Java is a trademark of Sun Microsystems, Inc.

JavaOne℠

Developing Stunning 3D User Experiences Using M3G on Mobile

Peter Horsman

ARM Ltd.

mali™ Graphics by ARM®

Introduction

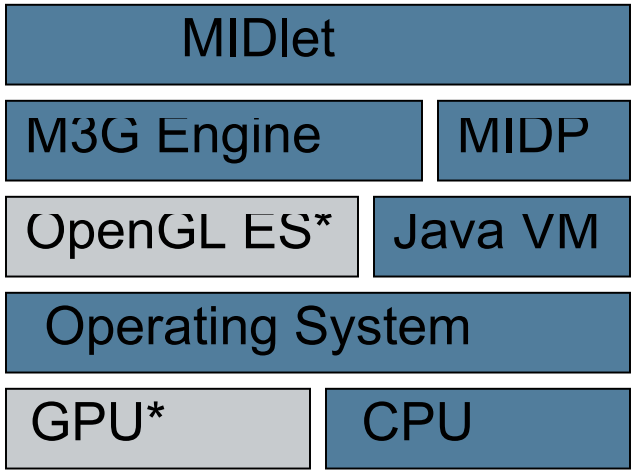
- Peter Horsman
- Staff Software Engineer
- Media Processing Division, ARM Ltd.
- M.Eng. Computer Science, Bristol UK
- Previous:
  - Implementation & integration
    - JSR-184 & JSR-297 (M3G 1.x & 2.0)
- Current:
  - Developer Relations

Contents

- M3G, past and current generations
- The application
  - Explanation
  - Code snippets
- Demo

The Current Generation

- M3G1.x very widely deployed
  - de facto mobile phone 3D gaming standard
  - Wide developer community
  - Hundreds of phone models
  - Hundreds of titles
- Aimed at:
  - OpenGL ES 1.x h/w
    - Or software renderer
  - <QCIF to >VGA
  - 10-100k Tri/s



* Optional

Handsets and Content

The Next Generation - M3G2

- Backward compatible (runs M3G1 code)
- Core block
  - Still supports OpenGL ES 1.x
- Advanced block
  - Implemented on OpenGL ES 2.0
  - Access to programmable shaders
    - Better visual effects
    - Lower resource consumption

The Application

- Simple time zone picker
  - Spinning globe
    - Won't benefit more once poly count sufficient
    - Increase shader complexity for added detail
- Helper functions
- Geometry
- World
- Texturing
- Uniforms
- ShaderAppearance
- User interaction
  - Pointer events (e.g. touch screen)

Helper Functions

- loadImage()
  - More tidy to hide try-catch block in function
- loadShader()
  - Shaders are in MIDlet's JAR as resources
  - Loaded in 1KB chunks via InputStream
  - Concatenated into a growing StringBuffer
    - (Accepted by shader constructor)
  - Hides exception handling code

Helper Functions (continued)

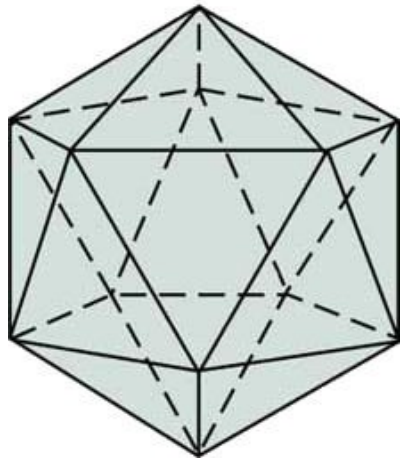
```
// Load an Image from specified location.
Image loadImage(String sLocation) {
   Image result = null;

   try {
      result = Image.createImage(sLocation);
   } catch (IOException e) {
      /* Error. */
   }

   return result;
}
```

Helper Functions (continued)

```
StringBuffer loadShader(String sLocation) {
    InputStream is = this.getClass().getResourceAsStream(sLocation);
    if(is == null) return null; // Error.

    StringBuffer result = new StringBuffer();
    int iRead = 0;
    byte b[] = new byte[1024];

    do {
        try { iRead = is.read(b); } catch (IOException e) { return null; } // Error.
        if(iRead == -1) continue;
        result.append(new String(b, 0, iRead));
    } while (read != -1);

    try { is.close(); } catch (IOException e) { /* Error. */ }

    return result;
}
```

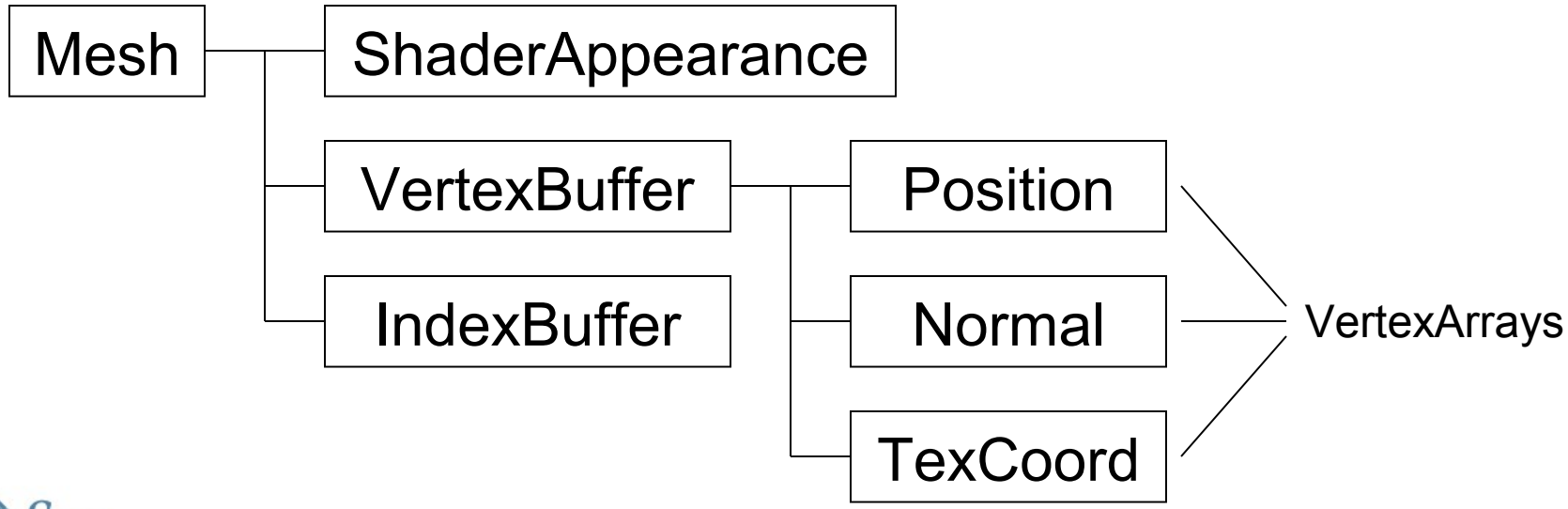
Geometry

- Vertices
  - Based on icosahedron
  - With recursion adding extra polygons
- Normals
  - Same as vertices (convenient!)
- Texture coordinates
  - Calculated from vertices



Geometry (continued)

- Create using API
  - (Could have loaded from M3G file)
- Implicit IndexBuffer
  - Describes list of triangles from vertices



Geometry (continued)

```
VertexArray myPositions = new VertexArray(iNumVertices, 3, VertexArray.FLOAT);
VertexArray myNormals = new VertexArray(iNumVertices, 3, VertexArray.FLOAT);
VertexArray myTexCoords = new VertexArray(iNumVertices, 2, VertexArray.FLOAT);
myPositions.set(0, iNumVertices, aVertex);
myNormals.set(0, iNumVertices, aNormal);
myTexCoords.set(0, iNumVertices, aTexCoord);

VertexBuffer myVB = new VertexBuffer();
myVB.setPositions(myPositions, 1.0f, null);
myVB.setNormals(myNormals);
myVB.setTexCoords(0, myTexCoords, 1.0f, null);

IndexBuffer myIB = new IndexBuffer(IndexBuffer.TRIANGLES, iNumTris, 0);

m_MeshSphere = new Mesh(1, 0);
m_MeshSphere.setVertexBuffer(myVB);
m_MeshSphere.setIndexBuffer(0, myIB);
```

World Members

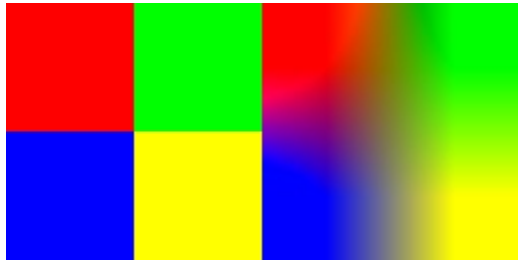
- Geometry (Mesh)
- Light
  - Simple directional light
  - Bind direction as shader uniform (see later)
- Background
  - Default gives black background
- Camera
  - Set aspect ratio from Canvas dimensions

World (continued)

```
m_World = new World();
m_World.addChild(m_MeshSphere);

m_Light = new Light();
m_Light.getTransform(m_LightTransform);
m_LightTransform.postRotate(-30.0f, 0.0f, 1.0f, 0.0f);
m_LightTransform.postRotate(30.0f, 1.0f, 0.0f, 0.0f);
m_Light.setTransform(m_LightTransform);
m_World.addChild(m_Light);

m_Camera = new Camera();
m_Camera.setPerspective(30.0f, getWidth() / (float)getHeight(), 1.0f, 1000.0f);
m_Camera.getTransform(m_CameraTransform);
m_CameraTransform.postTranslate(0.0f, 0.0f, 10.0f);
m_Camera.setTransform(m_CameraTransform);
m_World.addChild(m_Camera);
m_World.setActiveCamera(m_Camera);
```

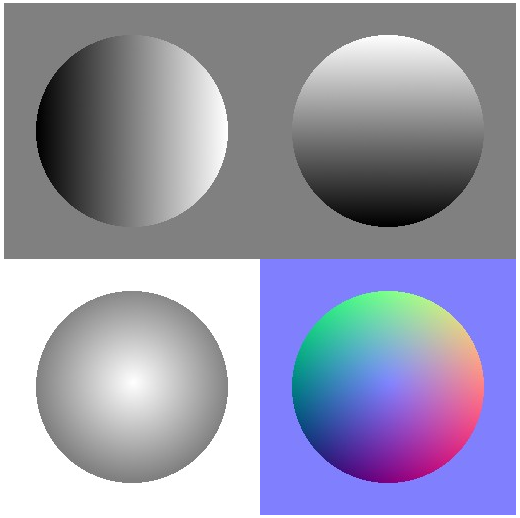
Texturing

- Bilinear filtering
  - Texture is stretched onto triangle
  - Linearly blend adjacent texels
    - In S & T

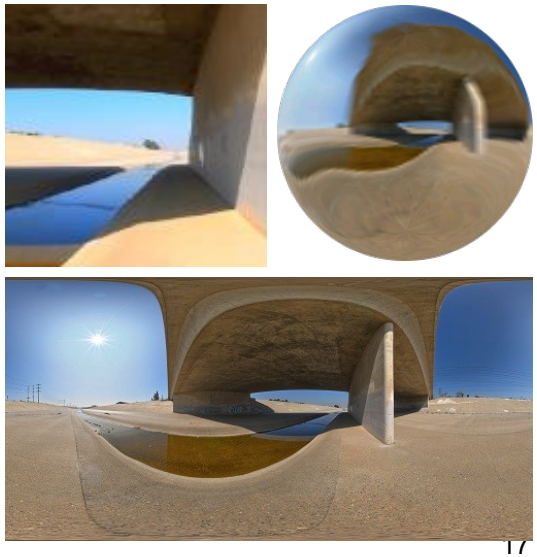


- MIP mapping
  - Texture is squished onto triangle
  - Have pre-calculated smaller textures
  - Select closest size
  - Visual quality and texture cache improvement

Shader Effects



- Bump mapping
  - Normal map in texture
  - Maths puts light and normal map in same space
  - Dot light direction with normal to give intensity



- Environment mapping
  - Spherical environment map
  - Shader gets eye-direction
  - 'reflect' this off surface normal
  - Use vector to look-up in texture

Uniforms

- Uniforms are like global variables
  - Read only
  - Visible to vert & frag shaders
  - Application modifies values
  - Don't change value during a draw call
- Examples include:
  - 'Flags' controlling shader behavior
  - Scene constants e.g. a cheat 'light' level

ShaderAppearance

- Globe texture loaded from PNG resource
  - Bound to shader uniform as Sampler2D
  - Wrap in X, clamp in Y
  - Chose bilinear filtering, with MIP mapping
- Texture geometry with Globe
- Plenty of scope for additional shader effects
  - Bump map - land high, sea low
  - Sea reflects Universe environment map
  - Vertex deformation for mountain ranges
  - Fancy lighting models

ShaderApperance (continued)

```
// Load shaders from JAR file.
m_VSS = loadShader("/shader.vert");
m_FSS = loadShader("/shader.frag");

// Set up shader appearance.
m_VS = new VertexShader(m_VSS);
m_FS = new FragmentShader(m_FSS);
m_SP = new ShaderProgram(m_VS, m_FS);

// Create the shader appearance.
m_SA = new ShaderAppearance();
m_SA.setShaderProgram(m_SP);

// Make the sphere have a shader appearence.
m_MeshSphere.setShaderAppearance(0, m_SA);
```

ShaderUniforms

```
// Load world texture.
m_ImgWorld = loadImage("/world.png");
m_TexWorld = new Texture2D(new Image2D(RGB, m_ImgWorld));
m_TexWorld.setWrapping(WRAP_REPEAT, WRAP_CLAMP);
m_TexWorld.setFiltering(FILTER_NEAREST, FILTER_LINEAR);

// Set up new uniforms object.
m_SU = new ShaderUniforms();
int iID = m_SU.addUniform("us2dTexture", SAMPLER_2D, 0);
m_SU.setUniformValue(iID, m_TexWorld);
iID = m_SU.addUniform("uv3LightDir", VEC3, 0);
m_SU.bindUniformValue(iID, m_Light, CAMERA_SPACE_DIRECTION);
m_SA.addShaderUniforms(m_SU);
```

User Interaction

- Globe is spun using pointer dragging
  - Calculate 'speed' from pixel distance over time
  - Concept of friction, so globe decelerates

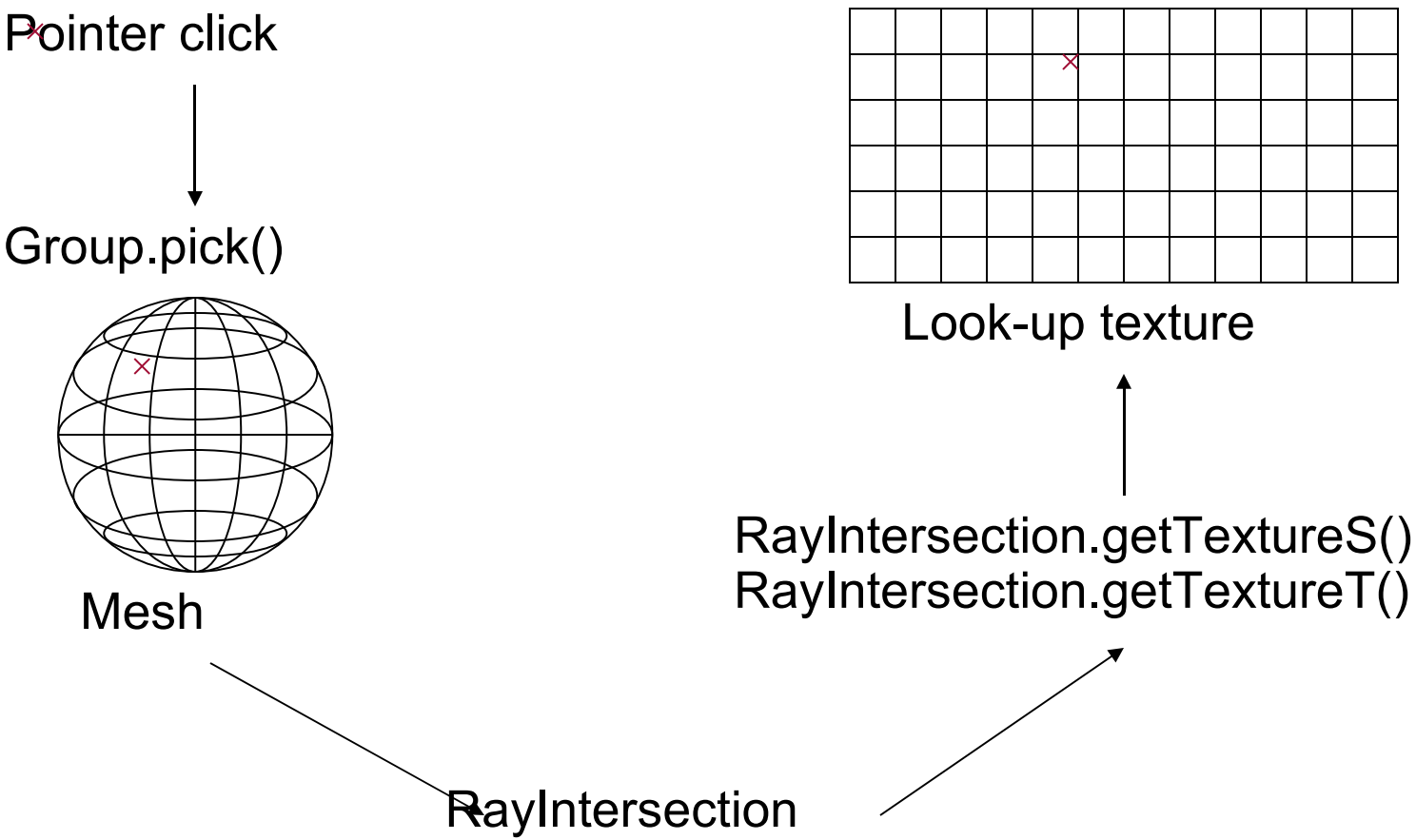
User Interaction (continued)

```
// On press, record time and X coord.
protected void pointerPressed(int iX, int iY) {
    long lNow = System.currentTimeMillis();
    m_iDragStartTime = (int)(lNow - m_lStartTime);
    m_iDragStartX = iX;
}

// On release, calculate distance over time and apply to angular speed.
protected void pointerReleased(int iX, int iY) {
    long lNow = System.currentTimeMillis();
    int iDragEndTime = (int)(lNow - m_lStartTime);
    int iDist = iX - m_iDragStartX;
    int iTime = iDragEndTime - m_iDragStartTime;
    m_fSpeed = (iDist / (float)m_iCanvasWidth * 22.5f) / (iTime / 1000.0f);
}
```

Time Zone Selection

- Time zone is selected from pointer click
  - Canvas coordinates collected
  - Empty RayIntersection created
  - Filled out by World.pick()
    - Canvas coordinates, Camera, RayIntersection as args
  - If intersection, can retrieve TexCoords
    - Identifies where on Mesh was clicked
- Simplest to have 2nd image with time zone map
  - Same shape as globe texture
  - Solves image map problem
  - Simple color look-up via RayIntersection TexCoord

Picking and RayIntersection

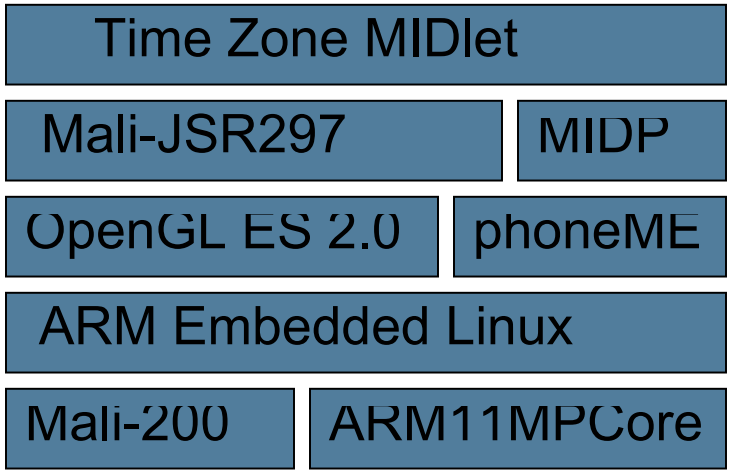


Time Zone Selection (continued)

```
protected void pointerPressed(int iX, int iY) {
    RayIntersection ray = new RayIntersection();

    int iPickX = iX / (m_CanvasWidth - 1.0f);
    int iPickY = iY / (m_CanvasHeight - 1.0f);

    boolean bResult;
    bResult = m_World.pick(-1, iPickX, iPickY, m_Camera, ray);

    if (bResult) {
        int iTX = (int)(ray.getTextureS(0) * 255.0f);
        int iTY = (int)(ray.getTextureT(0) * 255.0f);
    }
}
```

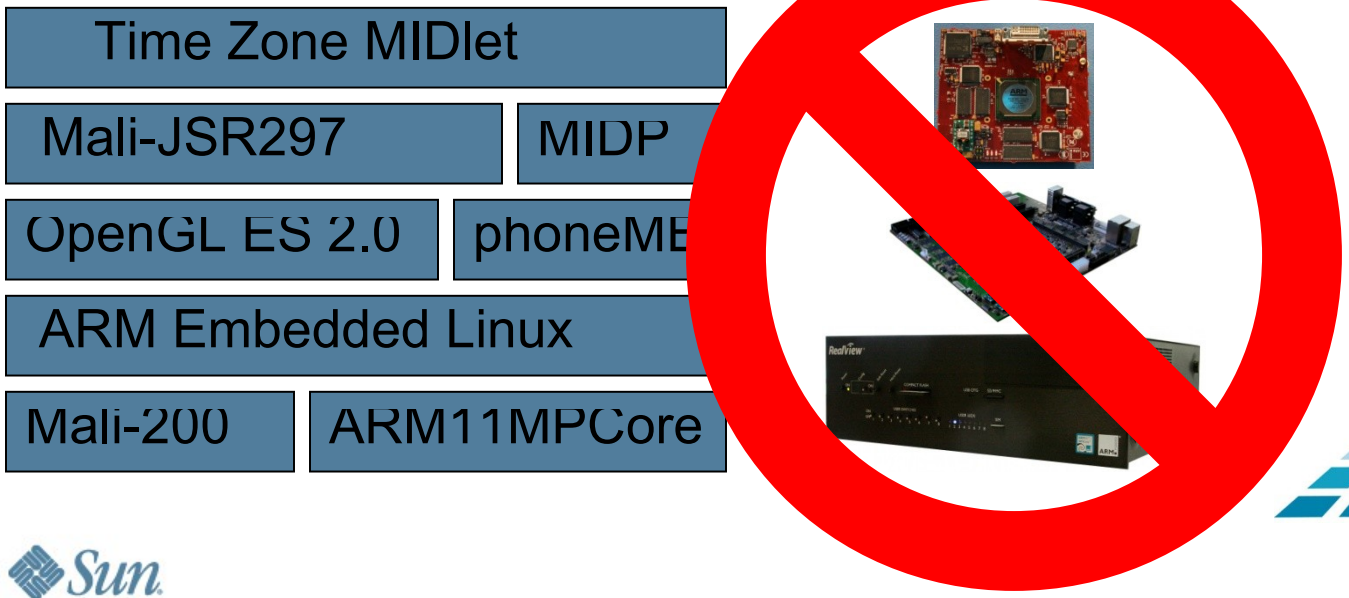
Demo

- Platform:
  - ARM's M3G2.0 implementation Mali-JSR297
  - PB11MPCore development board
  - Mali-200 test chip



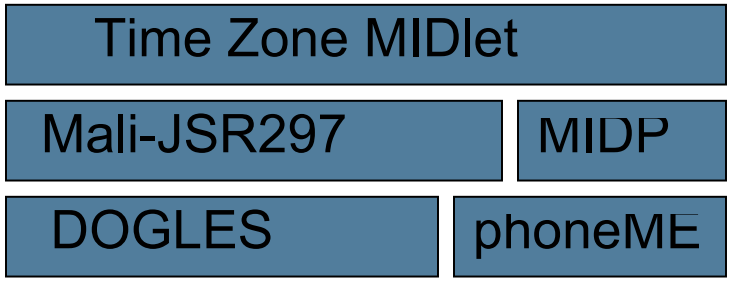
Demo

- Platform:
  - ARM's M3G2.0 implementation Mali-JSR297
  - PB11MPCore development board
  - Mali-200 test chip



Demo

- Platform:
  - ARM's M3G2.0 implementation Mali-JSR297
  - x86 Windows laptop with OpenGL 2.0 GPU
  - ARM's DOGLES (Desktop OpenGL-ES)
    - OpenGL-ES 2.0 to OpenGL 2.0 wrapper



Summary

- M3G2.0 publicly ratified
- Handsets deploying Q4'09
- Applications beyond gaming

Thank You

Peter Horsman
ARM Ltd.

Slide Heading: 36pt

Subhead: 28pt

- All text is **Arial**
- Level One bullet point: 28pt
  - Level Two bullet: 26pt
    - Level Three: 22pt
      - Level Four and subsequent: 18pt
- Text block is aligned to the left