



**MOTODEV**

*The Motorola developer network*

## **3D PROGRAMMING— LOADING M3G FILES *and* PLAYING ANIMATIONS**

**TECHNICAL ARTICLE**



## Copyright

Copyright © 2006, Motorola, Inc. All rights reserved. This documentation may be printed and copied solely for use in developing products for Motorola products. In addition, two (2) copies of this documentation may be made for archival and backup purposes. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form or by any means or used to make any derivative work (such as translation, transformation, or adaptation) without express written consent from Motorola, Inc.

Motorola reserves the right to make changes without notice to any products or services described herein. "Typical" parameters, which may be provided in Motorola Data sheets and/or specifications, can and do vary in different applications and actual performance may vary. Customer's technical experts will validate all "Typicals" for each customer application.

Motorola makes no warranty in regard to the products or services contained herein. Implied warranties, including without limitation, the implied warranties of merchantability and fitness for a particular purpose, are given only if specifically required by applicable law. Otherwise, they are specifically excluded.

No warranty is made as to coverage, availability, or grade of service provided by the products or services, whether through a service provider or otherwise. No warranty is made that the software will meet your requirements or will work in combination with any hardware or application software products provided by third parties, that the operation of the software products will be uninterrupted or error free, or that all defects in the software products will be corrected.

In no event shall Motorola be liable, whether in contract or tort (including negligence), for any damages resulting from use of a product or service described herein, or for any indirect, incidental, special or consequential damages of any kind, or loss of revenue or profits, loss of business, loss of information or data, or other financial loss arising out of or in connection with the ability or inability to use the Products, to the full extent these damages may be disclaimed by law.

Some states and other jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, or limitation on the length of an implied warranty, therefore the above limitations or exclusions may not apply to you.

This warranty gives you specific legal rights, and you may also have other rights, which vary from jurisdiction to jurisdiction.

Motorola products or services are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product or service could create a situation where personal injury or death may occur.

Should the buyer purchase or use Motorola products or services for any such unintended or unauthorized application, the buyer shall release, indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the designing or manufacturing of the product or service.

Motorola recommends that if you are not the author or creator of the graphics, video, or sound, you obtain sufficient license rights, including the rights under all patents, trademarks, trade names, copyrights, and other third party proprietary rights.

If this documentation is provided on compact disc, the other software and documentation on the compact disc are subject to the license agreement accompanying the compact disc. 3D Programming - Load M3G and Animation

January, 2007

For the latest version of this document, visit <http://developer.motorola.com>

3d Programming – Loading M3g Files and Playing Animations

Motorola, Inc.

<http://www.motorola.com>



## Contents

<b>About this Document</b>	<b>4</b>
Document Overview	4
Definitions, Abbreviations, Acronyms	4
<b>Loading 3D content from an M3G file</b>	<b>5</b>
Loader class	5
Loading an M3G file	5
Display World Scene	6
<b>Playing an Animation</b>	<b>8</b>
The animate() function	8
Updating an Animation	8
Conclusion	11
References	11

## List of Figures

Figure 1: Loading the M3G File	7
Figure 2: Playing the animation	11



## About this Document

### Document Overview

- **Introduction:** In a previous [3D programming article](#), we introduced some basic 3D programming concepts. This document describes the further 3D object operation, including loading 3D content from M3G files and playing animations. For more on the basic concepts in mobile 3D programming or to brush up on some basic terminology, please refer to the Technical Article titled [Mobile 3D Graphics Programming](#).
- **Loading 3D contents from M3G files:** introduces how to load 3D content from an M3G file.
- **Playing Animations:** describes how to play pre-defined animations in M3G files.

### Definitions, Abbreviations, Acronyms

M3G	Mobile 3D Graphics
GIF	Graphics Interchange Format
URI	Uniform Resource Identifier
WYSWYG	What You See is What You Get



## Loading 3D content from an M3G file

Loading ready-made pieces of 3D content from an M3G file is generally the most convenient way for an application to create and populate a 3D scene. The M3G file can be created easily by some graphic design tools such as 3DMax, so as a developer, you only need to focus on how to use those 3D objects. Specific instruction on generating an M3G file is beyond the scope of this article, so please refer to your specific graphics design tool's manual or the MOTODEV Technical Article [Developing 3D Applications for Mobile Devices](#), for further information on this subject. This article assumes you may be working with a person who is a graphics designer who is providing the M3G file.

### Loader class

The Loader class can read out and deserialize any 3D objects derived from the Object3D class in an M3G file. The Loader class cannot be instantiated, and its only members are the two static load methods as shown below.

```
public static Object3D[] load(java.lang.String name)
                                throws java.io.IOException

public static Object3D[] load(byte[] data,
                                int offset)
                                throws java.io.IOException
```

The parameter *name* is the name of the resource to be loaded, and is usually a M3G file URI. The load method returns all 3D objects in an M3G file as an Object3D array. Then you may operate on each 3D object individually or display the whole graph scene.

### Loading an M3G file

Now, we will load the skaterboy.m3g file and display it via a canvas object. This .m3g (skaterboy.m3g) file can be found in the [Sun WTK](#).

```
class MyCanvas extends Canvas {
    private World myWorld = null;

    public MyCanvas() {
        try {
            //Load M3Gfile
            Object3D[] roots = Loader.load("/skaterboy.m3g");

            //Assuming World is the first root node
            myWorld = (World)roots[0];

            //render canvas
            repaint();
        }
    }
}
```



```
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    ...
}
```

Here, we assume the World object is the first node of the Object3D array. If not, please consult your graphics designer.

Another way to locate World object is to use the User ID. You can also get the World object's User ID value from your graphics designer.

```
// The USER_ID_WORLD constant value should be the same with
// World's User ID value in M3G file
if (roots[0].getUserID() == USER_ID_WORLD) {
    myWorld = (World) roots[0];
} else {
    ...
}
```

## Display World Scene

The typical way to display an entire scene, is shown in the code segment below. The skating boy is illustrated in Figure 1.

```
protected void paint(Graphics g) {
    //Draw 3D Scene
    Graphics3D myGraphics3D = Graphics3D.getInstance();
    try {
        myGraphics3D.bindTarget(g);
        myGraphics3D.render(myWorld);
    } finally {
        myGraphics3D.releaseTarget();
    }
}
```



Figure 1: Loading the M3G File



## Playing an Animation

For M3G files created via graphic design software applications animations can be created to be played in the MIDlet application. This method gives developers a flexible and powerful way to develop 3D MIDlet applications.

### The *animate()* function

The **public final int animate(int time)** function is an Object3D's member function which is used to update all animation properties in an Object3D and all Object3Ds that are reachable from this Object3D. The parameter **time** is the world object's time which updates the animation. The `animate()` function returns the number of time units until this method needs to be called again for this or any reachable Object3D. Typically, the MIDlet application would call this method once per frame as shown in the code below. Please also be aware that the time in each frame may be different. The next animation frame time should be calculated according to the return value of the `animate()` function.

```
int vol = myWorld.animate((int)(System.currentTimeMillis()-
    startTime));
```

If this function is continuously called, the animation will be played as expected. This can be implemented via a Timer and a TimerTask. The class `RefreshTask` is derived from the `TimerTask` and it is used to invoke the `paint()` method upon each timer event. In the `paint()` method, the `animate()` function will be called to play the animation and next timer event is also scheduled.

```
/**
 * TimerTask
 */
class RefreshTask extends TimerTask {
    public void run() {
        //invoke paint() method of Canvas
        repaint();
    }
}
```

### Updating an Animation

Now, modify the above `MyCanvas` class to add a Timer and TimerTask, then add a variable to record the animation start time.

```
class MyCanvas extends Canvas {
    private World myWorld = null;

    //Keep the start time of animation
```





```
long startTime = 0;

Timer timer = new Timer();
RefreshTask myRefreshTask = null;
public MyCanvas() {
    try {
        // Load M3G file.
        Object3D[] roots = Loader.load("/skaterboy.m3g");
        // Assuming World is the first root node
        myWorld = (World) roots[0];

        //The animation start time
        startTime = System.currentTimeMillis();
        repaint();
    } catch (Exception e) {
        e.printStackTrace();
    }
    ...
}
```

In the `paint()` method, a new timer task is created and the timer is calculated by the return value of the `animate()` function. As in the above code, in the timer task, the `paint()` method will be invoked in such a manner as to simulate a continuous loop.

```
protected void paint(Graphics g) {
    if (g == null) {
        return;
    }

    // delete current timer task
    if(myRefreshTask != null)
    {
        myRefreshTask.cancel();
        myRefreshTask = null;
    }

    // play animation
```



```
int vol = myWorld.animate(
    (int)(System.currentTimeMillis()-
        startTime));

// render scene
Graphics3D myGraphics3D = Graphics3D.getInstance();
try {
    myGraphics3D.bindTarget(g);
    myGraphics3D.render(myWorld);
} finally {
    myGraphics3D.releaseTarget();
}

//create a new timer task
myRefreshTask = new RefreshTask();

// calculate next animatiton time
if(vol < 1)
{
    vol = 1;    //if vol is too less, then use 1
}

//if vol is too big, then use 1 second.
if(vol == 0x7fffffff)
{
    timer.schedule(myRefreshTask, 1000);
} else {
    timer.schedule(myRefreshTask, vol);
}
}
```

The animation is shown in Figure 2.



Figure 2: Playing the animation

## Conclusion

To play an animation pre-defined in an M3G file, developers should work closely with a 3D graphics designer. The graphics designer should create the key frame and animation track according to application requirements using a WYSWYG 3D designing tool. This approach will leave the designer free to focus on the 3D content, and on the other side, leave the programmer free to focus on how to control the 3D object and play the animation.

## References

- 1 JSR 184, Mobile 3D Graphics, <http://jcp.org/en/jsr/detail?id=184>
- 2 Mobile 3D Graphics Programming, [http://developer.motorola.com/docstools/technicalarticles/Developing\\_3D\\_Apps.pdf/](http://developer.motorola.com/docstools/technicalarticles/Developing_3D_Apps.pdf/)
- 3 Developing 3D Applications for Mobile Devices, [http://developer.motorola.com/docstools/developerguides/Developing\\_3D\\_Applications.pdf/](http://developer.motorola.com/docstools/developerguides/Developing_3D_Applications.pdf/)