Developing Mobile 3D Applications with OpenGL ES and M3G

K. Pulli and J. Vaarala and V. Miettinen and T. Aarnio and M. Callow

Abstract

Mobile phones offer exciting new opportunities for graphics application developers. However, they also have significant limitations compared to traditional desktop graphics environments, including absence of dedicated graphics hardware, limited memory (both RAM and ROM), limited communications bandwidth, and lack of floating point hardware. Existing graphics APIs ignore these limitations and thus are infeasible to implement in embedded devices.

This course presents two new 3D graphics APIs that address the special needs and constraints of mobile/embedded platforms: OpenGL ES and M3G. OpenGL ES is a light-weight version of the well-known workstation standard, offering a subset of OpenGL 1.5 capability plus support for fixed point arithmetic. M3G, Mobile 3D Graphics API for Java MIDP (Mobile Information Device Profile), also known as JSR-184, provides scene graph and animation support, binary file format, and immediate mode rendering that bypasses scene graphs. These APIs provide powerful graphics capabilities in a form that fits well on today's devices, and will support hardware acceleration in the future.

The course begins with a discussion of the target environments and their limitations, and general techniques for coping with platform/environment constraints (such as fixed point arithmetic). This is followed by detailed presentations of the APIs. For each API, we describe the included functionality and compare it to related workstation standards, explaining what was left out and why. We also discuss practical aspects of working with the APIs on the target platforms, and present strategies for porting existing applications and creating new ones.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Standards

1. Course Syllabus

1.1. Graphics for Mobile Devices

- characteristics and constraints of mobile devices
- brief history of mobile 3D graphics
- mobile 3D APIs

1.2. OpenGL ES Overview

- OpenGL, OpenGL ES, and Khronos
- design principles
- OpenGL ES 1.0: features, which parts of OpenGL were kept, what was added
- OpenGL ES 1.1 features
- EGL: interface between OpenGL ES and the OS
- devices, implementations, SDKs, demos

1.3. Using OpenGL ES

- Hello OpenGL ES on Symbian
- fixed point programming
- converting existing OpenGL code to OpenGL ES

1.4. Building Scalable 3D Applications

- mobile graphics platforms
- how to scale 3D applications

1.5. M3G API Overview

- mobile Java
- M3G design principles
- M3G basic structure and features
- performance tips
- deforming meshes
- keyframe animation
- demos

1.6. Using M3G

- game development process
- asset creation
- program development
- midlet structure
- midlet example
- challenges in mobile game development
- publishing your content

1.7. Closing and Summary

- current status of mobile graphics standards, both 3D and 2D
- roadmap to future

2. Speakers

Kari Pulli has been an active contributor in both OpenGL ES and M3G (JSR-184) standardization groups. Kari is a Research Fellow at Nokia and is currently a Visiting Scientist at MIT. Before joining Nokia in 1999, Kari worked on graphics at Microsoft, SGI, and Alias|Wavefront, obtained a PhD at University of Washington in graphics in 1997, and was the technical head of the Digital Michelangelo project at Stanford Graphics Lab in 1998-99. Kari is a member of the Eurographics Executive Committee.

Jani Vaarala is a Graphics SW Architect at Nokia. He has been actively involved with OpenGL ES standardization, and headed a project that developed a SW OpenGL ES engine and adapted EGL for Symbian OS. Jani started on 3D graphics in early 90's on an Amiga, on which he developed several award-winning graphics demos.

Ville Miettinen is the CTO and co-founder of Hybrid Graphics, Ltd. During the last decade he has been involved in the design and implementation of numerous software products in the games and 3D graphics industries. His research interests include dynamic code generation and software rasterizers, and he has authored conference and journal papers on graphics hardware and visibility optimization. He is a member of ACM SIGGRAPH, the Khronos Group and the JSR-184 expert group.

Tomi Aarnio is the specification editor and one of the main contributors in the M3G (JSR-184) standardization group, and a member of the OpenGL ES group. As a Senior Research Engineer at Nokia, he has been involved in designing and implementing several mobile graphics engines, most recently heading the implementation of M3G.

Mark Callow is Chief Architect at HI Corporation, the leader in 3D graphics engines for mobile devices, whose Mascot Capsuleő Micro3D Engine is found on more than 30,000,000 handsets. Mark leads an international team creating implementations of M3G and OpenGLőES and was active in the creation of both standards. Prior to HI, Mark was with Silicon Graphics for 11 years were he created InPerson, a collaborative desktop video conferencing system; Mark was also Cosmo Software's liaison to the MPEG-4 standards committee. He previously taught several well-received Siggraph courses and is a member of ACM, ACM Siggraph, IEE, the Khronos Group and the JSR-184 expert group.

3. Further information

Course materials and other related information can be found at http://people.csail.mit.edu/kapu/mobile_3D_course/.

Developing Mobile 3D Applications with OpenGL ES and M3G

Kari Pulli	Nokia Research Center
Jani Vaarala	Nokia
Ville Miettinen	Hybrid Graphics
Tomi Aarnio	Nokia Research Center
Mark Callow	HI Corporation

Today's program

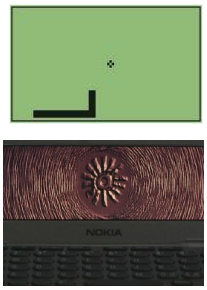
- Start at ?:??
- Intro & OpenGL ES overview
  25 min, Kari Pulli
- Using OpenGL ES
  40 min, Jani Vaarala
- OpenGL ES performance
  25 min, Ville Miettinen
- Break ?:?? – ?:??
- M3G API overview
  45 min, Tomi Aarnio
- Using M3G
  40 min, Mark Callow
- Closing & Q&A
  5 min, Kari Pulli

Challenges for mobile gfx

- Small displays
  - getting much better
- Computation
  - speed
  - power / batteries
  - thermal barrier
- Memory

State-of-the-art in 2001: GSM world

- The world's most played electronic game?
  - According to The Guardian (May 2001)
- Communicator demo 2001
  - Remake of a 1994 Amiga demo
  - <10 year from PC to mobile



State-of-the-art in 2001: Japan

- High-level API with skinning, flat shading / texturing, orthographic view

State-of-the-art in 2002: GSM world

- 3410 shipped in May 2002
  - A SW engine: a subset of OpenGL including full perspective (even textures)
  - 3D screensavers (artist created content)
  - FlyText screensaver (end-user content)
  - a 3D game

State-of-the-art in 2002: Japan

- Gouraud shading, semi-transparency, environment maps

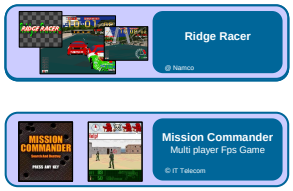
State-of-the-art in 2003: GSM world

- 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

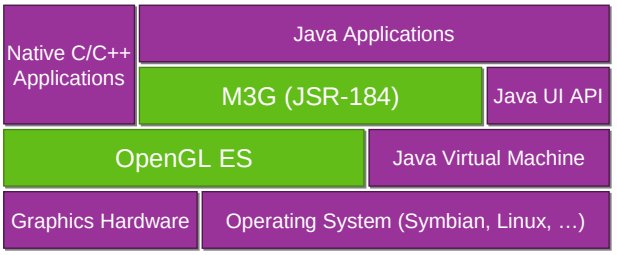


Mobile 3D in 2004

- 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 3D APIs

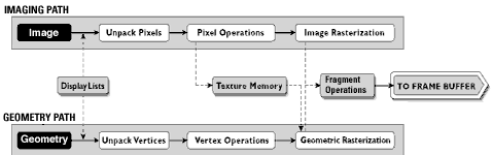


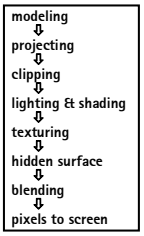
Overview: OpenGL ES

- 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

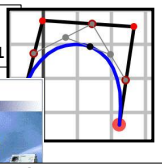
Outline

- 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?

Functionality: in / out? (1/7)

- Convenience functionality is <u>OUT</u>
  - 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)

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




```
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

```
glBegin(GL_POLYGON);
glColor3f (1, 0, 0);
glVertex3f(-.5, .5, .5);
glVertex3f( .5, .5, .5);
glColor3f (0, 1, 0);
glVertex3f( .5,-.5, .5);
glVertex3f(-.5,-.5, .5);
glEnd();
```

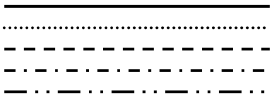
```
static const GLbyte verts[4 * 3] =
{    -1,  1,  1,      1,  1,  1,
      1, -1,  1,     -1, -1,  1 };
static const GLubyte colors[4 * 3] =
{   255,  0,  0,    255,  0,  0,
      0,255,  0,      0,255,  0 };
glVertexPointer( 3,GL_BYTE,0, verts );
glColorPointerf( 3,GL_UNSIGNED_BYTE,
                 0, colors );
glDrawArrays( GL_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 **IN**, DrawPixels and Bitmap **OUT**
- Rasterization
  - **OUT:** PolygonMode, PolygonSmooth, Stipple

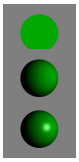


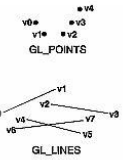
Functionality: in / out? (5/7)

- 2D texture maps <u>**IN**</u>
  - 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 <u>**IN**</u>
  - back materials, local viewer, separate specular **OUT**
- Primitives
  - **IN:** points, lines, triangles
  - **OUT:** polygons and quads





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

```
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

OpenGL ES 1.1: optional

- Draw Texture
  fast drawing of pixel rectangles using texturing units (data can be cached), constant Z, scaling
- Matrix Palette
  vertex skinning (>= 3 matrices / vertex, palette >= 9)

Outline

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

EGL glues OpenGL ES to OS

- EGL is the interface between OpenGL ES and the native platform window system
  - similar to GLX on X-windows, WGL on Windows
  - facilitates portability across OS's (Symbian, Linux, …)
- Division of labor
  - EGL gets the resources (windows, etc.) and displays the images created by OpenGL ES
  - OpenGL ES uses resources for 3D graphics

EGL surfaces

- Various drawing surfaces, rendering targets
  - *windows* – on-screen rendering ("graphics" memory)
  - *pbuffers* – off-screen rendering (user memory)
  - *pixmaps* – off-screen rendering (OS native images)

EGL context

- A rendering context is an abstract OpenGL ES state machine
  - stores the state of the graphics engine
  - can be (re)bound to any matching surface
  - different contexts can share data
    - texture objects
    - vertex buffer objects
    - lately even across APIs (OpenGL ES, OpenVG)

Main EGL 1.0 functions

- Getting started
  - eglInitialize() / eglTerminate(), eglGetDisplay(), eglGetConfigs() / eglChooseConfig(), eglCreate**X**Surface() (**X** = Window | Pbuffer | Pixmap), eglCreateContext()
- eglMakeCurrent(display, drawsurf, readsurf, context)
  - binds context to current thread, surfaces, display

Main EGL 1.0 functions

- eglSwapBuffer(display, surface)
  - posts the color buffer to a window
- eglWaitGL( ), eglWaitNative(engine)
  - provides synchronization between OpenGL ES and native (2D) graphics libraries
- eglCopyBuffer(display, surface, target)
  - copy color buffer to a native color pixmap

EGL 1.1 enhancements

- Swap interval control
  - specify # of video frames between buffer swaps
  - default 1; 0 = unlocked swaps, >1 save power
- Power management events
  - PM event => all Context lost
  - Disp & 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
- Vincent
  - Open-source OpenGL ES library
  - http://sourceforge.net/projects/ogl-es
- Reference implementation
  - Wraps on top of OpenGL
  - http://www.khronos.org/opengles/documentation/gles-1.0c.tgz

On-Device Implementations

- NokiaGL (SW)
- N93 (HW)
- Imagination MBX
- NVidia GoForce 3D
- ATI Imageon
- Toshiba T4G
- …

SDKs

- Nokia S60 SDK (Symbian OS)
  - http://www.forum.nokia.com
- Imagination SDK
  - http://www.pvrdev.com/Pub/MBX
- NVIDIA handheld SDK
  - http://www.nvidia.com/object/hhsdk_home.html
- Brew SDK & documentation
  - http://brew.qualcomm.com

OpenGL ES 1.1 Demos

Questions?

Using OpenGL ES

Jani Vaarala

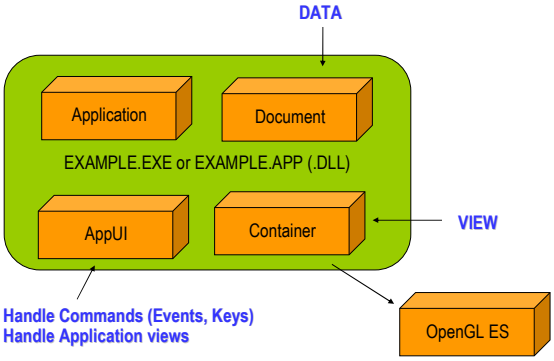
Nokia

Using OpenGL ES

- Simple OpenGL ES example
- Fixed point programming
- Converting existing code

“Hello OpenGL ES”

Symbian App Classes



“Hello OpenGL ES”

```
/* ============================================================
 *  "Hello OpenGL ES" OpenGL ES code.
 *
 *  Eurographics 2006 course on mobile graphics.
 *
 *  Copyright: Jani Vaarala
 * ============================================================
*/

#include <e32base.h>
#include "SigTriangleGL.h"

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

“Hello OpenGL ES”

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

“Hello OpenGL ES”

```
static void initGLES()
{
   glClearColor        (0.f,0.f,0.1f,1.f);
   glDisable           (GL_DEPTH_TEST);
   glMatrixMode        (GL_PROJECTION);
   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);
}
```

“Hello OpenGL ES”

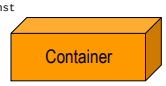
```
TInt CSigTriangleGL::DrawCallback( TAny* aInstance )
{
   CSigTriangleGL* instance = (CSigTriangleGL*) aInstance;

   glClear           (GL_COLOR_BUFFER_BIT);
   glLoadIdentity    ();
   glTranslatef      (0,0,-5.f);
   glDrawArrays      (GL_TRIANGLES,0,3);

   eglSwapBuffers    (instance->iEglDisplay,instance->iEglSurface);

   /* To keep the background light on */
   if (!(instance->iFrame%100))        User::ResetInactivityTime();

   instance->iFrame++;
   return 0;
}
```

“Hello OpenGL ES”

```
void CSigTriangleContainer::ConstructL(const TRect& /* aRect */)
{
   iGLInitialized = EFalse;

   CreateWindowL();
   SetExtentToWholeScreen();
   ActivateL();

   CSigTriangleGL* gl = new (ELeave) CSigTriangleGL( );
   gl->Construct(Window());

   iGLInitialized = ETrue;
}

CSigTriangleContainer::~CSigTriangleContainer()
{
}
```

“Hello OpenGL ES”

```
void CSigTriangleContainer::SizeChanged()
{
    if(iGLInitialized)
    {
        glViewport(0,0,Size().iWidth,Size().iHeight);
    }
}

void HandleResourceChange( TInt aType )
{
    if(aType == KEikDynamicLayoutSwitch)
    {
        // Screen resolution changed, make window fullscreen in a new resolution
        SetExtentToWholeScreen();
    }
}

TInt CSigTriangleContainer::CountComponentControls() const
{
    return 0;
}

CCoeControl* CSigTriangleContainer::ComponentControl(TInt /* aIndex */) const
{
    return NULL;
}
```



“Hello OpenGL ES”

```
/*************************************************************
 * Initialize OpenGL ES context and initial OpenGL ES state *
 *************************************************************/
void CSigTriangleGL::Construct(RWindow aWin)
{
   iWin = aWin;

   iEglDisplay = eglGetDisplay(EGL_DEFAULT_DISPLAY);
   if(iEglDisplay == NULL )    User::Exit(-1);

   if(eglInitialize(iEglDisplay,NULL,NULL) == EGL_FALSE)
                               User::Exit(-1);

   EGLConfig  config,colorDepth;
   EGLint     numOfConfigs = 0;
```

“Hello OpenGL ES”

```
switch( iWin.DisplayMode() )
{
     case (EColor4K):   { colorDepth = 12; break; }
     case (EColor64K):  { colorDepth = 16; break; }
     case (EColor16M):  { colorDepth = 24; break; }
     default:
                        colorDepth = 32;
}

EGLint attrib_list[] = {    EGL_BUFFER_SIZE, colorDepth,
                            EGL_DEPTH_SIZE,  15,
                            EGL_NONE             };

if(eglChooseConfig(iEglDisplay,attrib_list,&config,1,
     &numOfConfigs ) == EGL_FALSE) User::Exit(-1);
```

“Hello OpenGL ES”

```
iEglSurface = eglCreateWindowSurface(iEglDisplay, config, &iWin, NULL );
if( iEglSurface == NULL )                        User::Exit(-1);

iEglContext = eglCreateContext(iEglDisplay,config, EGL_NO_CONTEXT, NULL );
if( iEglContext == NULL )                        User::Exit(-1);

if( eglMakeCurrent( iEglDisplay, iEglSurface, iEglSurface,
              iEglContext ) == EGL_FALSE )       User::Exit(-1);
```

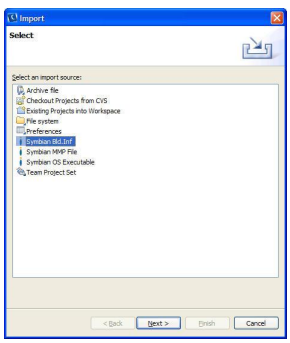
“Hello OpenGL ES”

```
/* Create a periodic timer for display refresh */
iPeriodic = CPeriodic::NewL( CActive::EPriorityIdle );

iPeriodic->Start( 100, 100, TCallBack(
                        SigTriangleGL::DrawCallback, this ) );

initGLES();
```

Carbide C++ Express

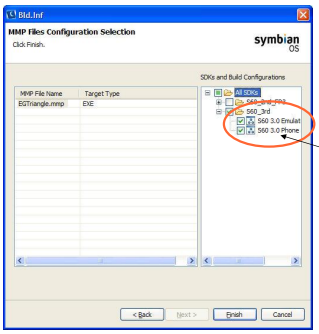
- Free IDE for S60 development from
  - http://www.forum.nokia.com
- Supports 2nd edition and 3rd edition SDKs
- Here we focus on 3rd edition
  - Future devices will be 3rd edition (e.g., N93)

Importing project



Importing project

Importing project

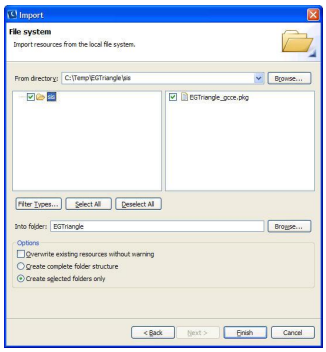


Select emulator configuration and phone configuration (GCCE) under S60_3rd.

Importing .PKG file (for .SIS)

- Select from menu: File -> Import
- Select “File System”
- Navigate to folder “sis” and import .PKG file
  - “EGTriangle_gcce.pkg”
- Build will automatically generate install file

Importing .PKG file

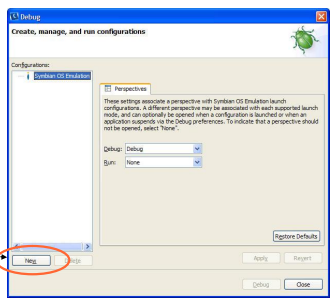


Compiling & Debugging

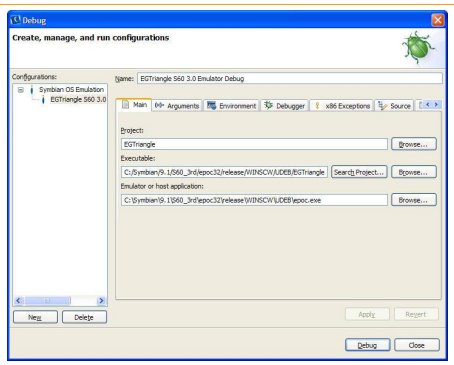
- Select from menu: Project -> Build ALL
- Select from menu: Run -> Debug

Creating debug config

Click “New” to create new debug config.



Creating debug config

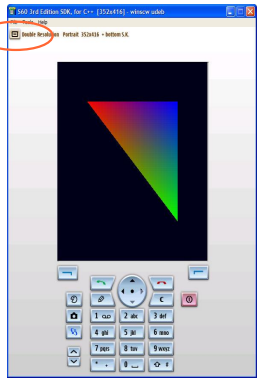


Selecting application

- When emulator starts, navigate to “Installat.” folder
- Select application to launch (EGTriangle)

Application

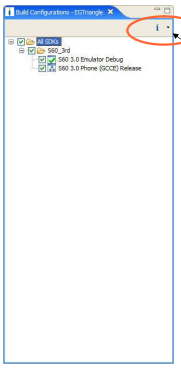
Click this button to cycle through resolutions and check that your application works in all resolutions.



Getting it to HW

- Go to menu: Window -> Open Perspective -> Other
- Select “Symbian (default)”
- Go to menu: Window -> Show view -> Build Configurations

Selecting build configuration



Click this button to open a list of possible build configurations. Select “S60 3.0 Phone (GCCE) Release”

Installation file

- Build the project (CTRL-B)
- Installation file is generated during build
- Select it from C/C++ Projects view
  - EGTriangle_GCCE.sis
- From context menu select “copy”
- Paste it to desktop and send using bluetooth

Fixed point programming

- Why to 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

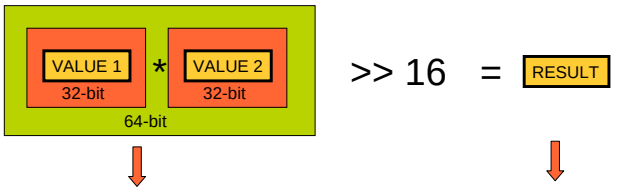
- There are many variants of fixed point:
  - Signed / Unsigned
  - 2's complement vs. Separate sign
- OpenGL ES uses 2's complement
- Numbers in the range of [-32768, 32768 [
- 16 bits for decimal bits (precision of 1/65536)
- All the examples here use .16 fixed point

Fixed point programming

- Examples:

```
0x0001 0000 =  "1.0f"
0x0002 0000 =  "2.0f"
0x0010 0000 =  "16.0f"
0x0000 0001 =  1/0x10000(0x10000 = 2^16)
0xffff ffff = -1/0x10000(-0x0000 0001)
```

Fixed point programming



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

Result overflow
- Redo range analysis
- Detect overflow, clamp

Fixed point programming

- Convert from floating point to fixed point

```
#define float_to_fixed(a)  (int)((a)*(1<<16))
```

- Convert from fixed point to floating point

```
#define fixed_to_float(a)  (((float)a)/(1<<16))
```

- Addition

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

- Multiply fixed point number with integer

```
#define mul_fixed_int(a,b) ((a)*(b))
```

Fixed point programming

- MUL two FP numbers together

```
#define mul_fixed_fixed(a,b) (((a)*(b)) >> 16)
```

  - If another multiplier is in] -1.0, 1.0 [, no overflow
- Division of integer by integer to a fixed point result

```
#define div_int_int(a,b) (((a)*(1<<16))/(b))
```

- 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) (((a)*(1<<16))/(b))
```

Fixed point programming

- Power of two MUL & DIV can be done with shifts
- 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)
  - Using unnecessary "extra bits" slows execution
- Always add debugging code to your fixed point math

Fixed point programming

```
#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 programming

- Complex math functions
  - Pre-calculate for the range of interest
- An example: Sin & Cos
  - Sin table between [0, 90°]
  - Fixed point angle
  - Generate other angles and Cos from the table
  - Store as fixed point ((short) (sin(angle) * 32767))
  - Performance vs. space tradeoff: calculate for all angles

Fixed point programming

- Sin
  - 90 ° = 2048 (our angle scale)
  - Sin table needs to include 0° and 90°

```
INLINE fp_sin(int angle)
{
   int phase                  = angle & (2048 + 4096);
   int subang                 = angle & 2047;

   if( phase == 0 )           return sin_table  (subang);
   else if( phase == 2048 )   return sin_table  (2048 - subang);
   else if( phase == 4096 )   return –sin_table (subang);
   else                       return –sin_table (2048 – subang);
}
```

Example: Morphing

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

```
#define DOMORPH_16(a,b,t) (TInt16)(((((b)-(a))*(t))>>16)+(a))

void MorphGeometry(TInt16 *aOut, const TInt16 *aInA, const TInt16
   *aInB, TInt aCount, TInt aScale)
{
   int i;

   for(i=0; i<aCount; i++)
   {
        aOut[i*3+0] = DOMORPH_16(aInB[i*3+0], aInA[i*3+0], aScale);
        aOut[i*3+1] = DOMORPH_16(aInB[i*3+1], aInA[i*3+1], aScale);
        aOut[i*3+2] = DOMORPH_16(aInB[i*3+2], aInA[i*3+2], aScale);
   }
}
```

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

```
_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

OpenGL porting

```
void _glEnable( par1, par2 )
{
  if( GLOBAL()->iSubmittingDisplayList )
  {
    *(GLOBAL()->dlist)++ = DLIST_CMD_GLENABLE;
    *(GLOBAL()->dlist)++ = (GLuint)par1;
    *(GLOBAL()->dlist)++ = (GLuint)par2;
  }
  else
  {
    glEnable(par1,par2);
  }
}
```

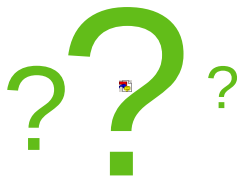
OpenGL porting

- 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 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?



Building scalable 3D applications

Ville Miettinen

Hybrid Graphics

What is this ”mobile platform”?

- CPU speed and available memory varies
  - Current range ~30Mhz - 600MHz, no FPUs
- Portability issues
  - Different CPUs, OSes, Java VMs, C compilers, ...
- Different resolutions
  - QCIF (176x144) to VGA (640x480), antialiasing on higher-end devices
  - Color depths 4-8 bits per channel (12-32 bpp)

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 HW
- Performance: 50K – 2M tris, 1M – 100M pixels

Dealing with diversity

- Problem: running the same game on 100+ different devices
  - Same gameplay but can scale video and audio
- Scalability must be built into game design
- Profile-based approach

3D content is easy to scale

- Separate low and high poly 3D models
- Different texture resolutions & compressed formats
- Scaling down special effects not critical to game play (particle systems, shadows)
  - Important to realize what is a ”special effect”
- Rendering quality controls
  - Texture filtering, perspective correction, blend functions, multi-texturing, antialiasing

Building scalable 3D apps

- OpenGL ES created to standardize the API and behavior
  - ES does not attempt to standardize performance
  - Two out of three ain’t bad
- Differences between SW/HW configurations
  - Trade-off between flexibility and performance
  - Synchronization issues

Building scalable 3D apps

- Scale upwards, not downwards
  - Bad experiences of retro-fitting HW titles to SW
  - Test during development on lowest-end platform
- Both programmers and artists need education
  - Artists can deal with almost anything as long as they know the rules…
  - And when they don’t, just force them (automatic checking in art pipeline)

Reducing state changes

- Don’t mix 2D and 3D calls !!!!
  - Situation may become better in the future, though...
- Unnecessary state changes root of all evil
  - Avoid changes affecting the vertex pipeline
  - Avoid changes to the pixel pipeline
  - Avoid changing textures

”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: Important to keep shaders as simple as possible
  - Do 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

Of models and stripping

- Use buffer objects of ES 1.1
  - Only models changed manually every frame need vertex pointers
  - Many LOD schemes can be done just by changing index buffers
- Keep data formats short and simple
  - Better cache coherence, less memory used

Triangle data

- Minimize number of rendering calls
  - Trade-off between no. of render calls & culling efficiency
  - Combine strips using degenerate triangles
  - Understanding vertex caching
    - Automatically optimize vertex access order
    - Triangle lists better than their reputation
- Optimize data in your art pipeline (exporters)
  - Welding vertices with same attributes (with tolerance)
    - Vertices/triangle ratio in good data 0.7-1.0
  - Give artists as much automatic feedback as possible

Transformations and matrices

- 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
  - Characters, natural motion: grass, trees, waves
- ES 1.1: Point sprites

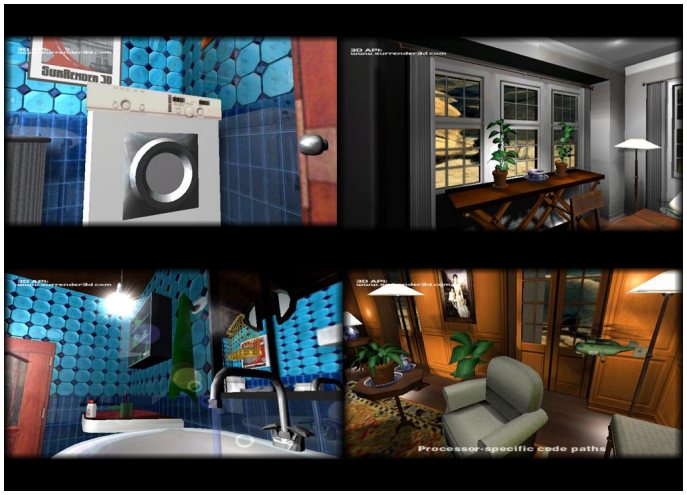
Lighting and materials

- 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

Lighting: the fast way

- While we're waiting for OpenGL ES 2.0...
  - 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
- Color material tracking for changing materials
- Flat shading is for flat models!

Illumination using multitexturing



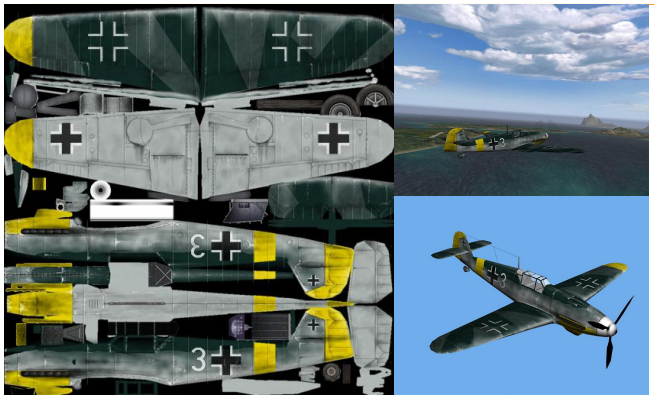
Textures

- Mipmaps always a Good Thing™
  - Improved cache coherence and visual quality
  - ES 1.1 supports auto mipmap generation
- Different strategies for texture filtering
- SW: Perspective correction not always needed
- Avoid modifying texture data
- Keep textures ”right size”, use compressed textures

Textures

- Multitexturing
  - Needed for texture-based lighting
  - Always faster than doing multiple rendering passes
  - ES 1.1: support at least two texturing units
  - ES 1.1: TexEnvCombine neat toy
- Combine multiple textures into single larger one
  - Reduce texture state changes (for fonts, animations, light maps)

Textures and shots from Kesmai’s Air Warrior 4 (never published)



Object ordering

- Sort objects into optimal rendering order
  - Minimize shader changes
  - Keep objects in front-to-back order
    - Improves Z-buffering efficiency
  - Satisfying both goals: bucketize objects by shader, sort buckets by Z

Thank you!

- Any questions?

M3G Overview

Tomi Aarnio

Nokia Research Center

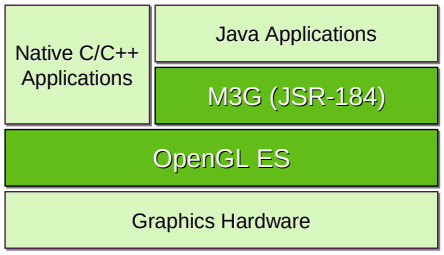
Objectives

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

Prerequisites

- Fundamentals of 3D graphics
- Some knowledge of OpenGL ES
- Some knowledge of scene graphs

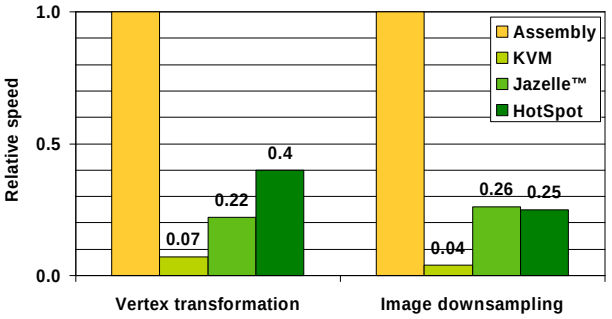
Mobile 3D Graphics APIs



Why Should You Use Java?

- It has the largest and fastest-growing installed base
  - 1.2B Java phones had been sold by June 2006 (source: Ovum)
  - Nokia alone had sold 350M Java phones by the end of 2005
  - Less than 50M of those also supported native S60 applications
- It increases productivity compared to C/C++
  - Memory protection, type safety ➔ fewer bugs
  - Fewer bugs, object orientation ➔ better productivity

Java Will Remain Slower



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

Why?

- Array bounds checking
- Dynamic type checking
- No stack allocation (heap only)
- Garbage collection
- Slow Java-native interface
- No access to special CPU features
- Stack-based (non-RISC) bytecode
- Unpredictable JIT compilers

No Java compiler or accelerator can fully resolve these issues

M3G Overview

Design principles

Getting started

Basic features

Performance tips

Deforming meshes

Keyframe animation

Summary & demos

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
  - Keep all data on the native side to avoid Java-native traffic

M3G Design Principles

#2 **Cater for both software and hardware**

- Do not add features that are too heavy for software engines
  - Such as per-pixel mipmapping or floating-point vertices
- Do not add features that break the OpenGL 1.x pipeline
  - Such as hardcoded transparency shaders

M3G Design Principles

#3 **Maximize developer productivity**

- 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”

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, function calls needed for simple things
  - No support for animation and scene management
  - Fails on Design Principles 1 (performance) and 3 (productivity)
  - …but may become practical with faster Java virtual machines
- Java 3D is too bloated
  - A hundred times larger (!) than M3G
  - Still lacks a file format, skinning, etc.
  - Fails on Design Principles 1, 3, and 4 (code size)

M3G Overview

Design principles

Getting started

Basic features

Performance tips

Deforming meshes

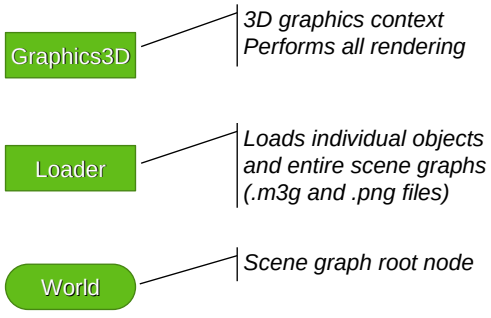
Keyframe animation

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
- Scene update is decoupled from rendering
  - **`render`** ➔ Draws an object or scene, no side-effects
  - **`animate`** ➔ Updates an object or scene to the given time
  - **`align`** ➔ Aligns scene graph nodes to others

Main Classes



Rendering State

- Graphics3D contains global state
  - Frame buffer, depth buffer
  - Viewport, depth range
  - Rendering quality hints
- 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

```
void paint(Graphics g) {
   try {
      myGraphics3D.bindTarget(g);
      myGraphics3D.render(world);
   } finally {
      myGraphics3D.releaseTarget();
   }
}
```

M3G Overview

Design principles

Getting started

Basic features

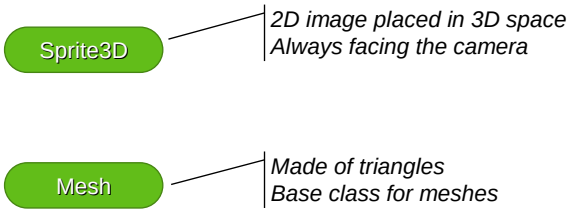
Performance tips

Deforming meshes

Keyframe animation

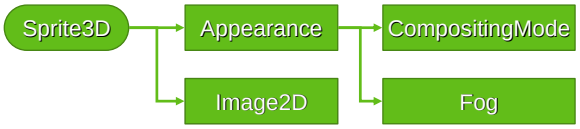
Summary & demos

Renderable Objects



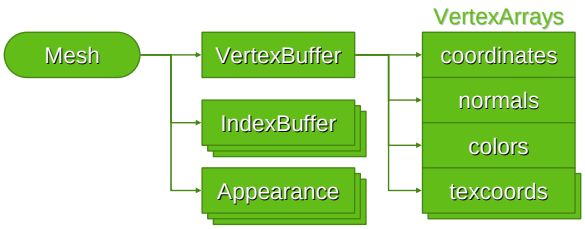
Sprite3D

- 2D image with a position in 3D space
- Scaled mode for billboards, trees, etc.
- Unscaled mode for text labels, icons, etc.
- Not useful for particle effects – too much overhead



Mesh

- A common VertexBuffer, referencing VertexArrays
- IndexBuffers (submeshes) and Appearances match 1:1



VertexBuffer Types

	Byte	Short	Fixed	Float	2D	3D	4D
Vertices	✓	✓	✗	✗	✗	✓	✗
Texcoords	✓	✓	✗	✗	✓	✓	✗
Normals	✓	✓	✗	✗		✓	
Colors	✓		✗	✗		✓	✓

Relative to OpenGL ES 1.1

IndexBuffer Types

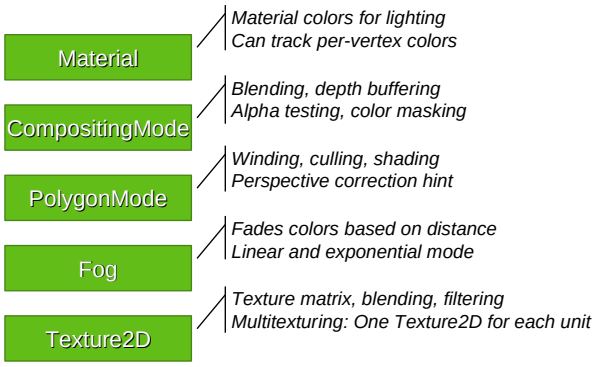
	Byte	Short	Implicit	Strip	Fan	List
Triangles	✗	✓	✓	✓	✗	✗
Lines	✗	✗	✗	✗	✗	✗
Points	✗	✗	✗			✗
Point sprites	✗	✗	✗			✗

Relative to OpenGL ES 1.1 + point sprite extension

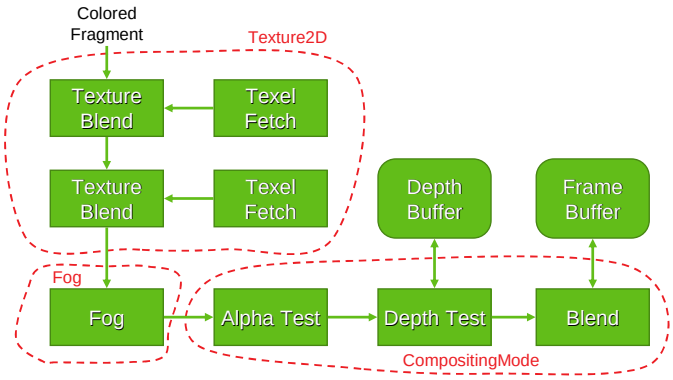
Buffer Objects

- Vertices and indices are stored on server side
  - Very similar to OpenGL Buffer Objects
  - Allows caching and preprocessing (e.g., bounding volumes)
- Tradeoff – Dynamic updates have some overhead
  - At the minimum, just copying in the Java array contents
  - In the worst case, may trigger vertex preprocessing

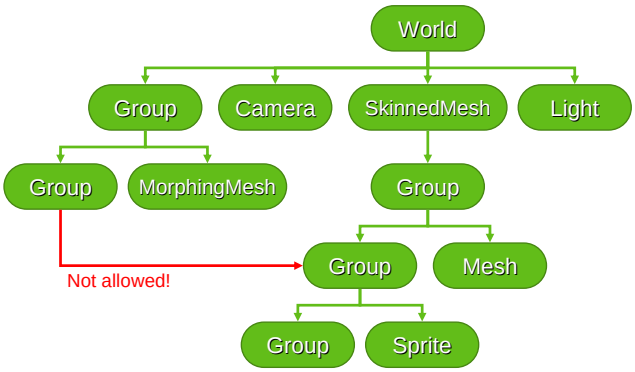
Appearance Components



The Fragment Pipeline

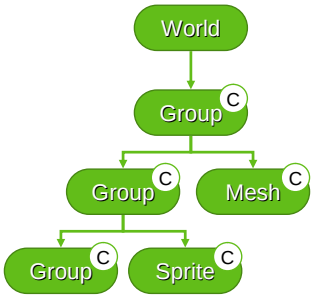


The Scene Graph



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
- Composite: **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

Design principles

Getting started

Basic features

Performance tips

Deforming meshes

Keyframe animation

Summary & demos

Retained Mode

- Use the retained mode
  - Do not render objects separately – place them in a World
  - Minimizes the amount of Java code and method calls
  - Allows the implementation to do 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

- Use layers to impose a rendering order
  - Appearance contains a layer index (an integer)
  - Defines a global ordering for submeshes & sprites
  - Can simplify shader state for backgrounds, overlays
  - Also enables multipass rendering in retained mode
- Optimize the rendering order
  - Shader state sorting done by the implementation
  - Use layers to force back-to-front ordering

Textures

- Use multitexturing to save in T&L and triangle setup
- Use mipmapping to save in memory bandwidth
- Combine small textures into texture atlases
- Use the perspective correction hint (where needed)
  - Usually 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
  - Lots of small meshes and sprites to render ➔ bad
  - Ideally, everything would be in one big triangle 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
  - HW: Do not mix 2D and 3D rendering
- In general, OpenGL ES performance tips apply

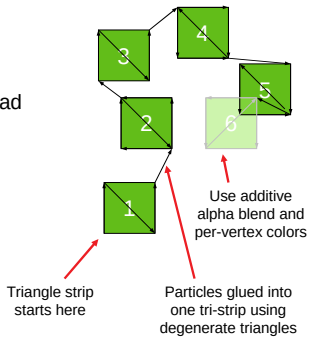
Particle Effects

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



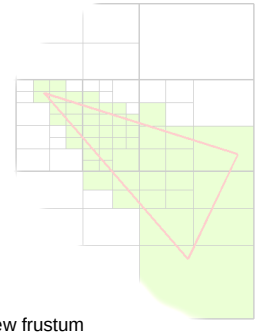
Terrain Rendering

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 the terrain into a quadtree
- Quadtree leaf node == Mesh object
- Quadtree inner node == Group object
- Enable nodes yourself, based on the view frustum



M3G Overview

Design principles

Getting started

Basic features

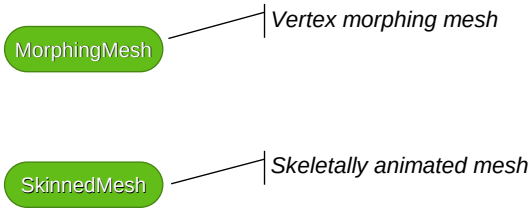
Performance tips

Deforming meshes

Keyframe animation

Summary & demos

Deforming Meshes



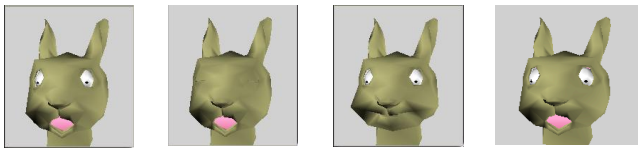
MorphingMesh

- Traditional vertex morphing animation
  - Can morph any vertex attribute(s)
  - A base mesh **B** and any number of morph targets $\mathbf{T}_i$
  - Result = weighted sum of morph deltas

$$\mathbf{R} = \mathbf{B} + \sum_i w_i (\mathbf{T}_i - \mathbf{B})$$

- Change the weights $\boldsymbol{w_i}$ to animate

MorphingMesh



Base | **Target 1 eyes closed** | **Target 2 mouth closed** | **Animate eyes and mouth independently**

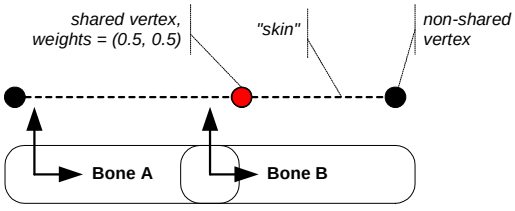
SkinnedMesh

- 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

$$v' = \sum_i w_i \mathbf{M}_i \mathbf{B}_i v$$

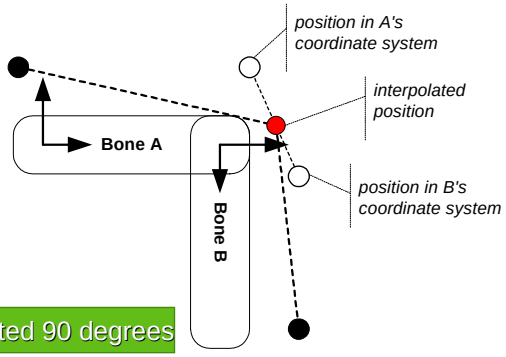
  - $\mathbf{M}_i$ are the node transforms – $\boldsymbol{v}$, $\boldsymbol{w}$, $\mathbf{B}$ are constant

SkinnedMesh



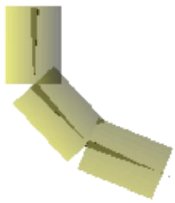
Neutral pose, bones at rest

SkinnedMesh

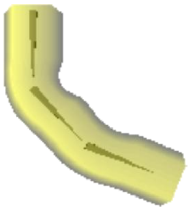


Bone B rotated 90 degrees

SkinnedMesh



No skinning



Smooth skinning
two bones per vertex

M3G Overview

Design principles

Getting started

Basic features

Performance tips

Deforming meshes

Keyframe animation

Summary & demos

Animation Classes

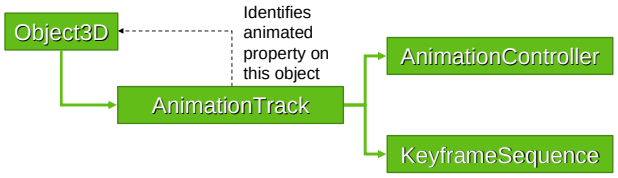
KeyframeSequence — *Storage for keyframes Defines interpolation mode*

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



KeyframeSequence

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)

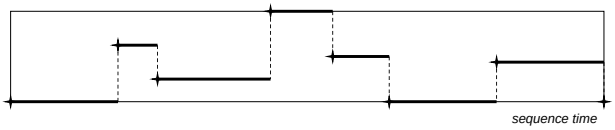


Diagram courtesy of Sean Ellis, Superscape

KeyframeSequence

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)

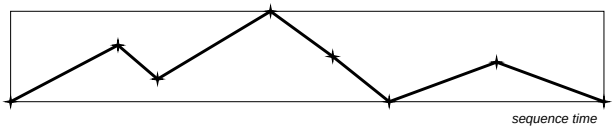


Diagram courtesy of Sean Ellis, Superscape

KeyframeSequence

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)

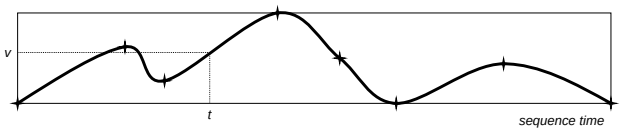


Diagram courtesy of Sean Ellis, Superscape

AnimationController

AnimationController

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

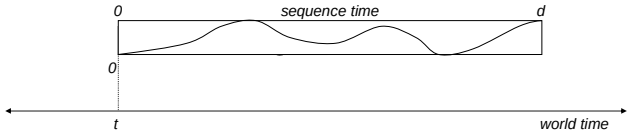


Diagram courtesy of Sean Ellis, Superscape

Animation

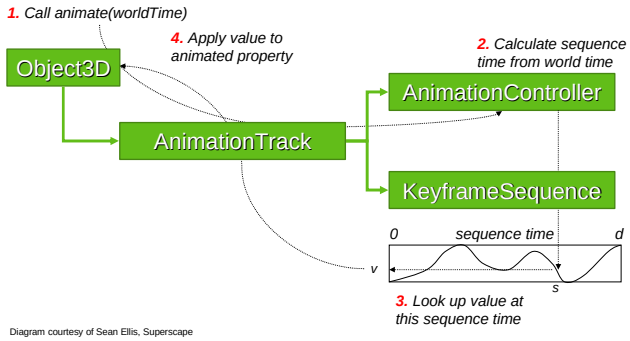


Diagram courtesy of Sean Ellis, Superscape

Animation

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

Design principles
Getting started
Basic features
Performance tips
Deforming meshes
Keyframe animation
Summary & demos

Predictions

- Resolutions will grow rapidly from 128x128 to VGA
  - Drives graphics hardware into all high-resolution devices
  - Software rasterizers can't compete above 128x128
- Bottlenecks will shift to Physics and AI
  - Bottlenecks today: Rasterization and any Java code
  - Graphics hardware will take care of geometry and rasterization
  - Java hardware will increase performance to within 50% of C/C++
- Java will reinforce its position as the dominant platform

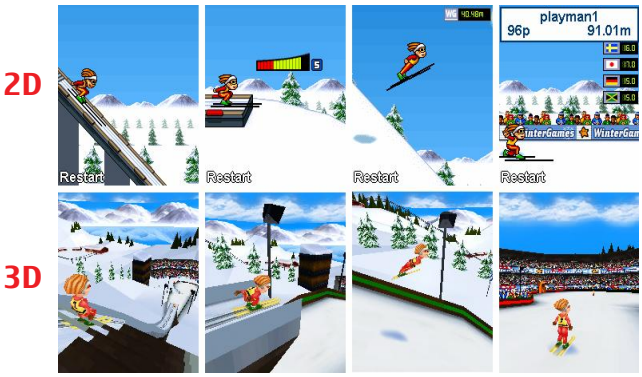
Summary

- M3G enables real-time 3D on mobile Java
  - By minimizing the amount of Java code along critical paths
  - Designed for both software and hardware implementations
- Flexible design leaves the developer in control
  - Subset of OpenGL ES features at the foundation
  - Animation & scene graph features layered on top

Installed base growing by the millions each month

Demos

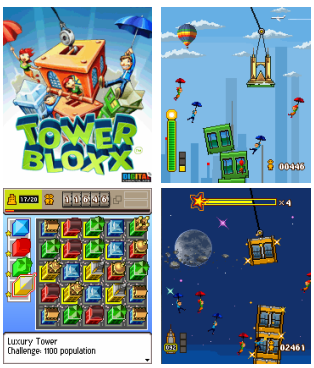
Playman Winter Games – Mr. Goodliving



Playman World Soccer – Mr. Goodliving

- An interesting 2D/3D hybrid
- Cartoon-like 2D characters set in a 3D scene
- 2D overlays for particle effects and status info

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

Q&A

Thanks: Sean Ellis, Kimmo Roimela,
Nokia M3G team, JSR-184 Expert Group,
Mr. Goodliving (RealNetworks),
Sumea (Digital Chocolate)

Using M3G

Mark Callow

Chief Architect

HI CORPORATION

Agenda

- Game Development Process
- Asset Creation
- Program Development
- MIDlet Structure
- A MIDlet Example
- Challenges in Mobile Game Development
- Publishing Your Content

M3G Game Demo

Game Development Process

- Traditional Java Game

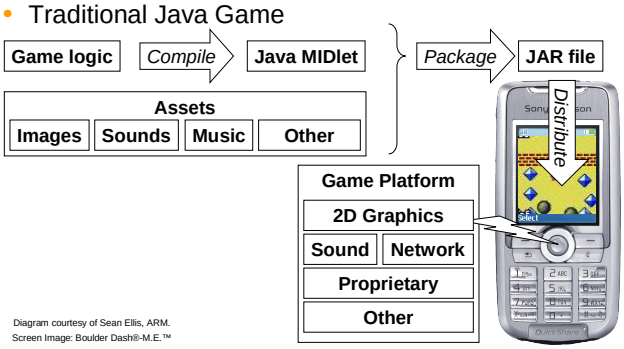


Diagram courtesy of Sean Ellis, ARM.
Screen Image: Boulder Dash®-M.E.™

M3G Development Process

- How M3G Fits

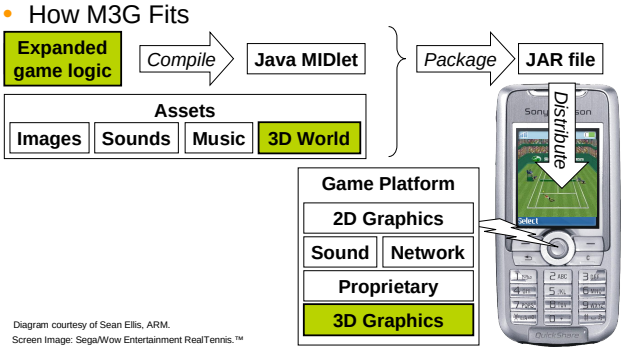
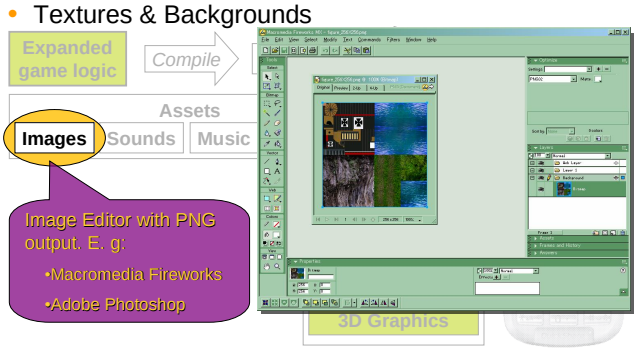


Diagram courtesy of Sean Ellis, ARM.
Screen Image: Sega/Wow Entertainment RealTennis.™

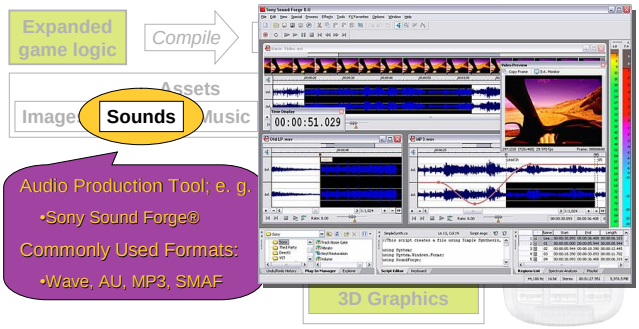
Asset Creation

- Textures & Backgrounds



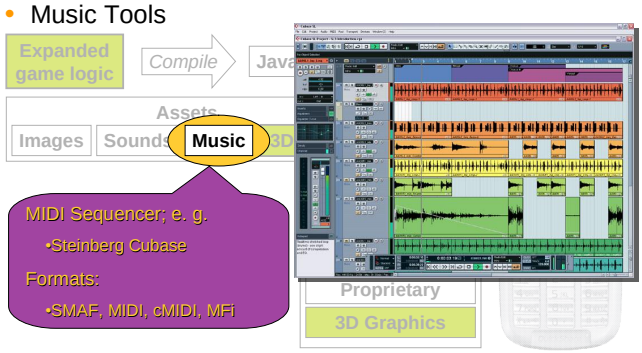
Asset Creation

- Audio Tools



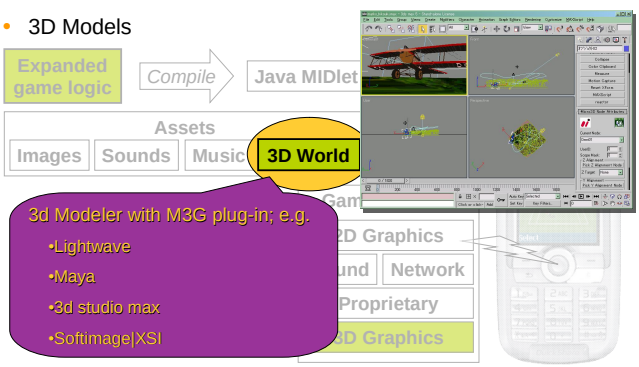
Asset Creation

- Music Tools

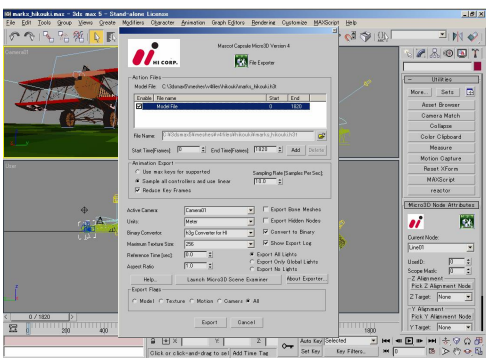


Asset Creation

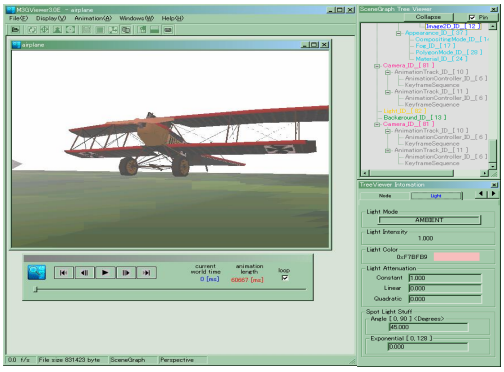
- 3D Models



Export 3d Model to M3G



M3G File Viewer



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

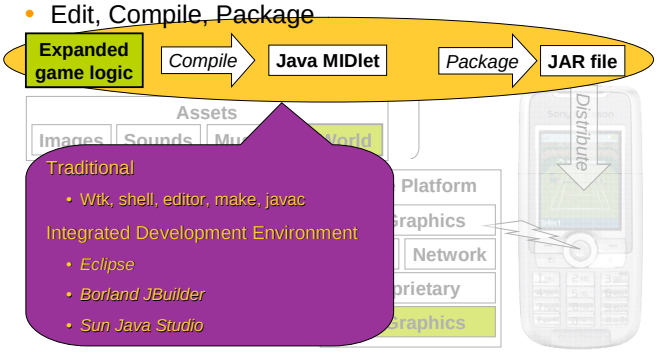
- *TIP: Use background images*
  - Can be scaled, tiled and scrolled very flexibly
  - Generally much faster than sky boxes or similar
- *TIP: Use sprites as impostors & labels*
  - Generally faster than textured quads
  - Unscaled mode is (much) faster than scaled
- *LIMITATION: Sprites are not useful for particle systems*

Agenda

- Game Development Process
- Asset Creation
- Program Development
- MIDlet Structure
- A MIDlet Example
- Challenges in Mobile Game Development
- Publishing Your Content

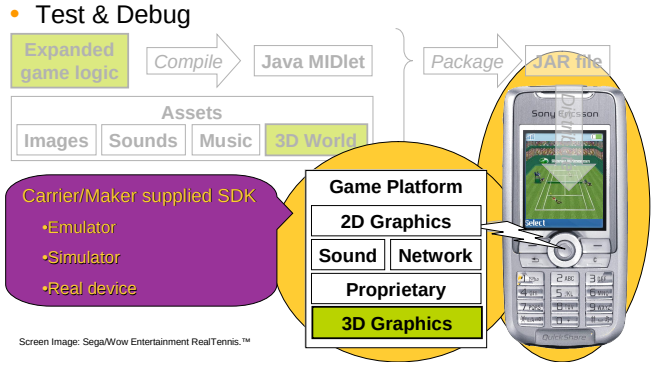
Program Development

- Edit, Compile, Package



Program Development

- Test & Debug



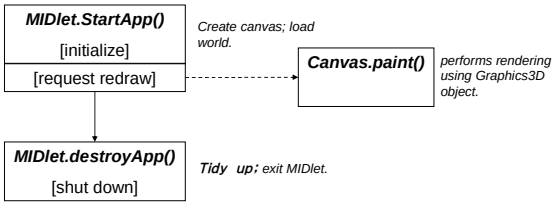
Screen Image: Sega/Wow Entertainment RealTennis.™

Agenda

- Game Development Process
- Asset Creation
- Program Development
- MIDlet Structure
- A MIDlet Example
- Challenges in Mobile Game Development
- Publishing Your Content

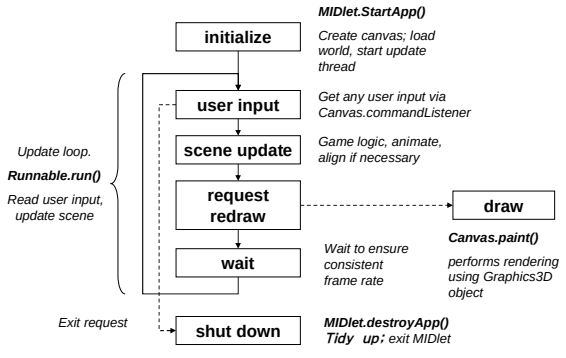
The Simplest MIDlet

- Derived from MIDlet,
- Overrides three methods



- And that's it.

A More Interesting MIDlet



Flow-chart courtesy of Sean Ellis, Superscape

MIDlet Phases

- Initialize
- Update
- Draw
- Shutdown

Initialize

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

Update

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

Update Tips

- *TIP: Don't create or release objects if possible*
- *TIP: Call* system.gc() *regularly to avoid long pauses*
- *TIP: cache any value that does not change every frame; compute only what is absolutely necessary*

Draw

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

Draw Tips

- *TIP: Don't do 2D drawing while Graphics3D is bound*

Shutdown

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

MIDlet Review

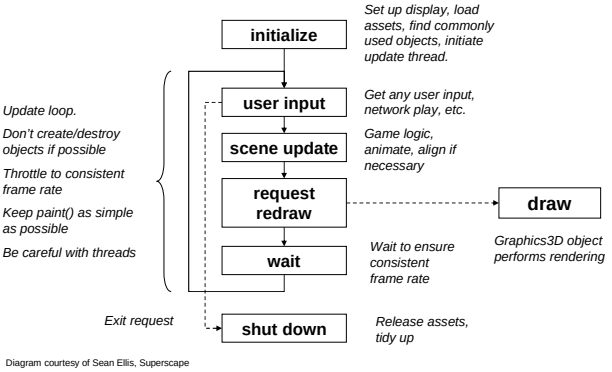


Diagram courtesy of Sean Ellis, Superscape

Agenda

- Game Development Process
- Asset Creation
- Program Development
- MIDlet Structure
- A MIDlet Example
- Challenges in Mobile Game Development
- Publishing Your Content

Demo: UsingM3G MIDlet

UsingM3G 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.

UsingM3G Framework

```
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();
 }
```

UsingM3G Framework

```
 public void pauseApp() {
   if (tcanvas.isPlaying)
     tcanvas.pausePlay();
   renderingT.yield();
   renderingT = null;
 }

 public void destroyApp(boolean u) {
   pauseApp()
   tcanvas = null;
 }
```

UsingM3G Framework

```
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();
  }
}
```

UsingM3G Initialization

```
// 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;
    }
  }
```

Loading the 3D data

```
// class TargetCanvas
void Load() throws IOException {
  loadObjs = Loader.load(fileName);
  if (loadObjs==null)
    throw new RuntimeException("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.)

```
    meshController =
      (AnimationController)gWorld.find(meshControllerId);
    morphingMeshController =
       (AnimationController)gWorld.find(morphingMeshControll
erId);
    skinnedMeshController =
       (AnimationController)gWorld.find(skinnedMeshControlle
rId);

    /* Clean up after the loading process. */
    System.gc();
 }
```

TargetCanvas *run* method

```
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

```
  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);
  }
```

TargetCanvas *render* method

```
  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() { }
}
```

Initializing CameraManip

```
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 );
      distToTarget = (float)Math.sqrt( matrix[3]*matrix[3]
                                     + matrix[7]*matrix[7]
                                     + matrix[11]*matrix[11] );

      curCamera.getTransform( curOriginalXform );
      rotate( 0, 0, 0 );
      world2Cam = null;
    }
}
```

Rotating the Camera

```
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

```
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

- Game Development Process
- Asset Creation
- Program Development
- MIDlet Structure
- A MIDlet Example
- Challenges in Mobile Game Development
- Publishing Your Content

Why Mobile Game Development is Difficult

- Application size severely limited
  - Download size limits
  - Small Heap memory
- Small screen
- Poor input devices
- Poor quality sound
- Slow system bus and memory system

Why Mobile Game Development is Difficult

- 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
  - Get makers to increase memory 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

- Game Development Process
- Asset Creation
- Program Development
- MIDlet Structure
- A MIDlet Example
- Challenges in Mobile Game Development
- Publishing Your Content

Publishing Your Content

- Can try setting up own site but
  - it will be difficult for customers to find you
  - impossible to get paid
  - may be impossible to install MIDlets from own site
- Must use a carrier approved publisher
- Publishers often run own download sites but always with link from carrier's game menu.
- As with books, publishers help with distribution and marketing

Publishing Your Content

- Typical end-user cost is $2 - $5.
- Sometimes a subscription model is used.
- Carrier provides billing services
  - Carriers in Japan take around 6%
  - Carriers in Europe have been known to demand as much as 40%! They drive away content providers.
- In some cases, only carrier approved games can be downloaded to phones
  - Enforced by handsets that only download applets OTA
  - Developers must have their handsets modified by the carrier

Publishers

- 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

Other 3D Java Mobile APIs

Mascot Capsule Micro3D Family APIs

- Motorola iDEN, Sony Ericsson, Sprint, etc.)
  - com.mascotcapsule.micro3d.v3 (V3)
- Vodafone KK JSCL
  - com.j_phone.amuse.j3d (V2), com.jblend.graphics.j3d (V3)
- Vodafone Global
  - com.vodafone.amuse.j3d (V2)
- NTT Docomo (DoJa)
  - com.nttdocomo.opt.ui.j3d (DoJa2, DoJa 3) (V2, V3)
  - com.nttdocomo.ui.graphics3D (DoJa 4) (V4)

Mascot Capsule Micro3D Version Number

Mascot Capsule V3 Game Demo

Summary

- Use standard tools to create assets
- Basic M3G MIDlet is relatively easy
- Programming 3D Games for mobile is hard
- Need good relations with carriers 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.c4d2m3g.com
  - Site appears to be defunct

Lightwave
- www.mascotcapsule.com/M3G/

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

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 Wireless Toolkit
  - java.sun.com/products/j2mewtoolkit/download-2_2.html

SDKs

- VFX SDK (Vodafone Global)
  - via.vodafone.com/vodafone/via/Home.do
- VFX & WTKforJSCL (Vodafone KK)
  - developers.vodafone.jp/dp/tool_dl/java/emu.php

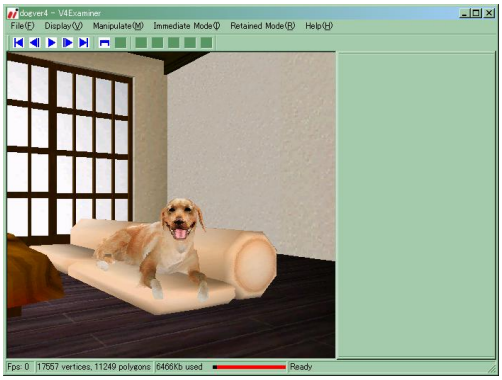
IDE's for Java Mobile

- Eclipse Open Source IDE
  - www.eclipse.org
- JBuilder 2005 Developer
  - www.borland.com/jbuilder/developer/index.html
- Sun Java Studio Mobility
  - www.sun.com/software/products/jsmobility
- Comparison of IDE's for J2ME
  - www.microjava.com/articles/J2ME_IDE_Comparison.pdf

Other Tools

- Macromedia Fireworks
  - www.adobe.com/products/fireworks/
- Adobe Photoshop
  - www.adobe.com/products/photoshop/main.html
- Sony SoundForge
  - www.sonymediasoftware.com/products/showproduct.asp?PID=961
- Steinberg Cubase
  - www.steinberg.de/33_1.html
- Yamaha SMAF Tools
  - smaf-yamaha.com/

犬友 (Dear Dog) Demo

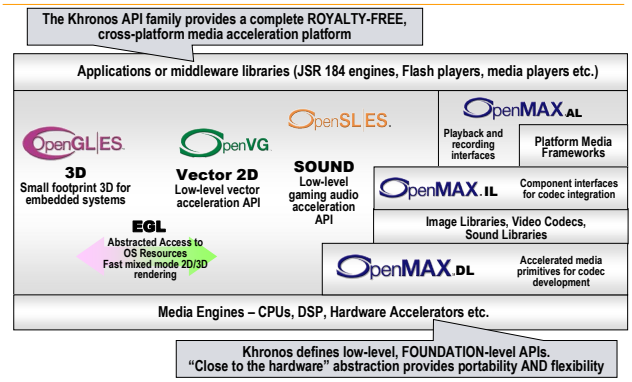


Thanks: HI Mascot Capsule Version 4 Development Team, Koichi Hatakeyama, Sean Ellis, JSR-184 Expert Group

Closing & Summary

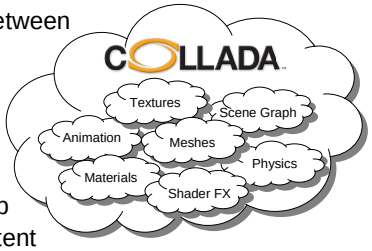
- We have covered
  - OpenGL ES
  - M3G

KHRONOS GROUP API palette



COLLADA

- 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



Shaders? Yes!

- OpenGL ES 2.0
  - subset of OpenGL 2.0, with very similar shading language
  - spec draft at SIGGRAPH 05, conformance tests summer 06, devices 08 (?)
- M3G 2.0
  - adds shaders and more to M3G 1.1
  - first Expert Group meeting June 06

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
  - work just starting

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

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
- Standards for 2D vector graphics