



SIGGRAPH2005

Using M3G

Mark Callow
Chief Architect



Agenda

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

M3G Game Demo



EXTREME AIR SNOWBOARDING™ SUMEA



Copyright 2005, Digital Chocolate Inc.

Game Development Process



SIGGRAPH2005

- Traditional Java Game

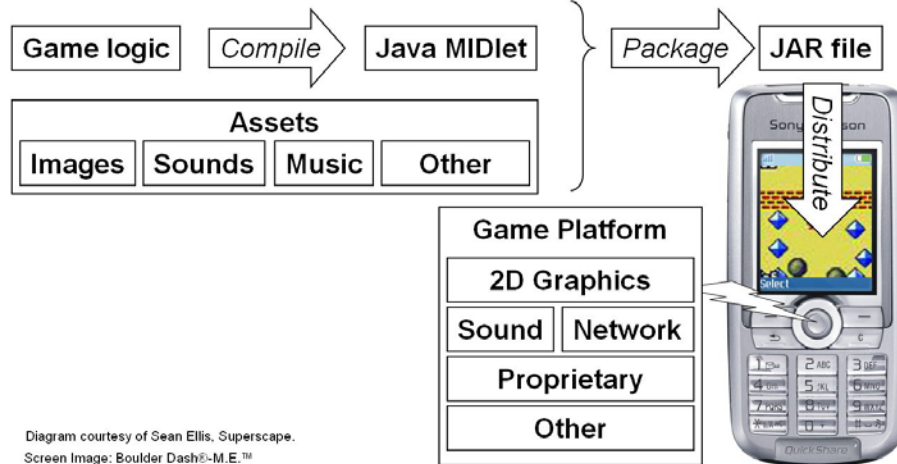


Diagram courtesy of Sean Ellis, Superscape.
Screen Image: Boulder Dash®-M.E.™

Let's have a quick look at the various steps involved in creating a traditional Java game. We have a game platform such as MIDP 2 in the mobile device. We need to write our game code targeted for this platform and compile it to a MIDlet. We package this into a JAR file together with the game assets such as images, sounds and music. Finally we distribute the game package to the customers.

I'll be discussing each of these steps during the presentation.

M3G Development Process



- How M3G Fits

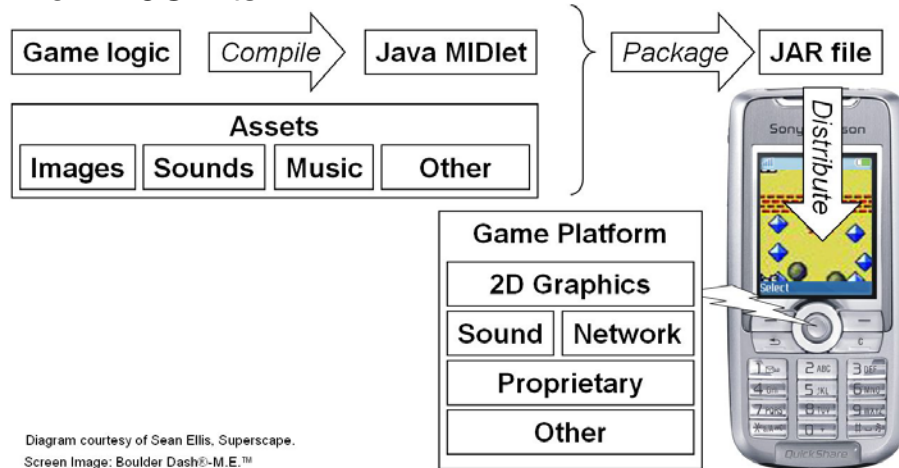
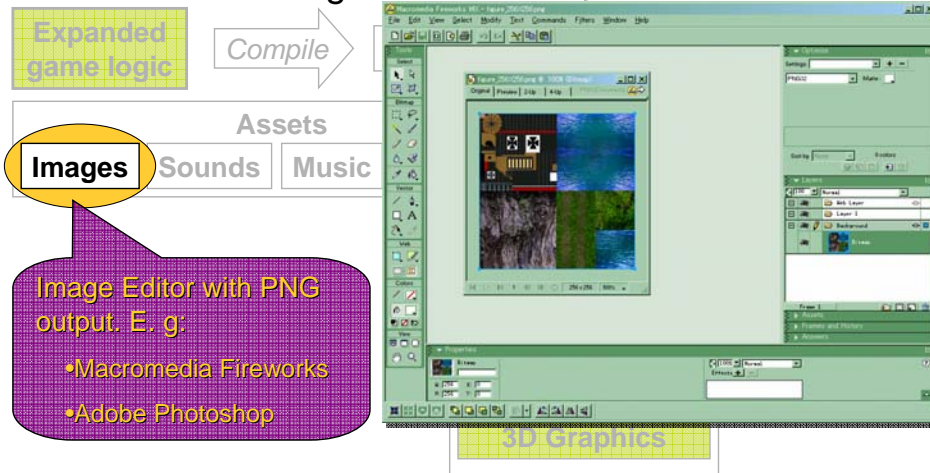


Diagram courtesy of Sean Ellis, Superscape.
Screen Image: Boulder Dash®-M.E.™

Now what does M3G bring to the party? First and foremost of course is 3D graphics. This means your assets will include 3d models or a 3d scene. You also have the opportunity to expand your game logic. Effective use of 3D influences all aspects of a game's design and must be considered from the beginning of the design process.

Asset Creation

- Textures & Backgrounds



For any real m3g application, some art assets have to be created before the program can do anything useful. So let's look first at creating the assets and then at the programming.

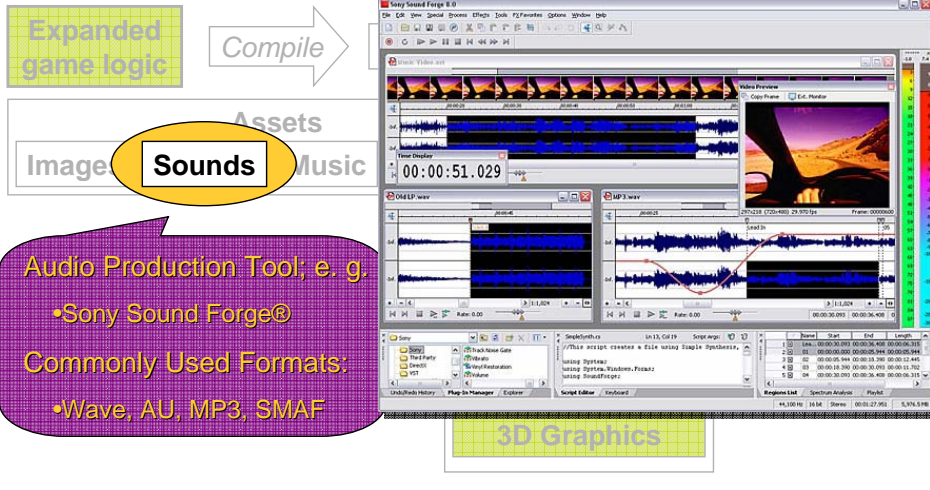
Textures and background images can be provided as PNG format files or the image data can be included directly in an M3G file. We recommend creating these assets in PNG format. PNG compresses better than plain zlib.

Some M3G plug-ins for 3d modeling tools automatically convert texture maps to PNG format. If so, you can use any texture map format supported by your modeling tool.

Do not use GIF files. Some M3G implementations appear to support GIF files as an accidental side-effect of the underlying MIDP implementation. Do not be fooled. The spec. does not require GIF support and many implementations do not support the format.

Asset Creation

- Audio Tools

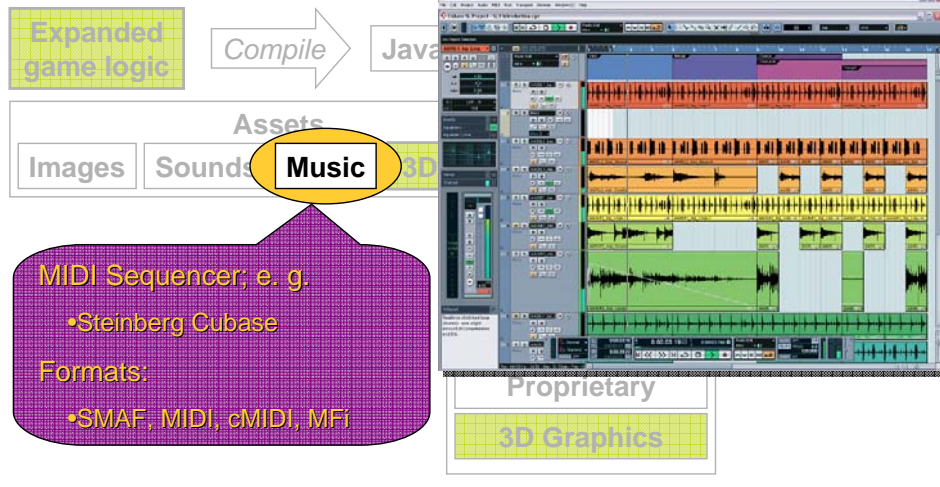


The J2ME (MIDP 2.0) specification does not seem to have a list of formats for which support is required. The formats listed here are commonly used.

SMAF (Synthetic music Mobile Application Format) is a Yamaha invented format directly supported by chips used in many handheld portable devices. The file extension is .mmf. SMAF files can have contain both recorded audio and synthesizer sequences.

Asset Creation

- Music Tools



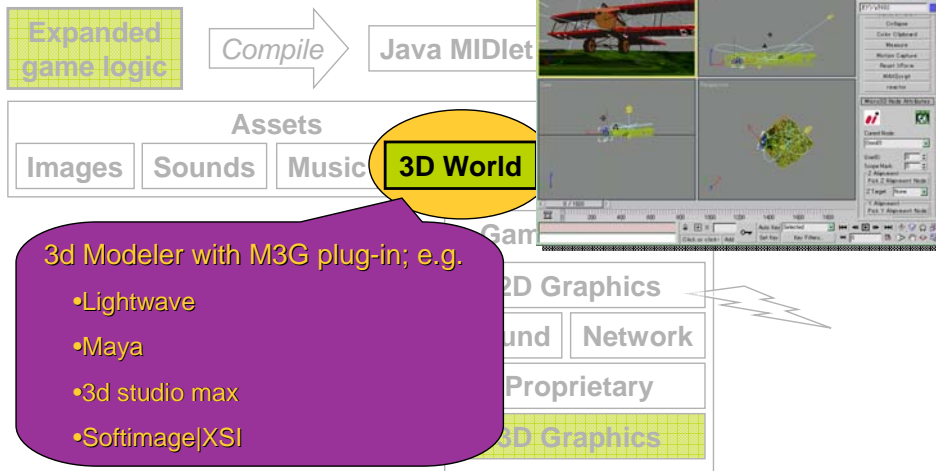
MFi (Melody Format for i-Mode) is supported on all i-Mode phones worldwide. As with SMAF, MFi can hold both MIDI-like data (cMIDI) and custom samples.

cMIDI is compact MIDI which reduces the range of allowed MIDI data thereby reducing the file size.

For all of your audio , you will mostly be dealing with hardware designed for ring tones. It is important that you understand the capabilities of the chip in your target phone.

Asset Creation

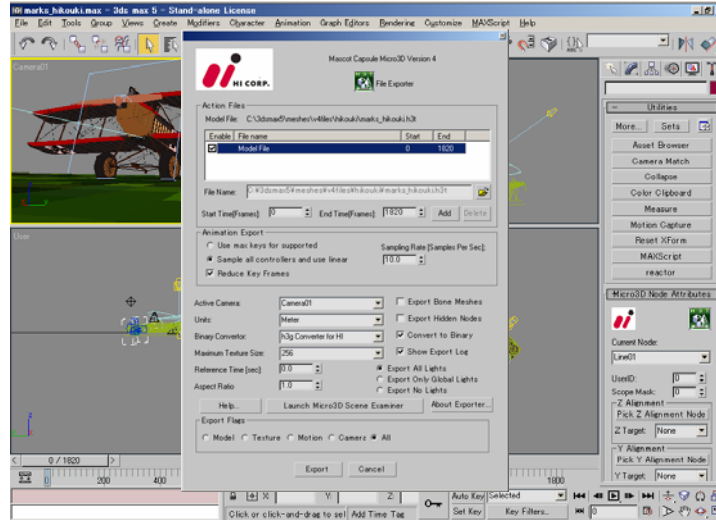
- 3D Models



A beta version of HI's SoftImage|XSI plug-in is expected by the end of August.

A new release of HI's 3d studio max plug-in is also expected by the end of August.

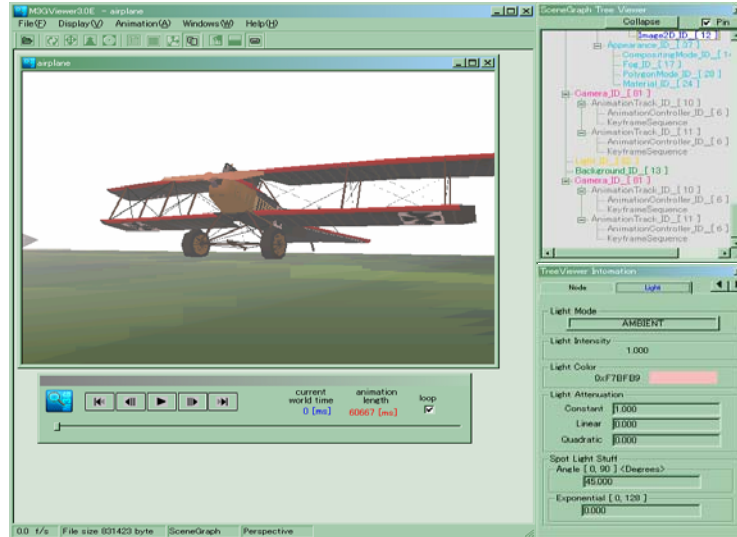
Demo: Export 3d Model to M3G



Demo: M3G File Check



SIGGRAPH2005



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*

Since we are looking at creating the 3D model assets, this is a good time for some tips for designers.

As mentioned earlier, when designing sound it is important to be aware of the capabilities of the target phone. Since these vary widely, it is best to create the original audio assets at the best possible quality.

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*



SIGGRAPH2005

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*

Sprites may not be faster than textured quads when a GPU is used for rendering.

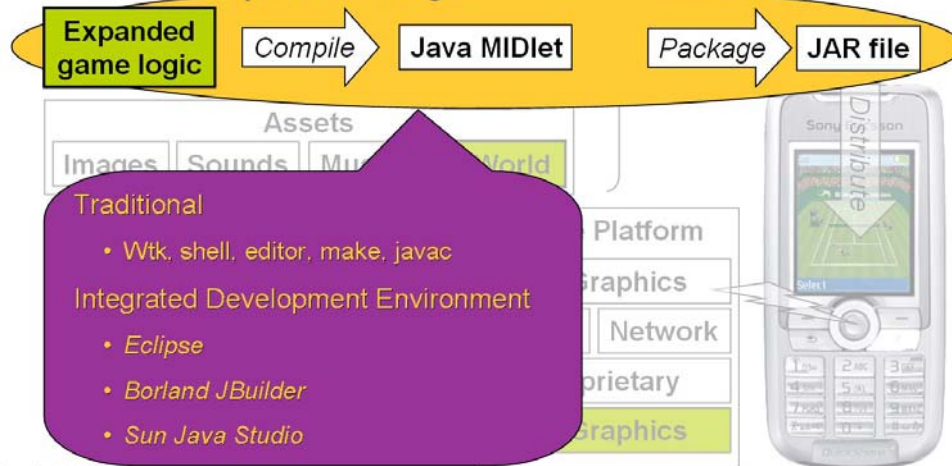
In some implementations `Loader.load("/img.png")` will load the image file via a MIDP image because native code is unable to read from a java stream. This requires more memory during loading.

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



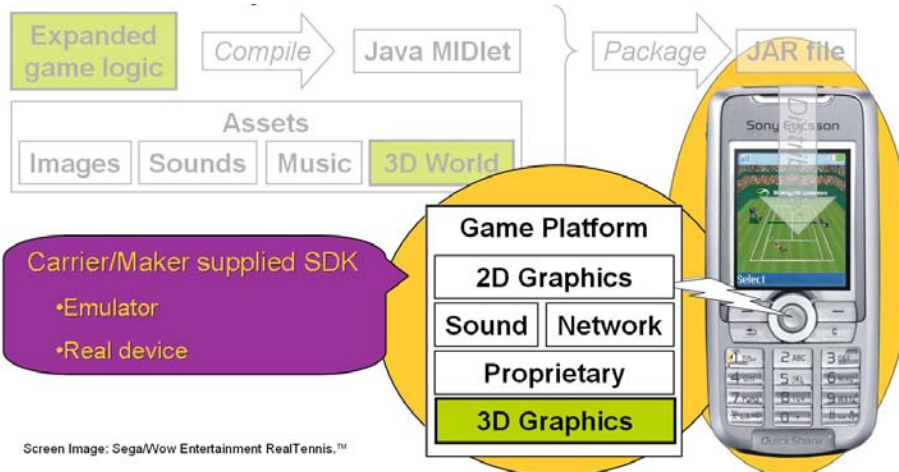
For the edit, compile build cycle you can use a traditional pipeline with a command line shell, programmer's editor, make and the standard java compiler from JDK 1.4.x.

You can also use Sun's Wireless Tool Kit, or similar, which saves you from having to write a make file and let's you build your MIDlet with the push of a button.

Alternatively you can use a full IDE such as Borland's JBuilder or Sun's Java Studio.

Program Development

- Test & Debug



For testing and debugging you need to use an SDK supplied by either the carrier or the handset maker. These SDKs contain an “emulator”, usually a PC application that provides the functional environment of the real device. In at least one case, Sony Ericsson, the SDK includes a way to link to a real handset allowing applications to be tested and debugged on the real device. This is the ideal arrangement.

Sun’s J2ME Wireless Toolkit (WTK 2.2) provides a generic emulator for MIDP/CLDC. Two are at least two known problems with this emulator.

- It will load GIF files as textures. This is permitted but not required by the M3G spec. As I noted earlier, you should avoid GIF files.
- It will not load M3G files that encode KeyframeSequence values as short. They must be encoded as float.

It is important to note that “emulators” can be quite different from the real devices. In many cases a different JVM and a different 3D renderer are used in the “emulator” than in the real device. There is typically no relationship between performance in an “emulator” and performance on the real device.

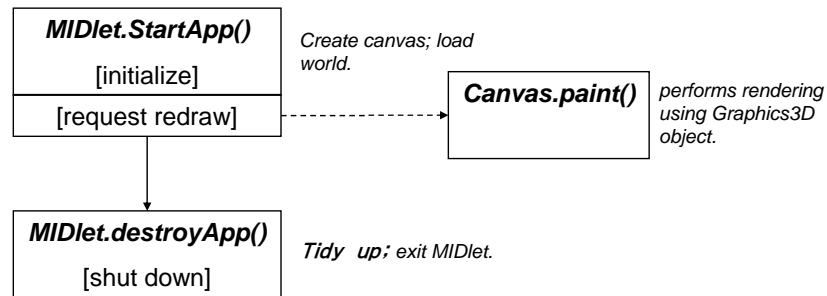
Agenda

- 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.

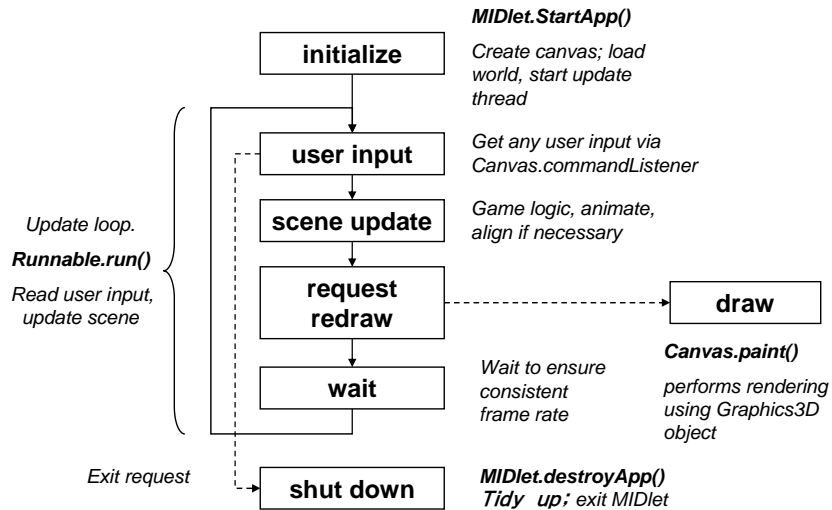
We've looked at creating assets and tools to use writing and debugging the programs. What does an actual program look like? Here we'll look at the structure of a MIDlet, beginning with the simplest possible example? It's a class derived from MIDlet that overrides just 3 methods.

startApp just creates a canvas for display and loads the world to display; it requests a redraw which results in the overridden paint method being called which renders a view to the screen. destroyApp does some tidying up. And that's it. Of course, that's not very interesting. We don't get any updates, and the display is static, but it shows the absolute basics. By modifying the world and repainting, you can easily create animated 3D scenes. Let's have a look at the structure of a MIDlet with an update loop.

A More Interesting MIDlet



SIGGRAPH2005



Flow-chart courtesy of Sean Ellis, Superscape

So, here's the diagram updated to show the main update loop. The MIDlet implements the Runnable interface, which means providing one more method, *run()* which contains the update loop.

The update loop reads user input, updates the scene, requests a redraw and then waits until the next frame is scheduled. Waiting ensures a consistent frame rate.

MIDlet Phases

- Initialize
- Update
- Draw
- Shutdown

Let's look at each of these phases in more detail.



SIGGRAPH2005

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

Initialization gets us into a state where we can start the game. First, we load all the assets we need, both for the 3D scene and any other UI elements, music, sounds, etc. We should then look up any frequently used objects in the World, to save time in the main game loop. For example, we can find the player's object, any non-player characters, etc. Of course, we need to initialize anything that the actual game logic requires (monster strengths, high-score tables, network connections to other players, or whatever). Then we initialize the display, and the timers we use to drive the main update loop, and kick off our first update.

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

The update is usually attached to timer and other events. Obviously, we need to respond to the user, so getting any input from them is the first thing to do, and get the current time. We get the current time once to avoid problems if the various steps here take significant time. The next thing to do is to run the game logic based on the user input. While this will be different for each game, the net effect of this is that it updates the state of objects in the world as necessary. Opened a door? Rotate the door object. Picked up a health bonus? Make it invisible, update your health, change size of health bar. One tip here that works well is to divorce the logic from the representation. Instead of rotating the door object to open it, just start the “Open Door” animation. This creates fewer dependencies between the assets and the logic, and allows the asset designers to use rotating, sliding, dilating or exploding doors as they see fit. Call animate to ensure that any animations actually run, then request a redraw.



SIGGRAPH2005

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 may not change every frame; compute only what is absolutely necessary*

If at all possible, don't create or release objects in the main loop. If you do have to do this, call `system.gc()` regularly to ensure that you don't get large garbage collections that ruin the flow of the game. Cache any values that are not changing every frame in order to avoid unnecessary recomputation.

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

After each update, we request a redraw. This usually results in a call to an overridden paint method on a canvas. This is fairly simple – we just need to bind the Graphics3D to the screen, render the world, and release it. Remember that there is only one Graphics3D so we need to release it whatever happens! (The best way to do this is in a finally clause.) Then we can do any 2D UI drawing (score, health, etc) and request another update in an appropriate amount of time.

Draw Tips



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

One restriction is that you can't do 2D drawing while the Graphics3D is bound to the screen, so you have to do it either before or after (or both).



SIGGRAPH2005

Shutdown

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

On shutdown, we just need to tidy up. It's usually friendly to ensure that the Graphics3D really has been released before exiting. This should also happen if a call is made to pauseApp, since the new application that is taking over the screen may also need to use 3D.

MIDlet Review

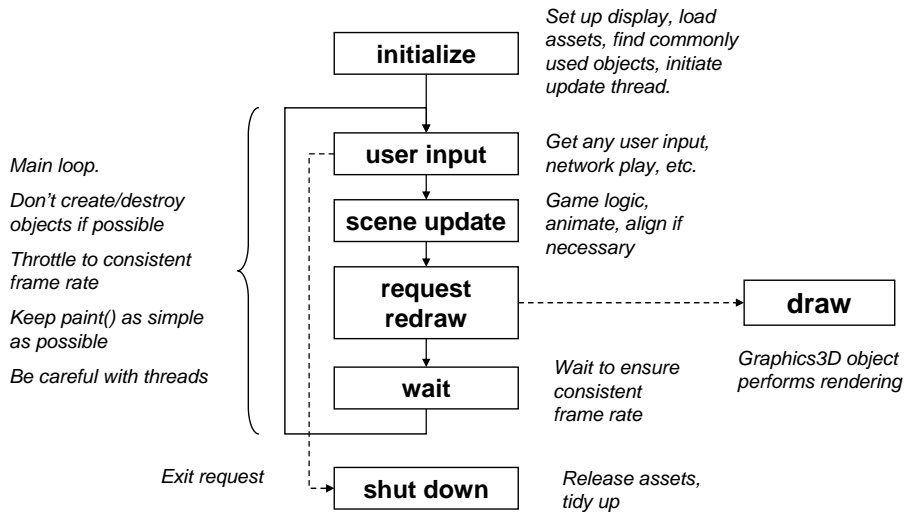


Diagram courtesy of Sean Ellis, Superscape

So, here's a diagram recapping what we have learned. Note that if nothing is happening, we don't need to continually redraw the screen – this will reduce processor load and extend battery life. Similarly, simple scenes on powerful hardware may run very fast; by throttling the framerate to something reasonable, we extend battery life and are more friendly to background processes.

Let's look at a real example

Agenda

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

Demo: UsingM3G MIDlet



Let's have a look at the MIDlet in action before diving into the code.

***UsingM3G* MIDlet**

- Display Mesh, MorphingMesh and SkinnedMesh
- Meshes loaded from .m3g files
- View can be changed with arrow keys
- Animation of individual meshes can be stopped and started.
- Animation can be stopped and started with a button push.
- 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();
    }
}
```

We've called our MIDlet class Cans. Here we see the override of startApp(). It initializes everything then kicks off the rendering thread. We can also see the declaration & initialization of a couple of command objects to handle our commads.

Thread.start() calls the thread's run() method. We'll look at that a bit later.

UsingM3G Framework



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

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

Here are the overrides of pauseApp() & destroyApp(). They are very similar.

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();
    }
}
```

Here we see how the command notifications are handled. These are the commands we initialized at the beginning of the class definition. We'll see how to set up command processing in the next slide.

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);
}
```

This initializes the canvas, and sets up a command listener to which it adds the exit & play-pause commands.

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;
    }
}
```

Now we begin to see the real work of initialization beginning to take place. The canvas constructor loads the 3d data and creates a CameraManip object. This handles rotation of the scene-graph camera.

Note that TargetCanvas extends Canvas not GameCanvas because GameCanvas swallows key strokes from the number keys and we use the number keys to as controls. It also implements the Runnable interface so we can run it from a Thread.

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?");
}
```

This method loads the m3g file using the M3G Loader and verifies that it contains a World node.

Loading the 3D Data (Cont.)



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

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

After loading the file successfully, the Load method finds some scene-graph objects we'll need while the MIDlet is running.

The two methods we've just examined are from the TargetCanvas class. Now let's look at the rest of it.

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) { }
        //}
    }
}
```

This is the Thread's run method, the heartbeat of the MIDlet.

Basically we have an infinite loop. First it checks the input at which point the scene may be modified. Then it initiates rendering by requesting a repaint. After this the thread will sleep, provided rendering the frame did not take too long.

An alternative option is to just increment the render time and return to the top of the loop. This is very unfriendly to the system but is necessary in order to measure the true frame rate.

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);
}
```

Here's the paint method. We bind the graphics to the rendering target, set up the viewport and then render the 3D scene. Note that we make sure to always call `releaseTarget()`. After that we use the 2D api to draw the frame rate.

TargetCanvas *render* method



```
void render()
{
    if (isPlaying) {
        frameCount++;
        fps = (int)((1000*frameCount) / renderTime) ;
        /* update the scene */
        gWorld.animate((int)renderTime);
    }
    g3d.render(gWorld);
}
```

Here's the render method. It's very simple. The world is animated to update everything to the current renderTime, then it is rendered.

Receiving Key Presses



```
protected void keyPressed(int keyCode) {  
    int action;  
    switch (keyCode) {  
        case KEY_NUM1: animState ^= MESH_ANIM; break;  
        case KEY_NUM2: animState ^= SKINM_ANIM; break;  
        case KEY_NUM3: animState ^= MORPHM_ANIM; break;  
        default: {  
            action = getGameAction(keyCode);  
            switch (action) {  
                case DOWN: keyState |= DOWN_PRESSED; break;  
                case LEFT: keyState |= LEFT_PRESSED; break;  
                case RIGHT: keyState |= RIGHT_PRESSED; break;  
                case UP: keyState |= UP_PRESSED; break;  
                default: break;  
            } } } }  
}
```

Here's the override of keyPressed. First the keyCode is checked and flags are set in the instance variable animState to indicate which parts of the scene should be animated. This handles presses of the 1, 2 & 3 keys.

Then the method checks for possible game actions and sets flags in the instance variable keyState when actions of interest are pressed.

Now let's check what we do with keyState & animState.



HandleInput method

```
protected void handleInput()
{
    int start = 0, end;
    int keyState = getKeyStates();
    int deltaX = 0, deltaY = 0;

    /* Stop & start animation of individual objects by setting
     * active interval on AnimationControllers.
     */
    if (meshController != null) {
        end = (animState & MESH_ANIM) != 0 ? 0 : 1;
        meshController.setActiveInterval(start, end);
    }
    if (skinnedMeshController != null) {
        end = (animState & SKINM_ANIM) != 0 ? 0 : 1;
        skinnedMeshController.setActiveInterval(start, end);
    }
    if (morphingMeshController != null) {
        end = (animState & MORPHM_ANIM) != 0 ? 0 : 1;
        morphingMeshController.setActiveInterval(start, end);
    }
}
```

getKeyStates works like the similarly named function on a GameCanvas. It returns the value of instance variable keyState clears it.

The animState flags are checked and the corresponding AnimationController's activeInterval is set to enable or disable the animation of the controlled objects. When start == end, the controller is always active.

***HandleInput* method (Cont.)**



SIGGRAPH2005

```
if ((keyState & DOWN_PRESSED) != 0) {
    deltaY -= DELTA;
}
if ((keyState & LEFT_PRESSED) != 0) {
    deltaX += DELTA;
}
if ((keyState & RIGHT_PRESSED) != 0) {
    deltaX -= DELTA;
}
if ((keyState & UP_PRESSED) != 0) {
    deltaY += DELTA;
}
if (deltaX != 0 || deltaY != 0)
    cameraManip.rotate( deltaY, deltaX, 0 );
}
```

Then the keyState is checked and rotation delta values are incremented or decremented as appropriate. Up & down rotate around the x axis. Left & right rotate around the Y axis. The delta values are passed to the CameraManip's rotate method which carries out the camera rotation.

Study of CameraManip is left as a class exercise.

Agenda

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

Now we'll move on to look at the special challenges of developing games for mobile phone handsets.

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

Download size limits are increasing thanks to 3G but 256k is still a common size limit.

Poor Input Devices: Input devices are typically limited to the 12 key-pad plus a navigation array and a few extra buttons. Yes game console style pads are coming but they are still the rare exception.

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

What is most important in the game is the operation, which functions as a communication line between the game and the player. Even within the same group of the mobile terminals, the sense of operation differs by how the buttons are placed, which as a result changes the difficulty of the game itself. These issues must be considered very carefully from the planning stage.

When porting onto other types of terminals, game operation is one of the items that generates problems in the development. For example, diagonal input may have worked on the original mobile terminal whereas it may be unavailable on the mobile terminal to which the game is being ported. Also there are some cases where terminals fails to recognize more than one button being pressed at the same time.

We cannot provide you with overall solution; however, I would like to introduce you some examples on how we coped with these issues in our past contents.

- 1) Types of mobile terminals can be discerned to diversify the difficulty of the contents.
- 2) Let the player play in a lower difficulty level when diagonal input is ineffective by keeping a diagonal input flag in the program. When the diagonal input becomes effective, then the game can switch to its normal level of the difficulty.
- 3) Allocate the same features, such as “jump” and “attack” to multiple buttons or embed a key customize feature.

With these countermeasures, the problems can be alleviated to a certain extent. Depending on the types of the game, I surmise there may be more efficient way to solve the problem. So this is where planners and programmers can leverage their ideas.

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

This section describes the situation for the mobile phone market.

Don't even think about non-over-the-air distribution for mobile. It's not the way mobile works. Some carriers have MIDlet downloads from PC's disabled in their handsets.

Some carriers disable MIDlet downloads from anywhere but their own web sites. The villains may mostly be Japanese carriers. Perhaps the anti-monopoly authorities are more effective in other parts of the world. Vodafone KK does both of these things and, reportedly SIM-locks their handsets.

The bottom line is you must use a carrier approved publisher.

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

Common subscription model is a flat monthly fee for access to the publisher's entire game library.

Game add-ons are often used. For example, connection to game site to record high scores, chat with fellow players etc. Some sites even sell game upgrades (either for points won in the game or for cash) that will help you do better. A motorcycle racing game for example provides upgrades that make the bikes go faster.



SIGGRAPH2005

Publishers

- Find a publisher and build a good relationship with them
- **Japan:** Square Enix, Bandai Networks, Sega, Namco, Infocom, etc.
- **America:** Bandai America, Digital Chocolate, Jamdat, MForma, Glu Mobile (formerly Sorrent)
- **Europe:** Digital Chocolate, Superscape, Glu Mobile (formerly 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 (DoJa 2, DoJa 3) (V2, V3)
 - com.nttdocomo.ui.graphics3D (DoJa 4) (V4)

Mascot Capsule Micro3D Version Number

For sake of completeness, I'll mention some other 3D Java APIs you will find on various mobile devices. These are all based on HI's Mascot Capsule Micro3D Engine. Mascot Capsule Micro3D Version 3 pre-dates M3G by 1 year. Version 4 supports M3G. The APIs above are found on many handsets.

Mascot Capsule V3 Game Demo



DEEP LABYRINTH[®] DELUXE EDITION



Copyright 2005, by Interactive Brains, Co., Ltd.


Just because it's a really cool game...

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


Cinema 4D

- www.c4d2m3g.com

Lightwave

- www.mascotcapsule.com/M3G/ 

Maya

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

Blender

- www.bight.ca

Softimage|XSI

- www.mascotcapsule.com/M3G/ 

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)
 - <http://via.vodafone.com/vodafone/via/Home.do>
- VFX & WTKforJSCL (Vodafone KK)
 - http://developers.vodafone.jp/dp/tool_dl/java/emu.php

Vodafone global requires you become a partner of Via Vodafone. You have to submit a questionnaire before they will even talk to you. Very unfriendly.

Vodafone KK is a little more friendly. You just need to complete a simple registration before you can download the SDK. But the web page is in Japanese. There are 2 SDKs. VFX is Vodafone Global's SDK. WTKforJSCL has JSCL instead of M3G. Both are based on Sun's Wireless Toolkit (WTK).

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

Although Eclipse is largely written in Java and has many java development tools, it is not clear at the time of writing that Eclipse has a specific set of tools for supporting J2ME.

Sun Java Studio Mobility is available at no cost by “simply register[ing] for a Sun online account”.

Other Tools

- Macromedia Fireworks
 - <http://www.macromedia.com/software/fireworks/>
- Adobe Photoshop
 - <http://www.adobe.com/products/photoshop/main.html>
- Sony SoundForge
 - <http://www.sonymediasoftware.com/products/showproduct.asp?PID=961>
- Steinberg Cubase
 - http://www.steinberg.de/ProductPage_sb4b2a.html?Product_ID=2124&Langue_ID=4
- Yamaha SMAF Tools
 - <http://smaf-yamaha.com/>

Publishers, Japan

- Square Enix
 - <http://www.square-enix.com/jp>
- Bandai Networks
 - <http://www.bandai-net.com/>
- Sega
 - <http://www.sega.co.jp/>
- Namco
 - <http://www.namco.com>
- Infocom
 - <http://www.infocom.co.jp/>



SIGGRAPH2005

Publishers, America

- Bandai America
 - <http://www.bandai.com/>
- Digital Chocolate
 - <http://www.digitalchocolate.com/>
- Jamdat
 - <http://www.jamdat.com>
- MForma
 - <http://www.mforma.com/>
- Glu Mobile
 - <http://www.glumobile.com/>

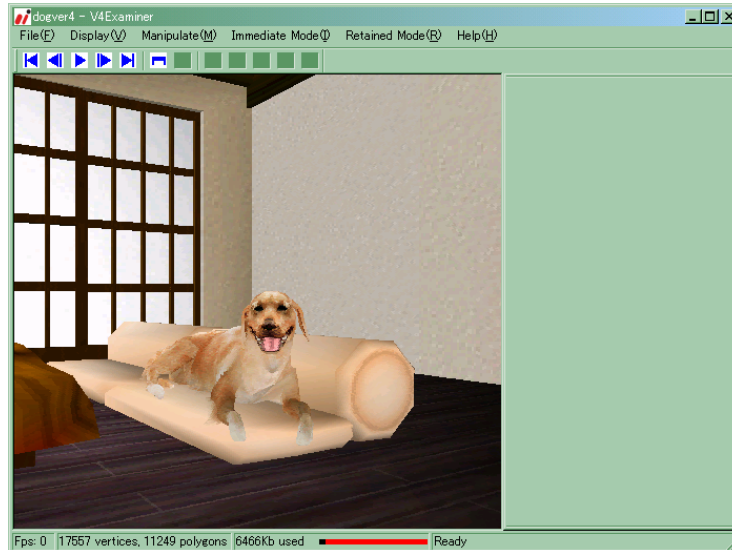
Publishers, Europe

- Digital Chocolate
 - <http://www.digitalchocolate.com/>
- Superscape
 - <http://www.superscape.com/>
- Glu Mobile
 - <http://www.glu.com>
- Upstart Games
 - <http://www.upstartgames.com/>

InuTomo (Dog Friend) Demo



SIGGRAPH2005



While I take your questions, I'll leave a final demo running. We created this to show the richness that is technically possible with M3G. Unfortunately this particular animation is too big to load into a real phone ... today.



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

Demonstrate dog animation