Group

Use `setRenderingEnable` based on the view frustum
M3G API Overview

Getting started
Rendering
Scene graph
Performance tips
Deformable meshes
Keyframe animation
Demos
Deforming Meshes

- **MorphingMesh**
  - Vertex morphing mesh
- **SkinnedMesh**
  - Skeletally animated mesh
MorphingMesh

Traditional vertex morphing animation

Can morph any vertex attribute(s)

A base mesh \( B \) and any number of morph targets \( T_i \)

Result = weighted sum of morph deltas

\[
R = B + \sum_i w_i (T_i - B)
\]

Change the weights \( w_i \) to animate
MorphingMesh

Base

Target 1
eyes closed

Target 2
mouth closed

Animate eyes
and mouth
independently
Articulated characters without cracks at joints
Stretch a mesh over a hierarchic “skeleton”
The skeleton consists of scene graph nodes
Each node (“bone”) defines a transformation
Each vertex is linked to one or more bones

\[ v' = \sum_{i} w_i M_i B_i v \]

\( M_i \) are the node transforms – \( v, w, B \) are constant
SkinnedMesh

Neutral pose, bones at rest
SkinnedMesh

Bone B rotated 90 degrees

- Position in A's coordinate system
- Interpolated position
- Position in B's coordinate system
SkinnedMesh

Mesh

SkinnedMesh
M3G API Overview

Getting started
Rendering
Scene graph
Performance tips
Deformable meshes
Keyframe animation
Demos
Animation Classes

KeyframeSequence
- Storage for keyframes
- Defines interpolation, looping

AnimationController
- Controls the playback of one or more sequences

AnimationTrack
- A link between sequence, controller and target

Object3D
- Base class for all objects that can be animated
Animation Classes

- Object3D
- AnimationTrack
  - Identifies animated property on this object
  - AnimationController
  - KeyframeSequence
KeyframeSequence: Interpolation Modes

...plus SLERP and SQUAD for quaternions
AnimationController: Timing and Speed

Maps world time into sequence time
Can control any number of sequences

Diagram courtesy of Sean Ellis, ARM
Animation

1. Call animate(worldTime)

Object3D

AnimationTrack

AnimationController

KeyframeSequence

2. Calculate sequence time from world time

3. Look up value at this sequence time

4. Apply value to animated property

Diagram courtesy of Sean Ellis, ARM
Animation

Tip: Interpolate quaternions as ordinary 4-vectors

Supported in HI Corp’s M3G Exporter

SLERP and SQUAD are slower, but need less keyframes

Quaternions are automatically normalized before use
M3G API Overview

Getting started
Rendering
Scene graph
Performance tips
Deformable meshes
Keyframe animation
Demos
Summary

M3G enables real-time 3D on mobile Java
Minimizes Java code along critical paths
Designed for both software and hardware
OpenGL ES features at the foundation
Animation & scene graph layered on top

30’000 devices sold during this presentation
Demos
Playman Winter Games – RealNetworks

2D

Perspective and depth

3D

Side view only

Restart

Restart

Restart

96p 91.01m

playman1

96p 91.01m

playman1

96p 91.01m

playman1

96p 91.01m

playman1

96p 91.01m
Playman World Soccer – RealNetworks

2D/3D hybrid
Cartoon-like
2D figures in a 3D scene
2D particle effects etc.
Tower Bloxx – Digital Chocolate

- Puzzle/arcade mixture
- Tower building mode is in 3D, with 2D overlays and backgrounds
- City building mode is in pure 2D
Mini Golf Castles – Digital Chocolate

3D with 2D background and overlays
Skinned characters
Realistic ball physics
Rollercoaster Rush – Digital Chocolate

- 2D backgrounds
- 3D main scene
- 2D overlays
Q&A

Thanks:
Sean Ellis (ARM)
Kimmo Roimela (Nokia)
Markus Pasula (RealNetworks)
Sami Arola (Digital Chocolate)
M3G in the Real World

Mark Callow
Chief Architect
An M3G Game
Rollercoaster Rush 3D™
Agenda

J2ME game development
Tools
COFFEE BREAK
The structure of a MIDlet
A walk through a sample game
Why mobile game development is hard
Publishing your content
Agenda

J2ME game development

• Tools
• COFFEE BREAK
• The structure of a MIDlet
• A walk through a sample game
• Why mobile game development is hard
• Publishing your content
Game Development Process

Traditional Java Game

Game logic ➔ Compile ➔ Java MIDlet ➔ Package ➔ JAR file

Assets

Images  Sounds  Music  Other

Game Platform

2D Graphics  Sound  Network  Proprietary  Other

Diagram courtesy of Sean Ellis, ARM.
Screen Image: Boulder Dash®-M.E.™
M3G Game Development Process

How M3G Fits

- Compile
  - Java MIDlet
- Package
  - JAR file

Assets
- Images
- Sounds
- Music
- 3D World

Game Platform
- 2D Graphics
- Sound
- Network
- Proprietary
- 3D Graphics

Diagram courtesy of Sean Ellis, ARM.
Screen Image: Sega/Wow Entertainment RealTennis™
Development Team Structure

- Planner/Producer
- Programmers
- Designers
Agenda

- J2ME game development
- Tools
- COFFEE BREAK
- The structure of a MIDlet
- A walkthrough a sample game
- Why mobile game development is hard
- Publishing your content
Tools Agenda

Tools

Creating your assets

Programming tools & development platforms
Creating Your Assets: Images

Textures & Backgrounds

- Expanded game logic
- Compile
- Assets
- Sounds
- Music

Images

- Image Editor with PNG output. E.g:
  - Adobe Fireworks
  - Adobe Photoshop

3D Graphics

Distribute

Game Platform

- 3D Graphics
- Sound
- 2D Graphics
- Network

Proprietary

Creating Your Assets: Images

• Textures & Backgrounds

Image Editor with PNG output. E.g:

- Adobe Fireworks
- Adobe Photoshop
Creating Your Assets: Sounds

Audio Tools

- Expanded game logic

Compile

Assets

Images

Sounds

Music

Distribute

Audio Production Tool; e.g., Sony Sound Forge®

Commonly Used Formats:
- WAV, AU, MP3, SMAF

3D Graphics
Creating Your Assets: Music

Music Tools

- MIDI Sequencer; e.g. Steinberg Cubase
  - SMAF, MIDI, cMIDI, MFi

Expanded game logic

Compile

Java

Assets

Images Sounds Music

Proprietary

3D Graphics
Creating Your Assets: 3d Models

<table>
<thead>
<tr>
<th>Modeling Tools</th>
<th>Compile</th>
<th>Java MIDlet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expanded game logic</td>
<td>Images</td>
<td>Sounds</td>
</tr>
</tbody>
</table>

**3D World**

3d Modeler with M3G plug-in; e.g.
- Lightwave
- Maya
- 3d studio max
- Softimage|XSI

Assets
- Images
- Sounds
- Music

Game Platforms
- 3D Graphics
- Sound
- 2D Graphics
- Network
- Proprietary

Java MIDlet Packages
- Package JAR file

Distribute
- Game Platform
Export 3d Model to M3G
M3G File Viewer
On an Emulator
Tips for Designers 1

**TIP: Don’t use GIF files**

The specification does not require their support

**TIP: Create the best possible quality audio & music**

It’s much easier to reduce the quality later than increase it

**TIP: Polygon reduction tools & polygon counters are your friends**

Use the minimum number of polygons that conveys your vision satisfactorily
Tips for Designers 2

**TIP: Use light maps for lighting effects**
- Usually faster than per-vertex lighting
- Use luminance textures, not RGB
- Multitexturing is your friend

**TIP: Try LINEAR interpolation for Quaternions**
- Faster than SLERP
- But less smooth
Tips for Designers 3

TIP: Favor textured quads over Background & Sprite3D

Background and Sprite3D will be deprecated in M3G 2.0

Were intended to speed up software renderers

but implementation is complex, so not much speed up and

no speed up at all with hardware renderers

Nevertheless Sprite3Ds are convenient to use for 2D

overlays and Backgrounds are convenient when

background scrolling is required.

LIMITATION: Sprites not useful for particle systems
Tools Agenda

Tools

– Creating your assets

Programming tools & development platforms
Program Development

Edit, Compile, Package

Expanded game logic

Compile
Java MIDlet
Package
JAR file

Assets
Images Sounds Music Resources

Distribute

Traditional
• WTK, shell, editor, make, javac

Integrated Development Environment
• Eclipse + EclipseME
• Borland JBuilder + J2ME Wireless Toolkit
• NetBeans IDE + Mobility Pack
Program Development

Test & Debug

- Expanded game logic
- Compile
- Java MIDlet

Assets
- Images
- Sounds
- Music
- 3D World

Game Platform
- 2D Graphics
- Sound
- Network
- Proprietary
- 3D Graphics

Operator/Maker supplied SDK
- Emulator
- Simulator
- Real device

Screen Image: Sega/Wow Entertainment RealTennis.™
Java Wireless Toolkit 2.5.1 for CLDC

KToolBar

Handset Emulator
NetBeans + Mobility Pack + SE SDK
Java Debugging

JPDA Debugger

{ NetBeans
  Eclipse, JBuilder }

Socket Connection

JavADebugWireProtocol

KVM in Emulator

Emulator

Debug Agent

Socket Connection

KvmDebugWireProtocol

On Device

Connection Proxy

Serial Connection

SLIP

KVM on Device

KVM on Device
Agenda

- J2ME game development
- Tools
- COFFEE BREAK
- The structure of a MIDlet
- A walk through a sample MIDlet
- Why mobile game development is hard
- Publishing your content
Agenda

- J2ME game development
- Tools
- COFFEE BREAK

The structure of a MIDlet

- A walk through a sample game
- Why mobile game development is hard
- Publishing your content
The Simplest MIDlet

Derived from MIDlet,

Overrides three methods

\[\text{MIDlet}.\text{StartApp}()\]
- \[\text{initialize}\]
- \[\text{request redraw}\]

\[\text{MIDlet}.\text{destroyApp}()\]
- \[\text{shut down}\]

\[\text{Canvas}.\text{paint}()\]
performs rendering using Graphics3D object.

Create canvas; load world.

Tidy up; exit MIDlet.

And that's it.
A More Interesting MIDlet

**MIDlet.StartApp()**
Create canvas; load world, start update thread

**user input**
Get any user input via Canvas.commandListener

**scene update**
Game logic, animate, align if necessary

**request redraw**
Wait to ensure consistent frame rate

**wait**
Canvas.paint()
performs rendering using Graphics3D object

**draw**

**initialize**
MIDlet.destroyApp()
Tidy up; exit MIDlet

**shut down**

**Update loop.**
Runnable.run()
Read user input, update scene

Flow-chart courtesy of Sean Ellis, ARM
MIDlet Phases

- Initialize
- Update
- Draw
- Shutdown
Initialize

Load assets: world, other 3D objects, sounds, etc.
Find any objects that are frequently used
Perform game logic initialization
Initialize display
Initialize timers to drive main update loop
Update

Usually a thread driven by timer events
Get user input
Get current time
Run game logic based on user input
Game logic updates world objects, if necessary
Animate
Request redraw
Update Tips

TIP: Don’t create or release objects if possible

TIP: Call system.gc() regularly to avoid long pauses

TIP: cache any value that does not change every frame; compute only what is absolutely necessary
Draw

- Usually on overridden paint method
- Bind Graphics3D to screen
- Render 3D world or objects
- Release Graphics3D
  - …whatever happens!
- Perform any other drawing (UI, score, etc)
- Request next timed update
Draw Tips

TIP: Don’t do 2D drawing while Graphics3D is bound
Shutdown

- Tidy up all unused objects
- Ensure once again that Graphics3D is released
- Exit cleanly
- Graphics3D should also be released during pauseApp
MIDlet Review

Update loop.
Don’t create/destroy objects if possible
Throttle to consistent frame rate
Keep paint() as simple as possible
Be careful with threads

Exit request

Set up display, load assets, find commonly used objects, initiate update thread.

Get any user input, network play, etc.

Game logic, animate, align if necessary

Wait to ensure consistent frame rate

Release assets, tidy up

Graphics3D object performs rendering

Diagram courtesy of Sean Ellis, ARM
Agenda

• J2ME game development
• Tools
• COFFEE BREAK
• The structure of a MIDlet
  A walk through a sample game
• Why mobile game development is hard
• Publishing your content
Demo: *GhostHunt*
**GhostHunt**

Arrow keys move a “plasma” racquet side to side to hit a “plasma” ball

Ball hits deform ghost houses and make the ghosts disappear

 Loads data from .m3g and .png files

 Uses Immediate mode

 Uses 2D for sky and scores
GhostHunt Models
GhostHunt Assets

Push start
Stage
Game over
Clear
Go!
GhostHunt Framework

MainApp.java – MIDlet specialization; handles initialization & data loading; contains run thread
SubApp.java – canvas specialization
Math2.java – math library
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.m3g.*;

class MainApp extends MIDlet implements CommandListener {
    MainApp() {
        exit_command = new Command("Exit", Command.EXIT, 0);
        select_command = new Command("Debug", Command.SCREEN, 0);

        /* Create canvas */
        subapp = new SubApp();
        subapp.addCommand (exit_command);
        subapp.addCommand (select_command);
        subapp.setCommandListener (this);

        SystemInit();
        prog_number = PROG_SPLASH;
        WorkInit();
        GameInit();
        DataLoad();
    }
}
GhostHunt: loading data

DataLoad() {
   try {
      image [TITLE_SP] = Image.createImage("/title.png");
      ...
   } catch (Exception e) {
      System.out.println("------------- SP Load");
      ApplicationEnd();
   }

   try {
      load_data [RACKET_DATA] = Loader.load("/racket.m3g");
   } catch (Exception e) {
      ...
   }

   mesh [RACKET_DATA] = (Mesh)load_data [RACKET_DATA][0];
   vbuf [RACKET_DATA] = mesh [RACKET_DATA].getVertexBuffer();
   ibuf [RACKET_DATA] = mesh [RACKET_DATA].getIndexOfBuffer(0);
   app [RACKET_DATA] = mesh [RACKET_DATA].getAppearance(0);
   ...
}
public void startApp () { 
    thread = new Thread () {
        public void run () {
            GameStart ();
        }
    };
    // Call the new thread’s run method.
    thread.start ();
}

public void pauseApp () {
    thread = null;
}

public void destroyApp (boolean unconditional) {
    ApplicationEnd();
}
void GameStart () {
    Thread thisThread = Thread.currentThread();
    Display.getDisplay (this).setCurrent (subapp);
    while (thread == thisThread) {
        prev_time = now_time;
        do {
            now_time = System.currentTimeMillis ();
        } while ((now_time - prev_time) < SYSTEM_SPEED);
        loop_rate = (now_time - prev_time) / SYSTEM_SPEED;
        if (loop_rate > 5.0f) { /* More than loop rate limit */
            loop_rate = 5.0f;
        }
        /* do game stuff here ... */
        try {
            Thread.sleep (1);
        } catch (InterruptedException e) {
            ApplicationEnd ();
        }
    }
}
`void GameStart () {
...
switch (prog_number) {
    case PROG_SPLASH: /* Splash */
        SplashProg ();
        break;
    case PROG_TITLE: /* Title */
        TitleProg ();
        break;
    case PROG_GAME: /* Game */
        GameProg ();
        break;
}
...
}`
void TitleProg ()
{
    key_dat = subapp.sys_key; /* Get keypresses */

    if ((key_dat & KEY_FIRE) != 0) /* it is fire key */
    {
        racket_tx = 0.0f;
        racket_tz = 0.0f; /* for initializing camera */
        WorkInit ();
        GameInit ();

        prog_number = PROG_GAME;
    }

    /*------ Updating------*/
    start_loop++;

    /*------ Drawing ------*/
    subapp.repaint ();
}

GhostHunt: TitleProg
public class SubApp extends Canvas {
    int cnt;
    static int keydata [] = { UP, LEFT, RIGHT, DOWN, FIRE };
    int length = keydata.length;

    static int sys_key = 0;

    synchronized public void paint (Graphics graphics) { }

    ... 

    protected void keyPressed (int key) { }

    protected void keyRepeated (int key) { }

    protected void keyReleased (int key) { }
}
GhostHunt: key handling

```
static int keydata[] = { UP, LEFT, RIGHT, DOWN, FIRE };

protected void keyPressed(int key) {
    for (cnt = 0; cnt < length; cnt++) { /* Search key data. */
        if (getGameAction(key) == keydata[cnt]) {
            sys_key |= (1 << cnt);
        }
    }
}

protected void keyReleased(int key) {
    for (cnt = 0; cnt < length; cnt++) { /* Search key data. */
        if (getGameAction(key) == keydata[cnt]) {
            sys_key &= ~(1 << cnt);
        }
    }
}
```
synchronized public void paint (Graphics graphics) {
    /*------- select drawing process -------*/
    switch (MainApp.prog_number) {
        case MainApp.PROG_SPLASH:
            SplashDraw (graphics); /* Splash */
            break;

        case MainApp.PROG_TITLE:
            TitleDraw (graphics); /* Title */
            break;

        case MainApp.PROG_GAME:
            GameDraw (graphics); /* Game */
            break;
    }
    Math2.Rand ();
}
void GameDraw (Graphics graphics) {
    ...
    graphics.drawImage (MainApp.image[MainApp.BG_SP], 0, 0, Graphics.TOP | Graphics.LEFT); /* 2D background sprite */

    MainApp.g3d.bindTarget (graphics);
    MainApp.g3d.clear (MainApp.background);

    /*------ camera setup ------*/
    ...
    /*------ draw 3D objects ------*/
    ...

    MainApp.g3d.releaseTarget ();

    /*------ draw score, items etc. in 2D ------*/
    ...
}
**GameDraw: camera set-up**

```java
MainApp.ctrans.setIdentity();
MainApp.ctrans.postTranslate( MainApp.camera_tx,
                             MainApp.camera_ty,
                             MainApp.camera_tz );
MainApp.ctrans.postRotate( MainApp.camera_ry,
                           0.0f, 1.0f, 0.0f );
MainApp.ctrans.postRotate( MainApp.camera_rx,
                           1.0f, 0.0f, 0.0f );
MainApp.ctrans.postRotate( MainApp.camera_rz,
                           0.0f, 0.0f, 1.0f );
MainApp.g3d.setCamera( MainApp.camera, MainApp.ctrans );
```
for (count = 0; count != MainApp.GHOST_MAX; count++)
{
    if (MainApp.ghost_draw_flag [count] != 0) {
        data = count * 2;
        x    = MainApp.ghost_xz [data + 0];
        z    = MainApp.ghost_xz [data + 1];
        r    = MainApp.ghost_r  [count   ];
        trans = MainApp.trans[MainApp.GHOST_M + count];

        trans.setIdentity ();
        trans.postTranslate (x, 0.0f, z);
        trans.postRotate   (r, 0.0f, 1.0f, 0.0f);
        trans.postScale   (MainApp.ghost_scale [count],
                           MainApp.ghost_scale [count],
                           MainApp.ghost_scale [count]);

        MainApp.g3d.render (MainApp.vbuf  [MainApp.GHOST_DATA],
                            MainApp.ibuf  [MainApp.GHOST_DATA],
                            MainApp.app   [MainApp.GHOST_DATA],
                            trans);
    }
}
GameProg

```c
void GameProg() {
    key_dat_old = key_dat; /*---- Get key data ----*/
    key_dat     = subapp.sys_key;

    CameraWorldSet ();
    if (Math2.DistanceCalc2D (0.0f, 0.0f, ball_tx, ball_tz) > 1.5f) {
        CameraSet (15.0f * (1.0f / loop_rate));
    }

    if (freeze_time == 0) /* The Game is not frozen */ {
        /*------- do game calculations ------*/
        ...
    }
    EffectProg ();
    subapp.repaint ();
    ...
}
```
**GameProg**: do game calculations

```c
RacketProg (key_dat, key_dat_old); /*-- Plasma Racket --*/

if (racket_break_flag != 1) /*-- Racket not destroyed --*/
    BallProg ();

GhostProg ();

if (racket_break_flag != 1) /*-- Racket not destroyed --*/ {
    BallHit ();      /*--- Collision Decision ---*/
    RacketBreakCheck ();
}

house = HouseCheck (); /*------ Final Check ------*/
if (house == 0)  /* All ghost houses are destroyed. */ {
    /*------ make all remaining ghosts disappear ------*/
    ...
    freeze_time = (int)(MOJI_CLEAR_WAIT * (1.0f/loop_rate));
    moji_number = MOJI_CLEAR;
}
```
BallProg: compute new ball position

```cpp
void BallProg () {
    ...          
    ball_speed_rate = ball_speed * loop_rate;
    dis = Math2.DistanceCalc2D(ball_tx, ball_tz, 0.0f, 0.0f);
    pd = Math2.DistanceCalc2D(ball_tx2, ball_tz2, 0.0f, 0.0f);
    if ((dis > 2.0f) && pd > dis)) /* Homing is necessary */ {  
        angle = Math2.AngleCalc (ball_tx, ball_tz, 0.0f, 0.0f);
        if (Math2.DiffAngleCalc (angle, ball_vec) > 0.0f) {
            ball_vec -= (0.6f * loop_rate);
        } else {
            ball_vec += (0.6f * loop_rate);
        }
    }
    Math2.RotatePointCalc (ball_speed_rate, ball_vec);
    ball_tx2 = ball_tx; /* Save the previous coordinates */
    ball_tz2 = ball_tz;
    ball_tx += Math2.calc_x;
    ball_tz += Math2.calc_y;
}
```
void BallHit () {
    ...
    /*------- racket collision detection -------*/
    ...
    /*------- ghost house collision detection -------*/
    ...
    /*------- ghost collision detection -------*/
    ...
    /*------- obstacle (cross) collision detection -------*/
    ...
    /*------- warp hole collision detection -------*/
    ...
    /*------- check for outside the field -------*/
    ...
}
void BallHit () {
    ... /* final static int Math2.ANGLE = 360 */
    dist = Math2.DistanceCalc2D (racket_tx, racket_tz,
                               ball_tx, ball_tz);
    if (dist <= BALL_RACKET_DISTANCE) {
        angle = Math2.AngleCalc (ball_tx, ball_tz, racket_tx,
                                  racket_tz);
        diff  = Math2.DiffAngleCalc (angle, ball_vec
                                      + (Math2.ANGLE/2.0f));
        if (Math2.Absf (diff) > (Math2.ANGLE / 4.0f)) {
            /* Feasible angle for collision */
            ball_vec = angle + (diff * -1.0f);

            Math2.RotatePointCalc (ball_speed_rate, ball_vec);
            ball_tx  = ball_tx2 + Math2.calc_x;
            ball_tz  = ball_tz2 + Math2.calc_y;
        }
    }
    ...
}
Improvement 1: simpler drawing

for (count = 0; count != MainApp.GHOST_MAX; count++)
{
    if (MainApp.ghost_draw_flag [count] != 0) {
        ...
        MainApp.g3d.render (MainApp.vbuf [MainApp.GHOST_DATA],
                            MainApp.ibuf [MainApp.GHOST_DATA],
                            MainApp.app [MainApp.GHOST_DATA],
                            trans);
        MainApp.g3d.render (MainApp.mesh[MainApp.GHOST_DATA],
                            trans)
    }
}
Improvement 2: no busy waiting

```java
while (thread == thisThread) {
    prev_time = now_time;
    do {
        now_time = System.currentTimeMillis();
    } while ((now_time - prev_time) < SYSTEM_SPEED);
    loop_rate = (now_time - prev_time) / SYSTEM_SPEED;
    if (loop_rate > 5.0f) { /* More than loop rate limit */
        loop_rate = 5.0f;
    }
    /* do game stuff here ... */
    try {
        Thread.sleep (1);
    } catch (InterruptedException e) {
        ApplicationEnd ();
    }
}
```
while (thread == thisThread) {
    prev_time = now_time;
    do {
        now_time = System.currentTimeMillis();
    } while ((now_time - prev_time) < SYSTEM_SPEED);
    now_time = System.currentTimeMillis();
    long sleep_time = SYSTEM_SPEED + prev_time - now_time;
    if (sleep_time < 0)
        sleep_time = 1; /* yield anyway so other things can run */
    try {
        Thread.sleep(sleep_time);
    } catch (InterruptedException e) {
        ApplicationEnd ();
    }
    if (thread != thisThread) return;
    now_time = System.currentTimeMillis();
    loop_rate = (now_time - prev_time) / SYSTEM_SPEED;
    if (loop_rate > 5.0f) { /* More than loop rate limit */
        loop_rate = 5.0f;
    }
    /* do game stuff here ... */
}
Programming Tricks

- Use per-object fog to highlight objects
- Use black fog for night time
- Draw large background objects last
- Draw large foreground objects first
- Divorce logic from representation
Agenda

• J2ME game development
• Tools
• COFFEE BREAK
• The structure of a MIDlet
• A walkthrough a sample game

Why mobile game development is hard

• Publishing your content
Why Mobile Game Development is Hard

Device Fragmentation

Porting platforms and tools are available:


Porting and testing services are available:

www.tirawireless.com

For some self-help using NetBeans see

J2ME MIDP Device Fragmentation Tutorial with Marv The Miner
Why Mobile Game Development is Hard

Severe limits on application size

- Download size limits
- Small Heap memory

Small screens

Poor input devices

Poor quality sound

Slow system bus and memory system
Why Mobile Game Development is Hard

No floating point hardware

No integer divide hardware

Many tasks other than application itself
  - Incoming calls or mail
  - Other applications

Short development period

Tight $100k – 250k budget
Memory

Problems

Small application/download size
Small heap memory size

Solutions

Compress data ①
Use single large file ①
Use separately downloadable levels ①
Limit contents ②

Optimize your Java: combine classes, coalesce var’s, eliminate temporary & local variables, … ②
Performance

Problems

- Slow system bus & memory
- No integer divide hardware

Solutions

- Use smaller textures
- Use mipmapping
- Use byte or short coordinates and key values
- Use shifts
- Let the compiler do it
User-Friendly Operation

Problems

- Button layouts differ
- Diagonal input may be impossible
- Multiple simultaneous button presses not recognized

Solutions

- Plan carefully
- Different difficulty levels
- Same features on multiple buttons
- Key customize feature
Many Other Tasks

Problem

- Incoming calls or mail
- Other applications

Solution

Create library for each handset terminal
Agenda

- J2ME game development
- Tools
- COFFEE BREAK
- The structure of a MIDlet
- A walkthrough a sample game
- Why mobile game development is hard

Publishing your content
Publishing Your Content Agenda

Publishing your content

Preparing contents for distribution

– Getting published and distributed
Preparing for Distribution: Testing

Testing on actual handsets essential

May need contract with operator to obtain tools needed to download test MIDlets to target handset.

May need contractor within operator’s region to test over-the-air aspects as handset may not work in your area

Testing services are available

e.g. www.tirawireless.com
Preparing for Distribution: Signing

Java has 4 security domains:

- Manufacturer
- Operator
- 3rd Party
- Untrusted

Most phones will not install untrusted MIDlets

If untrusted MIDlets are allowed, there will be limits on access to certain APIs

Operators will not allow untrusted MIDlets in their distribution channels
Preparing for Distribution: Signing

Your MIDlet must be certified and signed using a 3rd party domain root certificate.

Method varies by operator and country.

Many makers and operators participate in the Java Verified Program to certify and sign MIDlets for them.

To get certification, MIDlet must meet all criteria defined by JVP and must pass testing.
Publishing Your Content Agenda

Publishing your content

- Preparing contents for distribution

Getting published and distributed
Publishing Your Content: Distribution Channels

Game deck
  e.g. “More Games button”

Off deck, in portal
  e.g. AT&T Wireless’s Beyond MEdia Net

Off portal
  Independent of operator
  Premium SMS or web distribution
Distribution Channels: Game Deck

Customers find you easily

but many carriers only allow a few words of text to describe and differentiate the on-deck games

Operator does billing

No credit worries

Operator may help with marketing

or they may not

Shelf space limited
Distribution Channels: off Deck, in Portal

Hard to find you. Need viral marketing

Customers must enter search terms in operator’s search box

or find URL in some other way

Operator does billing, may help with marketing

May be able to get here without a publisher
**Distribution Channels:**
**off Deck, off Portal**

Very hard for customers to find you

Only 4% of customers have managed to buy from the game deck!

You have to handle billing

Typical game prices of $2 - $6 too low for credit cards. Must offer subscription service for CC billing.

Nobody is going to enter your url then billing information on a 9-key pad and very few people will use a PC to buy games for their phone.

Premium SMS or advertiser funded are about the only ways.

You take all the risks

Some handsets/carriers do not permit off-portal downloads
Publishing Your Content
Billing Mechanisms

- One-time purchase via micropayment
  - Flat-rate data? ➔ Larger, higher-cost games

- Subscription model via micropayment
  - Episodic games to encourage loyalty
  - Game arcades with new games every month

- Sending Premium SMS
  - Triggers initial download
  - Periodically refills scarce supplies
Operator Revenue Share 1999 - 2004

Source: www.roberttercek.com
Going On-Deck

Find a publisher and build a good relationship with them

**Japan:** Square Enix, Bandai Networks, Sega WOW, Namco, Infocom, etc.

**America:** Bandai America, Digital Chocolate, EA Mobile, MForma, Sorrent

**Europe:** Digital Chocolate, Superscape, Macrospace, Upstart Games
Going Off-Deck

There are off-deck distribution services:

- thumbplay, www.thumbplay.com
- playphone, www.playphone.com
- gamejump, www.gamejump.com free advertiser supported games

These services may be a good way for an individual developer to get started
Other 3D Java Mobile APIs

Mascot Capsule Micro3D Family APIs

Motorola iDEN, Sony Ericsson, Sprint, etc.
com.mascotcapsule.micro3d.v3 (V3)

Vodafone KK JSCL
com.j_phone.amuse.j3d (V2), com.jblend.graphics.j3d (V3)

Vodafone Global
com.vodafone.amuse.j3d (V2)

NTT Docomo (DoJa)
com.nttdocomo.opt.ui.j3d (DoJa2, DoJa 3) (V2, V3)
com.nttdocomo.ui.graphics3D (DoJa 4, DoJa 5) (V4)

(Vx) - Mascot Capsule Micro3D Version Number
Mascot Capsule V3 Game Demo

Copyright 2006, by Interactive Brains, Co., Ltd.
Summary

Use standard tools to create assets
Many J2ME SDKs and IDEs are available
Basic M3G MIDlet is relatively easy
Programming 3D Games for mobile is hard
Getting your content marketed, distributed and sold is a huge challenge
Exporters

3ds max
- Simple built-in exporter since 7.0
  - www.digi-element.com/Export184/
  - www.mascotcapsule.com
  - www.m3gexporter.com

Maya
- www.mascotcapsule.com
- www.m3gexport.com

Softimage|XSI
- www.mascotcapsule.com

Cinema 4D
- www.tetracon.de/public_main_modul.php?om=&ses=&page_id=453&document_id=286&unit=441299c9be098

Lightwave
- www.mascotcapsule.com

Blender
- www.nelson-games.de/bl2m3g/

M3GToolkit
- www.java4ever.com

Not a typo

genware?
SDKs

Motorola iDEN J2ME SDK

idenphones.motorola.com/iden/developer/developer_tools.jsp

Nokia Series 40, Series 60 & J2ME

www.forum.nokia.com/java

Softbank MEXA & JSCL SDKs

developers.softbankmobile.co.jp/dp/tool_dl/java/tech.php

developers.softbankmobile.co.jp/dp/tool_dl/java/emu.php
SDKs

Sony Ericsson

developer.sonyericsson.com/java

Sprint Wireless Toolkit for Java

developer.sprintpcs.com

Sun Java Wireless Toolkit 2.5.1 for CLDC


Vodafone VFX SDK

via.vodafone.com/vodafone/via/Home.do
IDE’s for Java Mobile

Eclipse Open Source IDE

www.eclipse.org & eclipseme.org

JBuilder 2005 Developer

www.borland.com/jbuilder/developer/index.html

NetBeans

www.netbeans.info/downloads/index.php

www.netbeans.org/products/

Comparison of IDE’s for J2ME

www.microjava.com/articles/J2ME_IDE_Comparison.pdf
Other Tools

Macromedia Fireworks
  www.adobe.com/products/fireworks/

Adobe Photoshop
  www.adobe.com/products/photoshop/main.html

Sony SoundForge

Steinberg Cubase
  www.steinberg.de/33_1.html

Yamaha SMAF Tools
  smaf-yamaha.com/
Other Tools

Java optimizer - Innaworks mBooster

www.innaworks.com/mBooster.html

Porting Platforms

www.tirawireless.com

www.javaground.com
Services

MIDlet verification & signing

www.javaverified.com

Porting & testing

www.tirawireless.com

Off deck distribution

www.thumbplay.com
www.playphone.com
www.gamejump.com
犬友 (Dear Dog) Demo
Thanks to: Koichi Hatakeyama; HI’s MascotCapsule Version 4 Development Team; Sean Ellis; JSR-184 & JSR-297 Expert Groups
M3G 2.0
Sneak Preview

Tomi Aarnio
Nokia Research Center
What is M3G 2.0?

Mobile 3D Graphics API, version 2.0

Java Specification Request 297
Successor to M3G 1.1 (JSR 184)

Work in progress

Public Review Draft is out (www.jcp.org)
Developer feedback is much appreciated!
<table>
<thead>
<tr>
<th>Who’s Behind It?</th>
<th>Hardware vendors</th>
<th>Device makers</th>
<th>Platform providers</th>
<th>Developers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AMD, ARM</td>
<td>Nokia, Sony Ericsson</td>
<td>Sun, Ericsson</td>
<td>Digital Chocolate</td>
</tr>
<tr>
<td></td>
<td>NVIDIA, PowerVR</td>
<td>Motorola, Samsung</td>
<td>HI, Aplix, Acrodea</td>
<td>RealNetworks</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Superscape</td>
</tr>
</tbody>
</table>
M3G 2.0 Preview

Design

Fixed functionality

Programmable shaders

New high-level features

Summary, Q&A
## Design Goals & Priorities

<table>
<thead>
<tr>
<th>Target all devices</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmable HW</td>
<td></td>
</tr>
<tr>
<td>No graphics HW</td>
<td></td>
</tr>
<tr>
<td>Fixed-function HW</td>
<td></td>
</tr>
</tbody>
</table>
Why Not Shaders Only?

Device sales in 2010?

No Graphics Hardware

Fixed Function Hardware

Shader Hardware
Shaders and Fixed Functionality

- M3G 2.0
- OpenGL ES 2.0
- OpenGL ES 1.1
Shaders and Fixed Functionality

M3G 2.0

OpenGL ES 1.1

OpenGL ES 2.0

Mandatory

Optional
## Design Goals & Priorities

<table>
<thead>
<tr>
<th>Target all devices</th>
<th>Enable reuse of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmable HW</td>
<td>Assets &amp; tools (.m3g)</td>
</tr>
<tr>
<td>No graphics HW</td>
<td>Source code (.java)</td>
</tr>
<tr>
<td>Fixed-function HW</td>
<td>Binary code (.class)</td>
</tr>
</tbody>
</table>
Backwards Compatible – Why?

Device vendors can drop M3G 1.1
Rather than supporting both versions (forever)
Cuts integration, testing & maintenance into half

Developers can upgrade gradually
Rather than re-doing all code, art, and tools
Backwards Compatible – How?

- M3G 2.0 Core
- M3G 2.0 Advanced
- M3G 1.1
Backwards Compatible – How?

- M3G 1.1
- M3G 2.0 Core
- Advanced

Mandatory
Optional
The Downsides

Must emulate fixed functionality on shader HW
   Extra implementation burden

The API is not as compact as it used to be
   A pure shader API could have ~20% fewer classes

Need to drag along obsolete features
   Flat shading, Sprite3D, Background image
   Can be deprecated, but not totally removed
Core vs. Advanced

High-level features are common to both

- Scene graph
- Animation

The differences are in rendering

- Core → OpenGL ES 1.1
- Advanced → OpenGL ES 2.0
What’s in the Core?

Everything that’s in M3G 1.1

Everything that’s in OpenGL ES 1.1
  Except for useless or badly supported stuff
  Such as points, logic ops, stencil, full blending

A whole bunch of new high-level features
What’s in the Advanced Block?

Everything that’s in OpenGL ES 2.0

- Vertex and fragment shaders
- Cube maps, advanced blending
- Stencil buffering
What’s not included?

Optional extensions of OpenGL ES

- Floating-point textures
- Multiple render targets
- Depth textures, 3D textures
- Non-power-of-two mipmaps
- Occlusion queries
- Transform feedback
M3G 2.0 Preview

Design

Fixed functionality

Programmable shaders

New high-level features

Summary, Q&A
M3G 2.0 Core vs. M3G 1.1

New capabilities
Better and faster rendering
More convenient to use
Fewer optional features
Point Sprites

Ideal for particle effects
Much faster than quads
Consume less memory
Easier to set up
Better Texturing

More flexible input
- ETC (Ericsson Texture Compression), JPEG
- RGB565, RGBA5551, RGBA4444
- Can set individual mipmap levels
- Dynamic sources (e.g. video)

Upgraded baseline requirements
- At least two texture units
- At least 1024x1024 size
- Perspective correction
- Mipmapping, bilinear filtering
Texture Combiners

Precursor to fragment shaders
Quite flexible, but not very easy to use
Bump Mapping

- Fake geometric detail
- Feasible even w/o HW
Bump Mapping + Light Mapping

Bump map modulated by projective light map
Floating-Point Vertex Arrays

**float (32-bit)**

- Easy to use, good for prototyping
- Viable with hardware acceleration

**half (16-bit)**

- Savings in file size, memory, bandwidth

**byte/short still likely to be faster**
Triangle Lists

Much easier to set up than strips
  Good for procedural mesh generation
  Avoids the expensive stripification

Sometimes even smaller & faster than strips
  Especially with a cache-friendly triangle ordering
## Primitives – M3G 1.x

<table>
<thead>
<tr>
<th></th>
<th>Byte</th>
<th>Short</th>
<th>Implicit</th>
<th>Strip</th>
<th>Fan</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangles</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Lines</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Points</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Point sprites</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Relative to OpenGL ES 1.1
## Primitives – M3G 2.0

<table>
<thead>
<tr>
<th></th>
<th>Byte</th>
<th>Short</th>
<th>Implicit</th>
<th>Strip</th>
<th>Fan</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangles</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Lines</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Points</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td></td>
<td>✗</td>
</tr>
<tr>
<td>Point sprites</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Relative to OpenGL ES 1.1
### VertexBuffer Types – M3G 1.x

<table>
<thead>
<tr>
<th></th>
<th>Byte</th>
<th>Short</th>
<th>Fixed</th>
<th>Float</th>
<th>2D</th>
<th>3D</th>
<th>4D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertices</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>×</td>
</tr>
<tr>
<td>TexCoords</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Normals</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Colors</td>
<td>✓</td>
<td></td>
<td>×</td>
<td>×</td>
<td></td>
<td>*</td>
<td>✓</td>
</tr>
<tr>
<td>PointSizes</td>
<td></td>
<td></td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* OpenGL ES 1.1 only supports RGBA colors. M3G also supports RGB.
### VertexBuffer Types – M3G 2.0

<table>
<thead>
<tr>
<th></th>
<th>Byte</th>
<th>Short</th>
<th>Fixed</th>
<th>Float Half</th>
<th>2D</th>
<th>3D</th>
<th>4D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertices</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>TexCoords</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Normals</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Colors</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>*</td>
<td>✔</td>
</tr>
<tr>
<td>PointSizes</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* OpenGL ES 1.1 only supports RGBA colors, M3G also supports RGB
Deprecation Features

- Background image
  Use a sky box instead

- Sprite3D
  Use textured quads or point sprites instead

- Flat shading
  Can’t have this on OpenGL ES 2.0!
Deprecated Features Cont’d

Two-sided lighting
   Requires duplicated geometry on OpenGL ES 2.0

Local camera lighting (a.k.a. local viewer)
   Only a hint that was poorly supported

Less accurate picking
   Skinning and morphing not taken into account
M3G 2.0 Preview

Design
Fixed functionality

Programmable shaders

New high-level features

Summary, Q&A
Shading Language

GLSL ES v1.00

Source code format only

Binary shaders would break the Java sandbox

Added a few preprocessor #pragma’s

To enable skinning, morphing, etc.

Apply for vertex shaders only
The Shader Package

Linked on construction, validated on first use

Compiled on construction
Why Multiple ShaderUniforms?

So that uniforms can be grouped

- Global constants – e.g. look-up tables
- Per-mesh constants – e.g. rustiness
- Per-frame constants – e.g. time of day
- Dynamic variables – e.g. position, orientation

Potential benefits of grouping

- Java object reuse – less memory, less garbage
- Can be faster to bind a group of variables to GL
A Fixed-Function Vertex Shader

A small example shader
Replicates the fixed-function pipeline using the predefined #pragma’s
Necessary Declarations

```c
#pragma M3Gvertex(myVertex)
#pragma M3Gnormal(myNormal)
#pragma M3Gtexcoord0(myTexCoord0)
#pragma M3Gcolor(myColor)

#pragma M3Gvertexstage(clipspace)

varying vec2 texcoord0;
varying vec4 color;
```

Names & roles for vertex attributes

Transform all the way to clip space

Variables to pass to the fragment shader
The Shader Code

void main() {
  m3g_ffunction();
  gl_Position = myVertex;
  texcoord0 = myTexCoord0.xy;
  color = myColor;
}

Applies morphing, scale/bias, skinning, model-view, projection

Results passed to the fragment shader
M3G 2.0 Preview

Design
Fixed functionality
Programmable shaders
New high-level features
Summary, Q&A
RenderPass

Automated render-to-texture (RTT)

First set up RenderTarget, World, Camera

Call `myWorld.preRender()` (e.g. every Nth frame)

This updates all dynamic textures in the scene

Finally, render the World as usual

RTT effects can now be authored in DCC tools

Advanced FX without programmer intervention

Reflection, refraction, HDR bloom, etc.
Transparency Sorting

Can sort blended submeshes back-to-front
  Toggled ON/OFF per Appearance and layer
  Based on the Mesh origin’s depth in eye space
  \[ \text{Depth} = \text{MODELVIEW\_MATRIX}(3, 4) \]

Individual triangles are not sorted
Level of Detail (LOD)

A Group node can select one of its children
Based on their size in screen pixels
Similar to mipmap level selection

Formally

Compute \( s = \) pixels per model-space unit
Select the node whose ideal scale \( s_i \) satisfies

\[
\max \{ s_i \mid s_i \leq s \}
\]
Level of Detail (LOD)

Example – from highest detail to lowest:

SkinnedMesh with 30 bones and 1000 vertices
SkinnedMesh with 15 bones and 500 vertices
MorphingMesh with 3 targets and 200 vertices
Tiny billboard with flip-book animation

Appearance detail scaled in the same way

E.g. from complex shaders to per-vertex colors
Bounding Volumes (BV)

To speed up view frustum culling & picking
  Processed hierarchically to cull entire branches

Can be specified for each node
  Bounding sphere = center & radius
  Bounding box = min & max extents
  If both are given, use their intersection
  If neither is given, use an internal BV
Combined Morphing & Skinning

First morph, then skin the result

Useful for animated characters

  Morph targets for facial animation
  Skinning for the rest of the body

Can morph and/or skin any vertex attribute

  Use the result in your own vertex shader

  #pragma M3Gvertexstage(skinned)
Subset Morphing

Can morph an arbitrary subset of vertices

Previously, the whole mesh was morphed

Now the morph targets are much smaller

Big memory savings in e.g. facial animation
Multichannel Keyframe Sequences

$N$ channels per KeyframeSequence

- Same number of keyframes in all channels
- Shared interpolation mode
- Shared time stamps

Huge memory savings with skinning

- M3G 1.1: two Java objects per bone (~60 total)
- M3G 2.0: two Java objects per mesh
Other Stuff

Event tracks associated with animations

E.g. play a sound when a foot hits the ground

Lots of new convenience methods

`findAll(Class)` – find e.g. all Appearances

Can enable/disable animations hierarchically

Can use quaternions instead of axis/angle

Easy pixel-exact 2D projection for overlays

Easy look-at orientation for camera control

Predefined, intuitive blending modes
File Format

Updated to match the new API

File structure remains the same

Same parser can handle both old & new

Better compression for

Textures (ETC, JPEG)

SkinnedMesh, IndexBuffer
**Things Under Consideration**

- Simple collision detection

- Fast Matlab-style array arithmetic
  - Based on floating-point VertexArrays
  - Compute the dot product of two arrays, etc.
  - Overcomes the Java Native Interface overhead
M3G 2.0 Preview

Design
Fixed functionality
Programmable shaders
New high-level features

**Summary, Q&A**
M3G 2.0 will replace 1.1, starting next year
Existing code & assets will continue to work

Several key improvements
Programmable shaders to the mass market
Fully featured traditional rendering pipeline
Advanced animation and scene management
Better perf across all device categories
Q&A

Thanks:
M3G 2.0 Expert Group
Dan Ginsburg (AMD)
Kimmo Roimela (Nokia)
Closing & Summary

We have covered

- OpenGL ES
- M3G

Let’s quickly see what else is there

- COLLADA
- 2D APIs: OpenVG, JSR 226, JSR 287
Khronos API family

Cross-platform graphics authoring/acceleration Ecosystem

Cross platform 2D/3D

Safety Critical 2D/3D

3D Authoring

Dynamic Media Authoring Standards

Dynamic Media Authoring

Embedded Media Acceleration APIs

Streaming Media

Enhanced Audio

2D/3D

Vector 2D

OS portability API

OpenKODE
An open interchange format

to exchange data between content tools
allows mixing and matching tools for the same project
allows using desktop tools for mobile content
Collada conditioning

Conditioning pipelines take authored assets and:

1. Strips out authoring-only information
2. Re-sizes to suit the target platform
3. Compresses and formats binary data for the target platform

Different target platforms can use the same asset database with the appropriate conditioning pipeline
2D Vector Graphics

OpenVG

- low-level API, HW acceleration
- spec draft at SIGGRAPH 05, conformance tests summer 06

JSR 226: 2D vector graphics for Java

- SVG-Tiny compatible features
- completed Mar 05

JSR 287: 2D vector graphics for Java 2.0

- rich media (audio, video) support, streaming
- may still complete in 07
OpenVG features

- Paints
- Mask
- Stroke
- Image transformation
- Fill rule
- Paths
JSR-226 examples

Game, with skins

Scalable maps, variable detail

Cartoon

Weather info
Combining various APIs

It’s not trivial to efficiently combine use of various multimedia APIs in a single application.

EGL is evolving towards simultaneous support of several APIs:
- OpenGL ES and OpenVG now
- all Khronos APIs later
OpenGL ES and OpenVG

**OpenGL ES**
Accurately represents PERSPECTIVE and LIGHTING

**OpenVG**
Accurately represents SHAPE and COLOR

OpenVG ideal for advanced compositing user interfaces
OpenGL ES for powerful 3D UI effects
Summary

Handheld devices are viable 3D platforms
  OpenGL ES, M3G, COLLADA

2D vector graphics is also available
  JSR 226, Flash, OpenVG, JSR 287

Download the SDKs

  and start coding on the smallest (physical size) yet largest (number of units) platform