The Mobile 3D Ecosystem

Kari Pulli  
Nokia Research Center

Jani Vaarala  
Nokia

Ville Miettinen  
NVidia / Hybrid Graphics

Robert Simpson  
AMD

Tomi Aarnio  
Nokia Research Center

Mark Callow  
HI Corporation
Today’s program: Morning

- Start at 8:30
- Intro & OpenGL ES overview
  40 min, Kari Pulli
- Using OpenGL ES 1.x
  50 min, Jani Vaarala
- OpenGL ES on PyS60
  5 min, Kari Pulli
- Demos
  10 min
- Break 10:15 – 10:30
- OpenGL ES performance considerations
  45 min, Ville Miettinen
- OpenGL ES 2.0
  60 min, Robert Simpson
- Break 12:15
Today’s program: Afternoon

- Start at 13:45
- M3G Intro
  10 min, Kari Pulli
- M3G API overview
  60 min, Tomi Aarnio
- M3G in the Real World 1
  25 min, Mark Callow
- Demos
  10 min
- Break 15:30 – 15:45
- M3G in the Real World 2
  60 min, Mark Callow
- M3G 2.0
  30 min, Tomi Aarnio
- Closing & Q&A
  15 min, Kari Pulli
- Finish at 17:30
Evolution of the Computer

Mainframe computer → Mini computer → Personal computer → Multimedia Computer → Laptop 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
Currently 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.
Towards the 3 Billion Milestone

Mobile phone subscriptions globally, millions

Current global penetration 33%

Source: Nokia

3 billion in 2007
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
Challenge? Power!

- 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?

allaboutssymbian.com
Changed?  

Computation!

- 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

GENKI 3D Characters
(C) 2001 GENKI

ULALA
(c)SEGA/UGA.2001

J-SH07 by SHARP

Space Channel 5
©SEGA/UGA,2001 ©SEGA/UGA,2002

Snowboard Rider
©WOW ENTERTAINMENT INC., 2000-2002 all rights reserved.

J-SH51 by SHARP

(c)SEGA/UGA,2001 ©SEGA/UGA,2002
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

Fathammer's Geopod on XForge
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: The Hero Returns
Sony Pictures

Software 3D
Spider-Man 2 3D: NY Subway
Sony Pictures

Accelerated 3D
Spider-Man 2
Activision
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

- Matrix control
  - M
  - T0
  - ... T_n-1
  - P

- Primitive assembly

- Material control
- Lighting

- User clip plane
  - User clip
Primitive processing

1. Eye coordinates
2. Clip coordinates
3. Normalized device coordinates
4. Window coordinates
5. Frustum clip
6. Perspective divide
7. Viewport transform
8. Backface cull
9. User clip
10. User clip plane
11. Rasterization & interpolation
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 GLubyte 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_TRIANGLES, 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 \textbf{IN}, DrawPixels and Bitmap \textbf{OUT}
- Rasterization
  - \textbf{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
The great “Floats vs. fixed-point” debate

• 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
    
```c
    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, good 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)
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(),
    eglCreateXSurface() (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

- **Gerbera from Hybrid**
  - Free for non-commercial use
  - [http://www.hybrid.fi](http://www.hybrid.fi)
- **Vincent**
  - Open-source OpenGL ES library
  - [http://sourceforge.net/projects/ogl-es](http://sourceforge.net/projects/ogl-es)
- **Reference implementation**
  - Wraps on top of OpenGL
  - [http://www.khronos.org/opengles/documentation/gles-1.0c.tgz](http://www.khronos.org/opengles/documentation/gles-1.0c.tgz)
HW implementations

- There are many
  - at least designs, though many are already shipping

- The following slides gives some idea
  - rough rules of thumb
    - 1-5 M Tri / sec
    - 1 pixel / clock
    - clock speeds 50MHz – 200+MHz
    - power consumption should be < 100 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

• 32-bit internal pipe
• 16-bit color and Z buffers

• Imageon 3D (for Qualcomm)
  – 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 Handheld Graphics

1st generation (Imageon 2300)
- OpenGL ES 1.0 (1st conformant implementation)
- Vertex and raster HW
- 32-bit internal pipe, 16-bit color and Z buffers
- Integrated QVGA buffer
- Imaging / Video codecs
- 1 Mtri/s, 100 Mpix/s

2nd generation (Imageon 2380)
- OpenGL ES 1.1
- Vertex shader, HyperZ
- Audio codecs, 3D audio
- 3.5 Mtri/s, 125 Mpix/s

3rd generation (to be announced)
- OpenGL ES 2.0
- Full HW OpenVG 1.1
- Unified Shaders
- OpenGL ES 2.0 and OpenVG cores are also available as IP
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
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
  - Significant reduction in impact of increases scene complexity
- 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.)
Falcon1000

- 2D/3D Graphics and Multimedia for handheld wireless devices
  - OpenGL ES 1.1 & 2.0 and D3D 9.0
  - Complete 2D engine
  - Vertex & pixel shaders
  - AA support
  - Video support
  - 9 MPoly/s (@100MHz)
  - 200MPixel/s (@100MHz)
  - Peak 45MPoly/sec, 1BPixel/sec
  - Power: 2D from 20mw, 3D from 50mw
Imagination Technologies

**PowerVR MBX family**
- OpenGL ES 1.x
- Vertex Geometry Processor (VS 1.1)
- 300M Pix/s (effective) @ 100 MHz
- 3.7M Tri/s (textured & lit) @ 100 MHz

**PowerVR SGX Family**
- OpenGL ES 2.x
- Sizes 2-8 mm² (90nm)
- 100-600M Pix/s (effective) @ 100 MHz
- 1-7M Tri/s (textured & lit) @ 100 MHz
- Vertex, Geometry, Pixel shaders
- Imaging & Video codecs

**Partners / Customers**
- ARM, Intel, Freescale, Philips, Renesas, Samsung, Sunplus, TI

**Tile-based deferred rendering**
- Buffer triangles
- Rasterize a block at a time
- FSAA, high internal color depth
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
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
- 10MPxl 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 vertices)
    - 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
Toshiba

• T5G
  – OpenGL ES 1.1
  – Raster and vertex HW, fixed functionality
  – Large embedded memory for
    • Color and Z buffer
    • Caches for vertex arrays, textures
    • Display lists (command buffer)
  – Cube maps, aniso textures, 2-stage multi tex, …
  – 1.2M vtx / sec, 100M pix / sec (@ 100 MHz)
  – 87 mW @ 100 MHz (incl. eDRAM)

• Partners / Customers
  – Sharp, Toshiba’s own phones, Vodafone
SDKs

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

- Imagination SDK
  - [http://www.pvrdev.com/Pub/MBX](http://www.pvrdev.com/Pub/MBX)

- NVIDIA handheld SDK

- Brew SDK & documentation
  - [http://brew.qualcomm.com](http://brew.qualcomm.com)
OpenGL ES 1.1 Demos

NOKIA presents

TIN STAR

PLAY
DEMO
EXIT

created by housemarque

face tomorrow
Questions?
Today’s program: Morning

• Start at 8:30
• Intro & OpenGL ES overview
  40 min, Kari Pulli
• Using OpenGL ES 1.x
  50 min, Jani Vaarala
• OpenGL ES on PyS60
  5 min, Kari Pulli
• Demos
  10 min

• Break 10:15 – 10:30
• OpenGL ES performance considerations
  45 min, Ville Miettinen
• OpenGL ES 2.0
  60 min, Robert Simpson
• Break 12:15
Using OpenGL ES

- Simple OpenGL ES example
- EGL configuration selection
- Fixed point programming
- Converting existing code
“Hello OpenGL ES”
Hello OpenGL ES::EGL initialization

/* ==============================================================
 * "Hello OpenGL ES" OpenGL ES code.
 * *
 * Siggraph 2007 course on mobile graphics.
 * *
 * Copyright: Jani Vaarala
 * ==============================================================
 */

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

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

```c
EGLint attrib_list[] =
{
    EGL_BUFFER_SIZE, 16,
    EGL_DEPTH_SIZE, 15,
    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  
};
Hello OpenGL ES::OpenGL ES part

```c
void init()
{
    glClearColor ( 0.f, 0.f, 0.1f, 1.f );
    glMatrixMode ( GL_PROJECTION );
    glDisable ( GL_DEPTH_TEST );
    glFrustumf          ( -1.f, 1.f, -1.f, 1.f, 3.f, 1000.f );
    glMatrixMode        ( GL_MODELVIEW );
    glShadeModel        ( GL_SMOOTH );
    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);
}
```
void drawcallback(void)
{
    glClear  ( GL_COLOR_BUFFER_BIT );
    glLoadIdentity ( );
    glTranslatef ( 0.f, 0.f, -5.f );
    glDrawArrays ( GL_TRIANGLES, 0, 3 );

    eglSwapBuffers( display, surface );
}

Hello OpenGL ES::OpenGL ES part
**EGL config sorting**

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>DEFAULT VALUE</th>
<th>SELECTION RULE</th>
<th>SORT ORDER</th>
<th>SORT PRIORITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>EGL_BUFFER_SIZE</td>
<td>0</td>
<td>AtLeast</td>
<td>Smaller</td>
<td>3</td>
</tr>
<tr>
<td>EGL_SAMPLE_BUFFERS</td>
<td>0</td>
<td>AtLeast</td>
<td>Smaller</td>
<td>4</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 EGL config selection

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 with or without 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

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 );
    glShadeModel           ( GL_SMOOTH );
    glClearColor              ( 0.f, 0.f, 0.1f, 1.f );
    glClearColor              ( 0.f, 0.f, 0.1f, 1.f );

    /* Enable vertex arrays */
    glEnableClientState  ( GL_VERTEX_ARRAY );
    glEnableClientState ( GL_TEXTURE_COORD_ARRAY );
Texture matrix example

/* Setup texture */
glEnable ( GL_TEXTURE_2D );

glGenTextures ( 1, texture_handle );

glBindTexture ( GL_TEXTURE_2D, texture_handle );

glTexImage2D ( GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0,
GL_RGB, GL_UNSIGNED_BYTE, texture_data );

glTexEnvi ( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE );

glTexParameteri ( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR );

glTexParameteri ( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR );

glTexParameteri ( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_EDGE );

glTexParameteri ( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE );

}
Texture matrix example

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 );
/* draw the object in front */
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 );

eglMatrixMode ( GL_TEXTURE );
glLoadIdentity ( );
glTranslatef ( 0.5f, 0.5f, 0.f ); /* [-0.5,5] -> [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) */
/* 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…" 
  0xffff ffff = -1/65536(-0x0000 0001)
Fixed point operations

- **Addition**
  
  ```c
  #define add_fixed_fixed(a,b) ((a)+(b))
  ```

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

- **MUL two FP numbers together**
  
  ```c
  #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 = original float values
- S = fixed point scale (e.g., 65536)

result = (a * S) * (b * S) = ((a * b) * S^2)

final = ((a * b) * S^2) / S = (a * b) * 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

\[ \text{VALUE 1} \times \text{VALUE 2} \gg 16 = \text{RESULT} \]

VALUE 1
32-bit

VALUE 2
32-bit

RESULT
48-bit

64-bit
Fixed point programming

- Division of integer by integer to a fixed point result
  
  ```
  #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
  
  ```
  #define div_fixed_int(a,b) ((a)/(b))
  ```

- Division of fixed point by fixed point
  
  ```
  #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 * 65536 = a << 16,  a / 256 = a >> 8
- Fixed point calculations overflow easily
- Careful analysis of the range requirements is required
- Always try to use as low bit ranges as possible
  - 32x8 MUL is faster than 32x32 MUL (some ARM variants)
  - Using unnecessary “extra bits” may slow execution
- Always add debugging code to your fixed point math
Fixed point programming

```c
#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 \((+1.0 \text{ not included !})\)
  - Negative values stored in the table (can represent -1.0)
void calculate_table(short *out)
{
    int i;

    for(i=0; i<2048; i++)
    {
        float angle = (0.5f*PI*i)/2048.0;
        out[i] = -(int)(sin(angle)*32768);
    }
}
inline int fp_sin(int angle)
{
    int ph = angle & (2048 + 4096);  /* phase */
    int ang = angle & 2047;          /* sub-angle */

    /* return negated values (was stored negated) */
    if(ph == 0) return -((int)table[ang]);
    else if(ph == 2048) return -((int)table[2048-ang]);
    else if(ph == 4096) return  (int)table[ang];
    else return  (int)table[2048-ang];
}
How to use fp_sin()

```c
void do_something(int ang)
{
    int i;

    for( i=0; i<1000; i++)
    {
        int tmp;
        tmp = (vin[i*3] * fp_sin(ang)) >> 16;
        vout[i*3] = tmp;
    }
}
```

- note: fp_sin returns integers
  => it can also return 32768 (1.0)

- it does not fit inside s0.16 fixed point number!
Performance

- \texttt{fp\_sin()} is rather complex
- Simple optimization: calculate 360 degrees
- Downside: takes more memory
- And: to handle 1.0 we have to use \( S = 16384 \)
void calculate_table(short *out)
{
    int i;

    for(i=0; i<2048*4; i++)
    {
        float angle = (2.f*PI*i)/2048.0;
        out[i] = (int)(sin(angle)*16384
    }
}
inline int fp_sin(int angle) {
    return ((int)table[angle & 8191]);
}
Example: Simple morphing (LERP)

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

```c
#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
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/glEnd 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 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
Questions?
Today’s program: Morning

- Start at 8:30
- Intro & OpenGL ES overview
  40 min, Kari Pulli
- Using OpenGL ES 1.x
  50 min, Jani Vaarala
- OpenGL ES on PyS60
  5 min, Kari Pulli
- Demos
  10 min

- Break 10:15 – 10:30
- OpenGL ES performance considerations
  45 min, Ville Miettinen
- OpenGL ES 2.0
  60 min, Robert Simpson
- Break 12:15
OpenGL ES on PyS60

Kari Pulli

Nokia Research Center
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):
    self.exiting = False
    self.frame, self.angle = 0, 0
    self.old_body = appuifw.app.body
    try:
        c = GLCanvas( redraw_callback = self.appcycle,
                      resize_callback = self.appresize )
        appuifw.app.body = c
    except Exception, e:
        appuifw.note( u"Exception: %s" % (e) )
        self.start_exit()
    return
    appuifw.app.menu = [(u"Exit", self.start_exit)]
    self.initgl()
    c.bind( EKeyLeftArrow, lambda:self.left() )
    c.bind( EKeyRightArrow, lambda:self.right() )
    self.canvas = c
def left(self):
    self.angle -= 10

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

def appresize(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
Initialize OpenGL ES

def initgl(self):
    glMatrixMode( GL_PROJECTION )
    glFrustumf( -1.0, 1.0, -1.0, 1.0, 3.0, 1000.0 )
    glMatrixMode( GL_MODELVIEW )
    glDisable( 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 appcycle(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
```
Today’s program: Morning

- Start at 8:30
- Intro & OpenGL ES overview
  40 min, Kari Pulli
- Using OpenGL ES 1.x
  50 min, Jani Vaarala
- OpenGL ES on PyS60
  5 min, Kari Pulli
- Demos
  10 min

- Break 10:15 – 10:30
- OpenGL ES performance considerations
  45 min, Ville Miettinen
- OpenGL ES 2.0
  60 min, Robert Simpson
- Break 12:15
High-performance OpenGL ES 1.x

Ville Miettinen

NVIDIA
Targeting the "mobile platform"

- CPU speed and available memory varies
  - Current range ~30Mhz - 600MHz, ARM7 to ARM11, no FPUs
- Different resolutions
  - QCIF (176x144) to VGA (640x480), 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 game play 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 game play 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
  – Vertex-based:
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
Today’s program: Morning

- Start at 8:30
- Intro & OpenGL ES overview
  - 40 min, Kari Pulli
- Using OpenGL ES 1.x
  - 50 min, Jani Vaarala
- OpenGL ES on PyS60
  - 5 min, Kari Pulli
- Demos
  - 10 min
- Break 10:15 – 10:30
- OpenGL ES performance considerations
  - 45 min, Ville Miettinen
- OpenGL ES 2.0
  - 60 min, Robert Simpson
- Break 12:15
ES 2.0 Overview

Robert J. Simpson
AMD
Contents

• The OpenGL ES 2.0 Pipeline
• API Overview
• GLSL ES 1.00 Overview
• Writing an ES 2.0 Application
• Examples
• Future Directions
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
Open GL Fixed Function pipeline

API

Primitive Processing

Transform and Lighting

Primitive Assembly

Rasterizer

Texture Environment

Colour Sum

Fog

Alpha Test

Depth Stencil

Colour Buffer Blend

Dither

Frame Buffer

Triangles/Lines/Points

Vertices
Open GL Fixed Function pipeline

- API
- Primitive Processing
- Vertex Buffer Objects
- Vertices
- Triangles/Lines/Points
- Vertex Shader
- Primitive Assembly
- Rasterizer
- Fragment Shader
- Depth Stencil
- Colour Buffer Blend
- Dither
- Frame Buffer
Programmer’s model

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

- 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 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.
  - Restrictions on shader complexity
  - Fewer sampler modes
GLSL ES Overview

• Comments
  //
  /*   */

• Control
  #if
  #ifdef
  #ifndef
  #else
  #elif
  #endif
  #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.

```cpp
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++;  // a is incremented
    b++;  // b is incremented
}

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

- Order of copy back is undefined

```cpp
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

• mediuimp 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 ± $2^{62}$
  - 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
Precision Qualifiers

• Precision depends on the operands:
  ```
  lowp float x;
  mediump float y;
  highp float z = x * y;
  ```
  (evaluated at medium precision)

• Literals do not have any defined precision
  ```
  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:
  ```java
  int n = 1;
  float x,y;
  x = float(n);
  y = float(2);
  ```
Constructors

• Concatenation:
  ```
  float x = 1.0, y = 2.0;
  vec2 v = vec2(x, y);
  ```

• Structure initialization
  ```
  struct S { int a; float b; };
  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

- Indexing operator
  ```cpp
  vec4 u,v;
  float x = u[0]; // equivalent to u.x
  ```

- Must use indexing operator for matrices
  ```cpp
  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
  ```
  vec4 gl_Position; // Write-only
  float gl_PointSize; // Write-only
  ```

- Fragment shader
  ```
  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
  \[ \text{mix}(x, y, \alpha) = x \times (1.0 - \alpha) + y \times \alpha \]
  \[ \text{step}(\text{edge}, x) = \begin{cases} 0.0 & \text{if } x \leq \text{edge} \\ 1.0 & \text{otherwise} \end{cases} \]
  \[ \text{smoothstep}(\text{edge}_0, \text{edge}_1, x) = \frac{t \times (3.0 - 2.0 \times t)}{t = \frac{x - \text{edge}_0}{\text{edge}_1 - \text{edge}_0}; t = \text{clamp}(t, 0.0, 1.0); \text{return } t \times t \times (3.0 - 2.0 \times t) \]

• Texture
  \[ \text{texture1D}, \text{texture2D}, \text{texture3D}, \text{textureCube} \]
  \[ \text{texture1DProj}, \text{texture2DProj}, \text{textureCubeProj} \]
GLSL ES Overview

- **Vector comparison** \((\text{vec}n, \text{ivec}n)\)
  - \(\text{bvec}n\) lessThan\((\text{vec}n, \text{vec}n)\)
  - \(\text{bvec}n\) lessThanEqual\((\text{vec}n, \text{vec}n)\)
  - \(\text{bvec}n\) greaterThan\((\text{vec}n, \text{vec}n)\)
  - \(\text{bvec}n\) greaterThanEqual\((\text{vec}n, \text{vec}n)\)

- **Vector comparison** \((\text{vec}n, \text{ivec}n, \text{bvec}n)\)
  - \(\text{bvec}n\) equal\((\text{vec}n, \text{vec}n)\)
  - \(\text{bvec}n\) notEqual\((\text{vec}n, \text{vec}n)\)

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

- **Matrix**
  - matrixCompMult \((\text{mat}n, \text{mat}n)\)
Invariance: The Problem

• Mathematical operations are not precisely defined
• Same code may produce (slightly) different results

Two cases to consider:
  - Invariance within a shader
  - Invariance between shaders
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.

\[
\begin{align*}
\text{MUL } & \text{R1, a, x} \\
\text{MUL } & \text{R2, b, y} \\
\text{MUL } & \text{R3, c, z} \\
\text{MUL } & \text{R4, d, w} \\
\text{ADD } & \text{R1, R1, R2} \\
\text{ADD } & \text{R3, R3, R4} \\
\text{ADD } & \text{R1, R1, R3}
\end{align*}
\]

or

\[
\begin{align*}
\text{MUL } & \text{R1, a, x} \\
\text{MADD } & \text{R1, b, y} \\
\text{MADD } & \text{R1, c, z} \\
\text{MADD } & \text{R1, d, w}
\end{align*}
\]
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 it is cheaper to recalculate a value rather than store it in a register.
    But will it be calculated the same way?

E.g.

```cpp
uniform sampler2D tex1, tex2;
...
const vec2 pos = ...;
vec4 col1 = texture2D(tex1, pos);
...
vec4 col2 = texture2D(tex2, pos); // is this
    // the same
    // value?

gl_FragColor = col1 - col2;
```
Invariance

- Solution is in two parts:
  - invariant keyword specifies that specific variables are invariant e.g.

```cpp
invariant varying vec3 LightPosition;
```

Currently can only be used on certain variables

- Global switch to make all variable invariant

```cpp
#pragma STDGL invariant(all)
```
Invariance

• Invariance flag controls:
  – Invariance within shaders and
  – 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.
Writing an application - Overview

- Initialize EGL
- Setup shader, pipeline state
- Create vertex buffers, textures
- Bind buffers
- Draw
Writing an App – Initialization

- Set up EGL
- 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

```
EGLint attr[MAX_EGL_ATTRIBUOTES];
attr[nAttrib++] = EGL_RED_SIZE;
attr[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, 1, &num_configs)

eglCreateWindowSurface(egl_display, egl_config, NativeWindowType (hWnd), NULL)
context = eglCreateContext(egl_display, egl_config, EGL_NO_CONTEXT, NULL);

eglMakeCurrent(egl_display, egl_surface, egl_surface, egl_context);
Compiling and using shaders

Vertex Shader

- glCreateShaderObject
- glShaderSource
- glCompileShader
- glDeleteObject

Fragment Shader

- glCreateShaderObject
- glShaderSource
- glCompileShader
- glDeleteObject
• Create objects

```c
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);
```
• Compile vertex shader (and fragment shader)

```c
char* vert_source = ...
const char* vert_gls[1] = {vert_source};
glShaderSource(vertex_shader_handle, 1, 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 = glGetUniformLocation (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, &envmapId);

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`:

  ```
  glUseProgram(prog_handle);
  ```

- Use `glGetUniformLocation` e.g.

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

- Can then set value e.g.

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

- Create buffer names
  
  ```c
  GLuint bufs[1];
  glGenBuffers (1, arrayBufs);
  ```

- Create and initialize buffer
  
  ```c
  glBindBuffer (GL_ARRAY_BUFFER, bufs[0]);
  glBufferData (GL_ARRAY_BUFFER, size_bytes, p_data, GL_STATIC_DRAW);
  ```
Setting up Attribute Buffers (cont)

- Specify format:

```c
glBindBuffer(GL_ARRAY_BUFFER, bufs[0]);
glVertexAttribPointer(0,       // index
                      4,       // size
                      GL_FLOAT,// type
                      GL_FALSE,
                      0,        // size
                      NULL );  // pointer
```
Drawing the frame

- Set render state
- Clear frame buffer
- Bind vertex buffers
- Specify type and layout (glVertexAttribPointer)
- Enable array
- DrawArray
Enable array and Draw

```c
glEnableVertexAttribArray( 0 );

glBindBuffer (GL_ELEMENT_ARRAY_BUFFER, 0);

glDrawArrays (GL_TRIANGLE_STRIP, 0, n_vertices);
```
Example: Water demo
Skybox

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

- Can use position to access cube map
- Don’t need to normalize.
- No need for 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;
}
precision highp float;
uniform vec4 scale;
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
Water

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

- Use noise texture for bump map.
- Exact texture not important
  - Try experimenting
Water: Vertex 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;
varing vec2 vTexCoord;
varing vec3 vNormal;
varing 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;
// 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);
Water Vertex Shader (cont)

```cpp
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.
    
    ```
    assert(!glError ());
    ```

- But remember `glError()` gets the last error:
  
  ```
  ... // 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
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
The projection matrix

- You need to provide a projection matrix e.g.

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

- 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.
Future Directions

• Sample Shaders
  – Enables alpha testing at per-sample resolution
  – Enables more of the fixed function pipeline to be removed.
  – Allows more programmability when using multi-sampling.
  – e.g. Read and write depth and stencil
Future Directions

• Object (Geometry) Shaders
  – Programmable tessellation
  – Higher order surfaces
  – Procedural geometry
  – Possibility of accelerating many more algorithms
e.g. shadows, occlusion culling.
Future ES Pipeline?
Today’s program: Afternoon

• Start at 13:45
• M3G Intro
  10 min, Kari Pulli
• M3G API overview
  60 min, Tomi Aarnio
• M3G in the Real World 1
  25 min, Mark Callow
• Demos
  10 min
• Break 15:30 – 15:45
• M3G in the Real World 2
  60 min, Mark Callow
• M3G 2.0
  30 min, Tomi Aarnio
• Closing & Q&A
  15 min, Kari Pulli
• Finish at 17:30
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
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.
Why?

- Array bounds checking
- Strong type checking
- No stack allocation
- Garbage collection
- Slow or non-existent Java Native Interface
- No access to special CPU instructions
- 100% hardware acceleration not feasible
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
M3G Design Principles

#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.
Today’s program: Afternoon

- Start at 13:45
- M3G Intro
  10 min, Kari Pulli
- M3G API overview
  60 min, Tomi Aarnio
- M3G in the Real World 1
  25 min, Mark Callow
- Demos
  10 min

- Break 15:30 – 15:45
- M3G in the Real World 2
  60 min, Mark Callow
- M3G 2.0
  30 min, Tomi Aarnio
- Closing & Q&A
  15 min, Kari Pulli
- Finish at 17:30
M3G Overview

Tomi Aarnio
Nokia Research Center
M3G Overview

Getting started
Basic features
Performance tips
Deforming meshes
Keyframe animation
Summary & demos
The Programming Model

- 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
The 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
Main Classes

- Graphics3D
  - 3D graphics context
  - Performs all rendering

- Loader
  - Loads individual objects and entire scene graphs (.m3g, .png and .jpg files)

- World
  - Scene graph root node
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
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();
}
```
M3G Overview

Getting started

**Basic features**

Performance tips

Deforming meshes

Keyframe animation

Summary & 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>

* OpenGL ES only supports RGBA colors
**Vertex and Index Buffer Objects**

- Vertices and indices are stored on server side
  - Very similar to OpenGL Buffer Objects
  - Avoids moving the same data across JNI every time
  - Also allows caching, bounding box computation, etc.

- Tradeoff – Dynamic updates have overhead
  - At the minimum, copying in the Java array contents
  - May also trigger vertex preprocessing
Appearance Components

Material
- Material colors for lighting
- Can track per-vertex colors

CompositingMode
- Blending, depth buffering
- Alpha testing, color masking

PolygonMode
- Winding, culling, shading
- Perspective correction hint

Fog
- Fades colors based on distance
- Linear and exponential mode

Texture2D
- Texture matrix, blending, filtering
- Multitexturing: One Texture2D for each unit
The Scene Graph

```
World
  ↓
Group
  ↓
Group
  ↓
MorphingMesh
  ↓
Group
  ↓
Mesh
  ↓
Sprite
```

Not allowed!
Node Transformations

- From this node to the parent node
- Composed of four parts
  - Translation $T$
  - Orientation $R$
  - Non-uniform scale $S$
  - Generic 3x4 matrix $M$
- $C = T R S M$
Other Node Features

• Automatic alignment
  – Aligns the node’s Z and/or Y axes towards a target
  – Recomputes the orientation component (R)

• Inherited properties
  – Alpha factor (for fading in/out)
  – Rendering enable (on/off)
  – Picking enable (on/off)

• Scope mask
The File Format

Characteristics

- Individual objects, entire scene graphs, anything in between
- Object types match 1:1 with those in the API
- Optional ZLIB compression of selected sections
- Can be decoded in one pass – no forward references
- Can reference external files or URIs (e.g. textures)
- Strong error checking
M3G Overview

- Getting started
- Basic features
- **Performance tips**
- Deforming meshes
- Keyframe animation
- Summary & demos
Retained Mode

- Use the retained mode
  - Do not render separate objects – use a World
  - Minimizes Java code and method calls
  - Allows view frustum culling, etc.

- Keep Node properties simple
  - Favor the T R S components over M
  - Avoid non-uniform scales in S
  - Avoid using the alpha factor
Rendering Order

• Layers define global ordering for submeshes
  – Within each layer, opaque objects come first
  – Transparent objects are not sorted

• Use layers for…
  – Multipass rendering
  – Making sure that overlays are drawn first
  – Making sure that distant objects are drawn last
Textures

- Multitexturing is faster than multipass
  - Transformation and setup costs cut by half
- Use mipmaps to save memory bandwidth
- Combine small textures into a texture atlas
- Use the perspective correction hint
  - Much faster than increasing triangle count
  - Nokia: 2% fixed overhead, 20% in the worst case
Meshes

• Minimize the number of objects
  – Per-mesh overhead is high
  – Per-submesh also fairly high
  – Ideally, everything would be in one giant tri-strip
  – But then view frustum culling doesn’t work → bad

• Strike a balance
  – Merge simple meshes that are close to each other
  – Criteria for “simple” and “close” will vary by device
Shading State

- **Software vs. hardware implementations**
  - SW: Minimize per-pixel operations
  - HW: Minimize shading state changes
  - SW: Mix 2D and 3D where necessary
  - HW: Do not mix 2D and 3D rendering

- **OpenGL ES performance tips apply**
Particle Effects

Several problems
- Point sprites are not supported
- Sprite3D has too much overhead

Put all particles in one Mesh
- One particle == two triangles
- All glued into one triangle strip
- Update vertices to animate
  - XYZ, RGBA, maybe UV

Use additive alpha blend and per-vertex colors
Triangle strip starts here
Particles glued into one tri-strip using degenerate triangles
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
- Quadtree leaf node == Mesh object
- Quadtree inner node == Group object
- Enable nodes yourself, based on the view frustum
M3G Overview

- Getting started
- Basic features
- Performance tips
- Deforming meshes
- Keyframe animation
- Summary & 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
SkinnedMesh

- 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

shared vertex, weights = (0.5, 0.5)

"skin"

non-shared vertex

Bone A

Bone B
SkinnedMesh

Bone B rotated 90 degrees

position in A's coordinate system

interpolated position

position in B's coordinate system
SkinnedMesh

No skinning

Smooth skinning
two bones per vertex
M3G Overview

- Getting started
- Basic features
- Performance tips
- Deforming meshes
- Keyframe animation
- Summary & demos
Animation Classes

- **KeyframeSequence**: Storage for keyframes. Defines interpolation mode.
- **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

AnimationController

KeyframeSequence

Identifies animated property on this object
Keyframe is a time and the value of a property at that time
Can store any number of keyframes
Several keyframe interpolation modes
Can be open or closed (looping)

Diagram courtesy of Sean Ellis, Superscape
KeyframeSequence

Keyframe is a time and the value of a property at that time
Can store any number of keyframes
Several keyframe interpolation modes
Can be open or closed (looping)

Diagram courtesy of Sean Ellis, Superscape
Keyframe is a time and the value of a property at that time
Can store any number of keyframes
Several keyframe interpolation modes
Can be open or closed (looping)
AnimationController

Can control several animation sequences together
Defines a linear mapping from world time to sequence time
Multiple controllers can target the same property

Diagram courtesy of Sean Ellis, Superscape
Animation

1. Call `animate(worldTime)`

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, Superscape
Tip: Interpolate quaternions as ordinary 4-vectors
- Supported in the latest M3G Exporter from HI Corp
- SLERP and SQUAD are slower, but need less keyframes
- Quaternions are automatically normalized before use
M3G Overview

Getting started
Basic features
Performance tips
Deforming meshes
Keyframe animation
Summary & demos
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 – Mr. Goodliving

2D

Restart

Restart

Restart

Restart

3D

Restart

Restart

Restart

Restart
Playman World Soccer – Mr. Goodliving

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

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

- 3D with 2D background and overlays
- Skinning used for characters
- Realistic ball physics
Thanks: Sean Ellis, Kimmo Roimela, Nokia M3G team, JSR-184 Expert Group, Mr. Goodliving (RealNetworks), Sumea (Digital Chocolate)
Today’s program: Afternoon

• Start at 13:45
• M3G Intro
  10 min, Kari Pulli
• M3G API overview
  60 min, Tomi Aarnio
• M3G in the Real World 1
  25 min, Mark Callow
• Demos
  10 min
• Break 15:30 – 15:45
• M3G in the Real World 2
  60 min, Mark Callow
• M3G 2.0
  30 min, Tomi Aarnio
• Closing & Q&A
  15 min, Kari Pulli
• Finish at 17:30
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
M3G Game Demo

Copyright 2005, Digital Chocolate Inc.
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

Screen Image: Boulder Dash®-M.E.™

Diagram courtesy of Sean Ellis, ARM.
M3G Game Development Process

- How M3G Fits

  **Compile**

  **Java MIDlet**

  **Package**

  **JAR file**

**Assets**
- Images
- Sounds
- Music
- **3D World**

**Expanded game logic**

**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

Assets
- Images
- Sounds
- Music

Expanded game logic

Compile

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

3D Graphics
Creating Your Assets: Sounds

- Audio Tools
  - Expanded game logic
  - Compile
  - Assets
  - Images
  - Music
  - Sounds

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

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

3D Graphics
Creating Your Assets: Music

- **Music Tools**

  - **Expanded game logic**

  - **Compile**

  - **Java MIDlet**

  - **Package JAR file**

- **Assets**
  - Images
  - Sounds
  - 3D World

- **Music**
  - MIDI Sequencer; e.g.
    - Steinberg Cubase
  - Formats:
    - SMAF, MIDI, cMIDI, MFi

- **Proprietary**
  - 3D Graphics
Creating Your Assets: 3d Models

- Modeling Tools

Assets
- Images
- Sounds
- Music

3D World

3d Modeler with M3G plug-in; e.g.
- Lightwave
- Maya
- 3d studio max
- Softimage|XSI
Demo: On a Real Phone
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 the 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

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

Assets
- Images
- Sounds
- Music
- 3D World

Compiled game logic

Java MIDlet

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

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

Screen Image: Sega/Wow Entertainment RealTennis,™
Java Wireless Toolkit 2.5 for CLDC

KToolBar

Handset Emulator
NetBeans + Mobility Pack
Agenda

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

- Derived from MIDlet,
- Overrides three methods

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

\[ \text{MIDlet}\text{.destroyApp()} \]
  [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

Update loop. `Runnable.run()`

Read user input, update scene

Exit request

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

Get any user input via `Canvas.commandListener`

Game logic, animate, align if necessary

`Canvas.paint()`
performs rendering using `Graphics3D` object

`MIDlet.destroyApp()`
Tidy up; exit MIDlet

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
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**

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

**user input**
- Get any user input, network play, etc.

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

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

**draw**
- Graphics3D object performs rendering

**wait**
- Release assets, tidy up

**shut down**
- Exit request

*Diagram courtesy of Sean Ellis, ARM*

---

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

---

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

---

*Diagram courtesy of Sean Ellis, ARM*
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
Demo: Using M3G MIDlet
Using M3G MIDlet

- Displays Mesh, MorphingMesh and SkinnedMesh
- Loads data from .m3g files
- View can be changed with arrow keys
- Animation can be stopped and started
- Animation of individual meshes can be stopped and started.
- Displays frames per second.
import java.io.IOException;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class Cans extends MIDlet implements CommandListener {
    Command cmdExit = new Command("Exit", Command.SCREEN, 1);
    Command cmdPlayPause = new Command("Ctrl", Command.SCREEN, 1);
    private TargetCanvas tcanvas = null;
    Thread renderingT = null;
    private String Filename = "/coffee.m3g";

    public void startApp() {
        if (tcanvas == null)
            init();

        renderingT = new Thread(tcanvas);
        renderingT.start();
        tcanvas.startPlay();
    }
}
public void pauseApp() {
    if (tcanvas.isPlaying)
        tcanvas.pausePlay();
    renderingT.yield();
    renderingT = null;
}

public void destroyApp(boolean u) {
    pauseApp()
    tcanvas = null;
}
synchronized public void commandAction(Command c, Displayable d)
{
    if (c==cmdExit) {
        notifyDestroyed();
        return;
    } else if (c==cmdPlayPause) {
        if (tcanvas.isPlaying)
            tcanvas.pausePlay();
        else
            tcanvas.startPlay();
    }
}
// From class Cans
public void init() {
    Display disp = Display.getDisplay(this);
    tcanvas = new TargetCanvas(Filename);
    if (tcanvas.hasException)
        notifyDestroyed();
    tcanvas.setCommandListener(this);
    tcanvas.addCommand(cmdExit);
    tcanvas.addCommand(cmdPlayPause);
    disp.setCurrent(tcanvas);
}

UsingM3G Initialization

class TargetCanvas extends Canvas implements Runnable
    ... // instance variable declarations elided
public TargetCanvas(String m3gFile)
{
    try
    {
        fileName = m3gFile;
        g3d = Graphics3D.getInstance();
        Load();
        w = getWidth();
        h = getHeight();
        cameraManip = new CameraManip(gWorld);
    }
    catch(IOException e)
    {
        System.out.println("loading fails:"+fileName);
        hasException = true;
    }
}
// class TargetCanvas
void Load() throws IOException {
    loadObjs = Loader.load(fileName);
    if (loadObjs==null)
        throw new RuntimeException("M3g m3g file error");

    /* find the world node */
    for (int i=0; i<loadObjs.length; ++i) {
        if (loadObjs[i] instanceof World) {
            gWorld = (World)loadObjs[i];
            hasWorld = true;
            break;
        }
    }

    if (!hasWorld)
        throw new RuntimeException(
            "World node not found; incorrect m3g file?");
Loading the 3D Data (Cont.)

```java
meshController = 
  (AnimationController)gWorld.find(meshControllerId);
morphingMeshController = 
  (AnimationController)gWorld.find(morphingMeshControllerId);
skinnedMeshController = 
  (AnimationController)gWorld.find(skinnedMeshControllerId);

/* Clean up after the loading process. */
System.gc();
```
public void run()
{
    for(;;) {
        long start, elapsed;
        start = System.currentTimeMillis();
        handleInput();
        repaint(); // Request paint()
        elapsed = System.currentTimeMillis() - start;
        // if (want to measure true frame rate)
        // Unfriendly to system!!
        renderTime += (int)elapsed;
        // else {
        renderTime += (elapsed < 50) ? 50 : (int)elapsed;
        try {
            if (elapsed < 50) Thread.sleep(50-elapsed);
        } catch (InterruptedException e) { }
        //}
    }
}
TargetCanvas *paint* method

```java
synchronized protected void paint(Graphics g) {
    if(loadObjs== null) return;
    g.setClip(0, 0, w, h);
    try {
        g3d.bindTarget(g);
        g3d.setViewport(0, 0, w, h);
        render();
    } finally { g3d.releaseTarget(); }
    g.setColor(0xffffffff);
    g.drawString("fps: " + fps, 2, 2, g.TOP|g.LEFT);
}
```
void render()
{
    if (isPlaying) {
        frameCount++;
        fps = (int)((1000*frameCount) / renderTime) ;
        /* update the scene */
        gWorld.animate((int)renderTime);
    }
    g3d.render(gWorld);
}
Camera Manipulation

/**
 * A camera manipulator. This class applies rotations to a World's activeCamera that make it rotate around the prime axes passing through the World's origin.
 */

public class CameraManip
{
    public CameraManip(World world) { }

    public void buildCameraXform() { }

    public void baseRotate(float dAngleX, float dAngleY, float dAngleZ){ }

    public void rotate(float dAngleX, float dAngleY, float dAngleZ) { }

    public void setCameraXform() { }
}

public CameraManip(World world) {
    Transform world2Cam = new Transform();
    float[] matrix = new float[16];
    /* ... class variable initialization elided */
    
    curCamera = world.getActiveCamera();
    if (curCamera != null) {
        curCamera.getTransformTo( world, world2Cam );
        world2Cam.get( matrix );

        curCamera.getTransform( curOriginalXform );
        rotate( 0, 0, 0 );
        world2Cam = null;
    }
}
public void rotate(float dAngleX, float dAngleY, 
    float dAngleZ) {
    if (curCamera == null) return;

    baseRotate( dAngleX, dAngleY, dAngleZ );
    Transform rotTrans = new Transform();

    rotTrans.postRotate( angleY, 0, 1, 0 );
    rotTrans.postRotate( angleX, 1, 0, 0 );

    float pos[] = { 0, 0, distToTarget, 1 }; 
    rotTrans.transform( pos );
    dx = pos[0];
    dy = pos[1];
    dz = pos[2] - distToTarget;

    buildCameraXform();
    setCameraXform();
    rotTrans = null;
}
Building the Camera Transform

```java
public void buildCameraXform() {
    cameraXform.setIdentity();
    rotateXform.setIdentity();
    transXform.setIdentity();

    transXform.postTranslate(dx, dy, dz);

    // rotate about the x-axis then the y-axis
    rotateXform.postRotate(angleY, 0, 1, 0);
    rotateXform.postRotate(angleX, 1, 0, 0);

    cameraXform.postMultiply(transXform);
    cameraXform.postMultiply(rotateXform);
}

public void setCameraXform() {
    cameraXform.postMultiply(curOriginalXform);
    curCamera.setTransform(cameraXform);
}
```
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
• Device Fragmentation
• 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 budget, typically $100k – 250k
Memory

• Problems
  ① Small application/download size
  ② Small heap memory size

• Solutions
  – Compress data ①
  – Use single large file ①
  – Use separately downloadable levels ①
  – Limit contents ②
  – Java Optimizations: combine classes, coalesce var’s, eliminate temporary & local variables, … ②
Performance

• Problems
  ① Slow system bus & memory
  ② No integer divide hardware

• Solutions
  – Use smaller textures ①
  – Use mipmapping ①
  – Use byte or short coordinates and key values ①
  – Use shifts ②
  – Let the compiler do it ②
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 it 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

- Java has 4 security domains:
  - Manufacturer
  - Operator
  - 3rd Party
  - Untrusted

- Most phones will not install untrusted MIDlets
  - If unsigned 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

- 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 it published and distributed
Publishing Your Content: Distribution Channels

- **Game Deck**
  - e.g. “More Games button”

- **Off deck, in portal**
  - e.g. Cingular’s *Beyond MEdia Net*

- **Off deck, 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
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 2005, by Interactive Brains, Co., Ltd.
Summary

- Use standard tools to create assets
- Basic M3G MIDlet is relatively easy
- Programming 3D Games for mobile is hard
- Need good relations with operators and publishers to get your content distributed
Exporters

3ds max
- Simple built-in exporter since 7.0
- www.digi-element.com/Export184/
- www.mascotcapsule.com/M3G/
- www.m3gexporter.com

Maya
- www.mascotcapsule.com/M3G/
- www.m3gexport.com

Softimage|XSI
- www.mascotcapsule.com/M3G/

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

Lightwave
- www.mascotcapsule.com/M3G/

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

Not a typo vapourware?
SDKs

- Motorola iDEN J2ME SDK
  - idenphones.motorola.com/iden/developer/developer_tools.jsp
- Nokia Series 40, Series 60 & J2ME
  - www.forum.nokia.com/java
- Sony Ericsson
  - developer.sonyericsson.com/java
- Sprint Wireless Toolkit for Java
  - developer.sprintpcs.com
- Sun Java Wireless Toolkit 2.5 for CLDC
SDKs

• VFX SDK (Vodafone Global)
  – via.vodafone.com/vodafone/via/Home.do

• VFX & WTKforJSCL (Softbank KK)
  – developers.vodafone.jp/dp/tool_dl/java/emu.php
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
- Adobe Photoshop
- Sony SoundForge
- Steinberg Cubase
  - [www.steinberg.de/33_1.html](http://www.steinberg.de/33_1.html)
- Yamaha SMAF Tools
  - [smaf-yamaha.com/](http://smaf-yamaha.com/)
犬友 (Dear Dog) Demo
Thanks: HI Mascot Capsule Version 4 Development Team, Koichi Hatakeyama, Sean Ellis, JSR-184 Expert Group
Today’s program: Afternoon

- Start at 13:45
- M3G Intro
  10 min, Kari Pulli
- M3G API overview
  60 min, Tomi Aarnio
- M3G in the Real World 1
  25 min, Mark Callow
- Demos
  10 min

- Break 15:30 – 15:45
- M3G in the Real World 2
  60 min, Mark Callow
- M3G 2.0
  30 min, Tomi Aarnio
- Closing & Q&A
  15 min, Kari Pulli
- Finish at 17:30
M3G 2.0
Sneak Peek

Tomi Aarnio
Specification Lead
Nokia Research Center
Agenda

- Design & Structure
- Fixed functionality
- Shaders
- Scene Graph
- Animation
Mobile 3D Graphics 2.0 (JSR 297)

- Successor to M3G 1.1 (JSR 184)
  - Work started mid-2006
  - Ratification expected Q1 2008
  - First devices maybe Q4 2008

- All information here is preliminary!
  - Represents the status as of April 2007
  - Slides will be updated for SIGGRAPH
Expert Group

- AMD, ARM, NVIDIA, PowerVR
- Nokia, Motorola, Samsung, Sony Ericsson
- Digital Chocolate, Musiwave, RealNetworks
- Acrodea, Aplix, HI, Sun, Superscape
+ A few private members
## Design Goals & Priorities

<table>
<thead>
<tr>
<th>Target all devices</th>
<th>Backwards compatible</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Programmable</td>
<td>1. Art assets (.m3g)</td>
</tr>
<tr>
<td>2. Software-only</td>
<td>2. Source code (.java)</td>
</tr>
<tr>
<td>3. Fixed-function</td>
<td>3. Binary code (.class)</td>
</tr>
</tbody>
</table>
Fully Backwards Compatible

M3G 1.1

M3G 2.0 Core

M3G 2.0 Advanced
M3G 2.0
Advanced

M3G 2.0 Core

M3G 1.1

Fully Backwards Compatible

Mandatory

Optional
Shaders and Fixed Functionality

M3G 2.0

OpenGL ES 1.1

OpenGL ES 2.0
Shaders and Fixed Functionality

- M3G 2.0
- OpenGL ES 2.0
- OpenGL ES 1.1

Mandatory

Optional
Why Not Shaders Only?

- No Graphics Hardware
- Fixed Function Hardware
- Shader Hardware

Device sales in 2010?
Serving the Low End…

- Basic Content
- M3G Core
- OpenGL ES 1.1
- CPU
### the Mid Category

<table>
<thead>
<tr>
<th>Basic Content</th>
<th>Enhanced Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>M3G Core</td>
<td></td>
</tr>
<tr>
<td>OpenGL ES 1.1</td>
<td></td>
</tr>
<tr>
<td>Fixed Function Graphics Hardware</td>
<td></td>
</tr>
</tbody>
</table>
...and the High End

<table>
<thead>
<tr>
<th>Basic</th>
<th>Enhanced</th>
<th>Premium Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>M3G Core</td>
<td>M3G Advanced</td>
<td></td>
</tr>
<tr>
<td>OpenGL ES 2.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programmable Graphics Hardware</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Downsides

- Must support fixed functionality on ES 2.0
  - Extra implementation burden

- The API is not as compact as it used to be
  - 15-20 new classes (previously had 30)

- Can only deprecate features, not remove
  - Sprite3D, Background image
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
Packages

- `javax.microedition.m3g`
  - Contains the entire Core Block
  - Also some Advanced features, e.g. cube maps

- `javax.microedition.m3g.shader`
  - Only present in Advanced implementations
What’s in the Core?

• Everything that’s in M3G 1.1
  – Fully backwards compatible
  – Well, except for flat shading

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

- Ericsson Texture Compression
- Half-float (FP16) vertex arrays
- New scene graph features (see later)
- New animation features (see later)
What’s in the Advanced Block?

• Everything that’s in OpenGL ES 2.0
  – Vertex and fragment shaders
  – Cube maps, advanced blending
  – Stencil buffering
New Core Features

SIGGRAPH 2007
Added Value of M3G 2.0 Core vs. 1.1

- 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 Quality Texturing

• Upgraded the baseline
  – At least two texture units
  – At least 1024x1024 maximum size

• Mandated optional features
  – Perspective correction
  – Mipmapping
  – Bilinear filtering
Bump Mapping

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

- Bump map modulated by projective light map
Texture Combiners

- Precursor to fragment shaders
  - Can do a lot more than bump and light mapping
  - Not very easy to use, though
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
  - Trivially expanded to float if necessary

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

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

- No performance penalty
  - Can be even faster with good vertex ordering
  - Assuming a vertex cache
## 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
### VertexBuffer Types – M3G 2.0

<table>
<thead>
<tr>
<th></th>
<th>Byte</th>
<th>Short</th>
<th>Fixed</th>
<th>Float</th>
<th>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>
<td>✔</td>
</tr>
<tr>
<td>TexCoords</td>
<td>✔</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>
<td>✔</td>
</tr>
<tr>
<td>Colors</td>
<td>✔</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>
<td>✔</td>
</tr>
</tbody>
</table>

* OpenGL ES 1.1 only supports RGBA colors
Deprecated 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!
Shaders

Programs
Uniforms
Attributes
Shading Language

• GLSL ES, but source code only
  – Binary shaders would break the Java sandbox

• Added a few built-in functions
  – For morphing, skinning, etc.
The Shader Package

- Appearance
- Program
  - Uniforms
  - Uniforms
  - Uniforms
  - ...
- VertexShader
- FragmentShader
ShaderProgram

• Consists of
  – VertexShader
  – FragmentShader

• Linked on construction
  – Construction fails if linking fails
  – Validated on first use

• Can query active uniforms & attributes
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
## Vertex Attributes

<table>
<thead>
<tr>
<th></th>
<th>Byte</th>
<th>Short</th>
<th>Fixed</th>
<th>Half</th>
<th>Float</th>
</tr>
</thead>
<tbody>
<tr>
<td>1..4 components</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Scene Graph & Content Creation

SIGGRAPH 2007
Scene Graph

• Added automatic render-to-texture

• Otherwise mostly unchanged

• Some convenience methods
  – Can use quaternions instead of axis/angle
  – Can enable/disable animations hierarchically
The File Format

• Updated to match the new API
  – File structure remains the same
  – Same parser can handle both old & new

• Better compression
  – Textures (ETC, JPEG)
  – SkinnedMesh, IndexBuffer
Animation
Multichannel Keyframe Sequences

- $N$ channels per KeyframeSequence object
  - 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
  - M3G 2.0: two Java objects per mesh
Conclusion
Summary

• M3G 2.0
  – Brings programmable shaders to the mass market
  – Fully exploits fixed-function graphics hardware
  – Improves performance on CPU-only devices
Things Under Consideration

• Bounding volumes
• Collision detection
• Texture compression (encoding)
• Texture generation
• Particle systems
• Mesh modifiers
• …
Today’s program: Afternoon

- Start at 13:45
- M3G Intro
  10 min, Kari Pulli
- M3G API overview
  60 min, Tomi Aarnio
- M3G in the Real World 1
  25 min, Mark Callow
- Demos
  10 min
- Break 15:30 – 15:45
- M3G in the Real World 2
  60 min, Mark Callow
- M3G 2.0
  30 min, Tomi Aarnio
- Closing & Q&A
  15 min, Kari Pulli
- Finish at 17:30
Closing & Summary

- We have covered
  - OpenGL ES
  - M3G
Khronos API family

Cross-platform graphics authoring/acceleration ecosystem

Cross platform 2D/3D

Safety Critical 2D/3D

3D Authoring

Dynamic Media Authoring Standards

Embedded Media Acceleration APIs

“DirectX-like” set of native APIs
Includes mixed media acceleration and OS portability APIs
• 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
- Stroke
- Image transformation
- Fill rule
- Mask
- Paths
JSR-226 examples

Game, with skins
Scalable maps, variable detail
Cartoon
Weather info
EGL evolution

• 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 composting user interfaces
OpenGL ES for powerful 3D UI effects
Summary

• Fixed functionality mobile 3D is reality NOW
  – these APIs and devices are out there
  – go get them, start developing!
• Better content with Collada
• Solid roadmap to programmable 3D
• New standards for 2D vector graphics