Understanding M3G 2.0
and its Effect on Producing Exceptional 3D Java-Based Graphics

Sean Ellis
Consultant Graphics Engineer
ARM, Maidenhead
Introduction

- M3G 1.x Recap
- ARM M3G Integration
- M3G 2.0 Update
- M3G 2.0 Hints and Tips
Audience

- Who were we addressing with M3G 1.x?

  Experienced Software Engineers

  "Technical Artists"

  3D design skill

  Java design skill

- The same people need to be able to use M3G 2.0.
Basic MIDlet Architecture

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

initialize
Load assets, find commonly used objects, set up display, initialize game state

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

scene update
Game logic, animate, align if necessary

request redraw

wait
Wait to ensure consistent frame rate

paint()
Canvas calls Graphics3D to perform rendering

shut down
Release assets, tidy up

Exit request

ARM Developers’ Conference & Design Pavilion 2007
Java with Native Library

Typical Relative Performance

Interaction from user
3D object control

In using SwV classes

FAST
Most work done here - at high performance

Scene management
Animations
3D transformation
Lighting
Triangle drawing
Texturing
Transfer to 2D

2D UI Library (J2ME)
Network Library (J2ME)
Input Library (J2ME)

M3G Classes
M3G Library

OS
Hardware

ARM Developers’ Conference & Design Pavilion 2007
Class Hierarchy

Object3D
  - Transformable
    - VertexBuffer
      - Appearance
      - Material
      - Image2D
      - Background
    - Texture2D
      - VertexArray
      - Fog
      - PolygonMode
      - AnimationTrack
    - IndexBuffer
      - CompositingMode
      - KeyframeSequence
      - AnimationController
  - Node
    - Camera
      - Light
    - Group
      - Mesh
      - SkinnedMesh
      - MorphingMesh
  - World
  - Sprite3D
  - Graphics3D
  - Loader
  - RayIntersection
  - Transform
M3G Integration – Top End

Java Application

Java Interface Layer

C Interface Layer

C++ Implementation

Some C++ libraries pre-linked

Example API implementations shipped as source

Supplied as a linkable library

API interface code is auto-generated for KNI, JNI, C

Interface Description

ARM Developers’ Conference & Design Pavilion 2007
M3G Integration – Bottom End

C Interface Layer

C++ Implementation

Some C++ libraries pre-linked

OS Abstraction

GL Abstraction

Surface Abstraction

Example implementations shipped as source

Abstraction APIs

Surface abstraction needs access to Java VM internals (pixel buffers for images and Graphics objects)
M3G 2.0 Integration Tip

- Good news, everyone – almost exactly the same as M3G 1.1
- Some software-renderer-only integration APIs removed
- New abstraction interface for DynamicImage2D
M3G 2.0 New Features

- **Core block**
  - Fixed function (OpenGL ES 1.1)
  - M3G 1.x compatible
  - Mali55™ or Mali200™

- **Advanced block**
  - Programmable hardware
  - Mali200
  - Can mix modes in API, with some restrictions
  - Can implement both blocks on OpenGL ES 2.0
  - Must be completely present or absent – less fragmentation
  - Javax.microedition.m3g.shader package is major new addition

- **Disclaimer**
  - Spec still in draft – details subject to change
New Image Types

- Image2D
- Slow path
- Fast path
- Renderer
- Java Video Control
- Native Video Source
Mesh and its components

Vertex arrays now can contain values of type byte, short, 16.16 fixed point, float, or half-float.
Mesh and its components

Mesh

- VertexBuffer
- IndexBuffer
- Appearance

Point Array
Explicit or sequential vertex indices

Index into…
Appearance and its components

- Material: Diffuse, emissive, specular, ambient colors, shininess
- Polygon Mode: Perspective correction, winding, culling
- Compositing Mode: Screen buffer write mode, thresholds, depth offset
- Fog: Fog density, color
- Texture2D: Texture application mode, coordinate transform, wrapping
- Image2D

Confidential
Appearance and its components

Shader Appearance

Shader Program

Vertex Shader

Fragment Shader

Shader Uniforms

Texture2D

TextureCube

Image2D

ImageCube
Vertex Shader

- First, vertex shader transforms the position of each vertex in an object.
- Often also calculates other parameters
Vertex Shader

- Same vertex shader runs on all vertices in the object.
- Can assign different shaders to other objects.
- Results are interpolated across primitives.
Fragment Shader

- Fragment shader takes interpolated ("varying") variables, and calculates the actual color at each pixel.
Fragment Shader

- Or it abandons the pixel, rendering nothing, and leaving a hole.
Fragment Shader

- Same fragment shader runs on all pixels in the object.
- Can assign different shaders to other objects.

You get the idea
M3G 2 Shader Model

```
uniform mat4 modelViewMatrix;
attribute vec4 vertexPos;
void main()
{
    gl_Position = modelViewMatrix * vertexPos;
}
```

Disclaimer: Specification subject to change

ARM Developers’ Conference & Design Pavilion 2007
Why keep Appearance?

- ShaderAppearance can do anything than Appearance can
- So why keep it?

- Easy transition
- More artists know fixed function pipeline than shaders.
- Perfectly adequate for a lot of content.
- Familiarity means faster development.
- Can use all the legacy content out there.
- Can mix and match both in one scene.
- Therefore, shader designers can concentrate effort where it’s going to be noticed.
Shader Example
Shader Walkthrough

- So, what does a shader look like?

```cpp
// Astoundingly dull fragment shader
uniform vec4 flatColor;
void main()
{
    gl_FragColor = flatColor;
}
```

Isn't that just C?
Same Geometry, Dull Shader
C-like language

- Syntax familiar to most programmers
- Simplified: no pointers, no characters
- New types: vectors, matrices, samplers
- New qualifiers: attribute, uniform, varying
- Limited size
- 95% common between Vertex and Fragment shaders
RenderPass

Any Node can have a...

Mesh

VertexBuffer

IndexBuffer

Appearance

Texture2D

Image2D

Renders to...

RenderPass

World

Meshes
M3G2 Shader Tips

- TANSTAAFL
- Shaders are not free – they take time, even on hardware
- Complex shaders take more time
- Specialize shaders for common parameter values
The #1 Important Shader Optimization Tip

- DON’T DO IT
- …yet

- Premature optimisation is the root of all evil – C.A.R. Hoare

- Profile!
- Is it really the shaders at all?
- How fast does my shader run?
- What’s really holding it up?
- Then you can really improve things
M3G2 Shader Tips

- Compilation time is fast, but not insignificant
- Compile once, use many times
- (This is why compilation happens at construction)
- Don’t create shaders on the fly

- It’s a balancing act
- More specialized shaders = faster execution
- More specialized shaders = slower startup
M3G2 Shader Tips

- Tools support often at the low level
- Requires cooperation from the hardware
- Mali hardware has debugging/profiling counters
- Low-level performance analysis tools
  - Framerate
  - Throughput
  - Cache usage
- Higher level analysis tools under development:
  - Shader debugger
  - Shader profiler
Animation Changes

- Content developers requested changes to animation system.
- Designed for ease-of-use.
- Details currently under discussion in expert group.
You can contribute

- Early Draft Review Spec available at:
  - http://jcp.org/aboutJava/communityprocess/edr/jsr297
- Or just go to jcp.org and enter 297 in the “Go to JSR” box. The download link is on the resulting page.

- We want to hear feedback, especially from content developers
- We want you to understand it
- We want you to use it
- We want you to make good-looking applications exceptional
Q&A

http://jcp.org/aboutJava/communityprocess/edr/jsr297