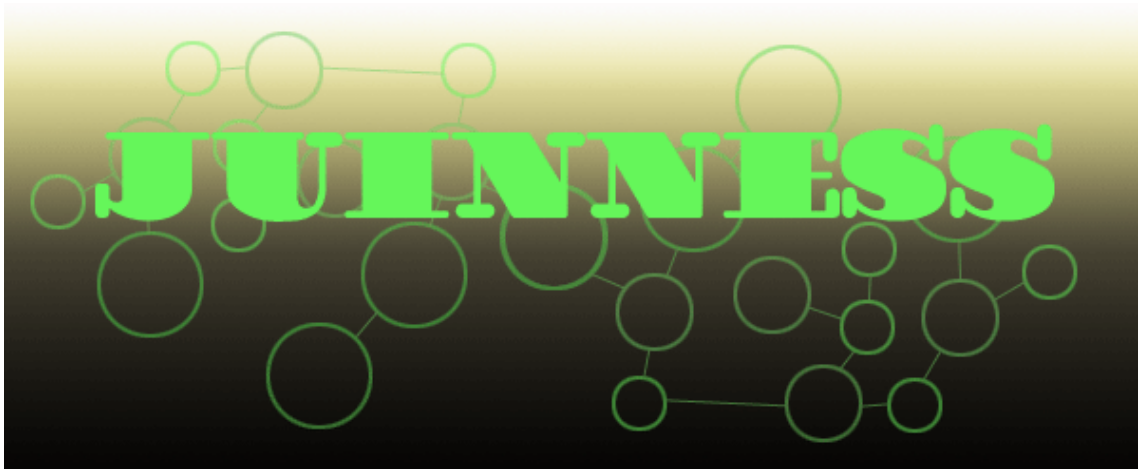
HELSINKI UNIVERSITY OF TECHNOLOGY 12.1.2005
Telecommunications Software and Multimedia Laboratory
T-111.500 Multimedia Seminar
Programming Exercise Documentation
Fall term 2004: juinness@maijavilhelmiina.net



Maija Savolainen
Markus Ylikerälä

JUINNESS

TABLE OF CONTENTS

1 INTRODUCTION

JSR-184, that is, the Java Mobile 3D Graphics (M3G) API [1], is an optional packet for Java 2 Micro Edition (J2ME). M3G is not the only Java based 3D technology but there is also Java 3D (J3D) JSR912 [2]. Although these are differences between J3D and M3G technologies, there also are similarities. Thus, the M3G can be seen as a limited but also as a more compact version of the J3D. The J3D API and the M3G API provide only very basic ways of creating 3D objects hat is very time consuming and error prone. In reality complex objects are modeled with some modeling software and exported as a file according some model format. The content of the file can further be loaded into the scene graph, which is a data structure that has the form of a directed acyclic graph (DAG). Thus, the author of the virtual world needs not to be a programmer but rather an artist.

A method to get Wavefront models into M3G is described in [3] but the method bypasses utilization of the M3G file format and instead code is written directly into the mobile application. There has claimed to be also another project for developing the M3G converter [4] but the status of this project at this moment is unknown. Although M3G can be considered as a relative new technology, we see the poor support for model loading into M3G application as a flaw.

File converters are nothing new but many of them exports objects and textures only [5]. To enable richer 3D environments to be created [6] describes an application that produces X3D [7] from Unreal-based editors. There are commercial plug-ins, such as the Swerve Studio [8] that can be installed to 3D modeling software like Maya [9] and 3D Studio Max [10]. These convert 3D graphics into M3G format, which allows 3D scene graphs to be described and loaded into the M3G application such as the midlet. The 3D Studio Max, which is popular 3D modeling software, supports in its latest version (7.0) at this moment also direct conversion into M3G files. Still no low cost or free open source or shareware products have yet been released. For this reason there is a need for the M3G file conversion. Thus, our focus is on the Juinness file converter, a novel solution that we have developed to handle the conversion between arbitrary 3D formats and the M3G file format.

2 DEVELOPMENT OF JUINNESS

There are several 3D file formats available that describe scene graphs, which can consist of one or several nodes, such as geometry. Some of the 3D file formats are open and the other are vendor specific. Although J3D does not provide a J3D file format, it provides an interface for loading arbitrary 3D files. Whereas it's possible to load common 3D files into the J3D application, it is not possible to load these files directly into the M3G application because the only 3D file format that M3G supports for loading scene graphs is the M3G file format.

Thus we develop the Juinness file converter, a novel solution that handles the conversion of arbitrary scene graphs loadable with J3D into the M3G file format. Thus, the M3G file can further be loaded into some M3G applications such as the MIDlet. The

architecture for the Juinness is shown in Fig. 2.1. The process for the M3G conversion, with the Juinness, consists of the following parts:

- Offering an arbitrary 3D model by getting an existing model or creating a new one
- Phases of the Juinness
- Loading the M3G file into a M3G application

Especially, the phases of the Juinness consist of the following parts:

- Loading the model with a J3D loader
- Constructing a J3D scene graph
- Translating the J3D scene graph into a M3G scene graph according the mapping between the J3D and M3G APIs
- Writing the M3G scene graph according the M3G file format specification

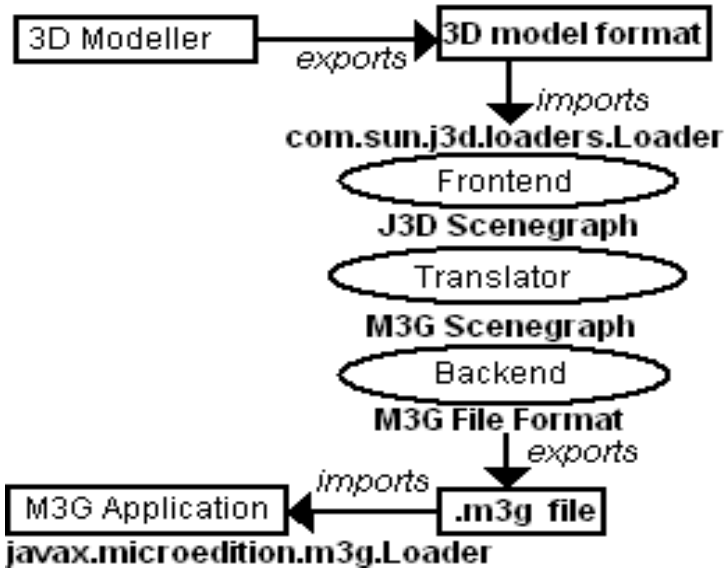


Figure 2.1: Juinness Architecture

A more detailed but implementation specific architecture of the current Juinness is shown in Figure 2.2. In the Figure the data structures are in boldface and the components that do the conversion between the data structures are in circles.

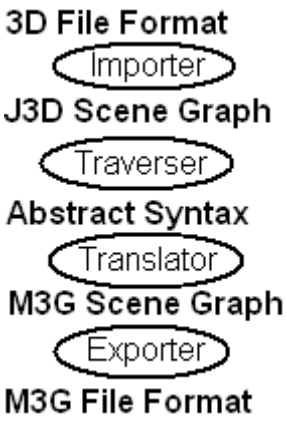


Figure 2.2: Juinness Architecture

2.1 Importer

The Juinness can load arbitrary model formats. This is based on the fact that there are several model loaders available for J3D [11]. In addition, J3D provides a common representation for different model formats with the J3D scene graph. Thus, we utilize the J3D scene graph as a common interface. Thus, the Importer imports, that is, loads, the 3D file with the J3D Loader. The result of this operation is a J3D SceneGraphObject, which can further be inserted into a more content richer scene graph that have also been loaded from a file, such as X3D, or created programmatically.

2.2 Traverser

Despite that it is called a graph, as for J3D scene graph, the scene graph is actually a tree structure [2] so tree traversal algorithm can be used. For the conversion to be possible, a mapping between the J3D API and M3G API is needed. A conceptual mapping is shown in Table. 2.1 The term N/A stands for Not Available and indicates that J3D/J2SE does not contain such a type. In addition, the term N/S=Not Specified stands for Not Specified and indicates that we do not specify the mapping until further notice.

Table 2.1: M3G ObjectType Values

M3G	**J3D/J2SE**
Header Object	N/A
AnimationController	N/S
AnimationTrack	N/S
Appearance	Appearance
Background	Background
Camera	ViewingPlatform/TransformGroup
Composoting Mode	N/S
Fog	Fog
PolygonMode	N/S
Group	BranchGroup, TransformGroup, N/S

Image2D	BufferedImage
TriangleStripArray	Geometry
Light	Light
Material	Material
Mesh	Shape3D
MorphingMesh	N/S
SkinnedMesh	N/S
Texture2D	Texture2D
Sprite	N/S
KeyframeSequence	N/S
VertexArray	Geometry
VertexBuffer	Geometry
World	root BranchGroup
External Reference	N/A

According to [1] the M3G scene graph must be serialized in such an order that referenced objects are serialized before the object that reference to them. This is called leaf-first order and also reference based order. The suitable tree traversal algorithms for a M3G scene graph are reversed level order or post order. In the former all of the nodes at level N+1 are processed before any node at level N. In the latter the parent node is processed before its children.

Thus one of the tree traversal algorithms can also be utilized for the traversal of the J3D scene graph. Especially, if the exporting were done without constructing an internal data structure before exporting, the objects could be serialized in the right order. But, the M3G file format defines the size of a vertex consisting of two bytes for each coordinate in contrary to J3D that uses four bytes. Thus, a uniform scaling, pseudo code shown in Figure 2.3, is needed for all of the meshes before serialization or data is lost with rounding errors and overflows. For this reason, it does not matter in which tree traversal algorithm is used because internal data structure is constructed from the whole J3D scene graph before serializing it. The internal data structure is Abstract Syntax. Thus, the Traverser implements the mapping between the J3D API and Abstract Syntax.

Although, it is still possible to do the conversion without internal data structure, the content of the vertices should be written twice. The first time before scaling when the tree is traversed and the second time after the parameters for the uniform scaling are resolved. Because using the secondary memory consisting of the file operations is much slower compared to primary memory the Juinness does not do this. But, if the primary memory is not enough to hold the internal data structures, some of the internal data structures could explicitly actually be written into secondary memory. But, we assume that it is enough to implicitly let the operating systems (OS) take care of the memory management.

```
factor = abs(maxCoordinateValueOfAnyJ3Dvertex);
scale = 32767 / factor; //unsigned two bytes:-32768...+32767
for(eachJ3DVertex){
  M3GVertex[i] = (short)(J3DVertex[j]*scale);
}

VertexArray positions.set(0, numVertices, M3GVertex);
```

```
VertexBuffer vertexBuf.setPositions(positions, scale,
bias);
```

Figure 2.3: Pseudo Code for the Uniform Scaling

2.3 Abstract Syntax

To make a clear interface between the J3D API and the M3G API, we introduce a layer between them so that the conversion is eased. Thus, the Translator does not need to know anything about the J3D scene graph and the scaling can be made in a clean way. The initial purpose of the Juinness is to produce M3G file format. But, the Abstract Syntax enables also other formats than M3G to be exported if there is a mapping between these APIs because the Abstract Syntax is ultimately a wrapper around the J3D API.

2.4 Translator

The Translator translates the Abstract Syntax into serializable M3G. Thus, the Translator implements the mapping between the Abstract Syntax and M3G API and serializes the objects in a reference based order.

Instead of defining own class hierarchy, we utilize the M3G objects because they already contains the attributes that is written into M3G file and methods for the attributes. Some M3G classes contain methods just to set but not to get data. Because Exporter ought to know the data of the M3G object, we extend the serializable M3G classes into sub M3G classes so that get methods can be implemented for the classes that do not define them. Although some of the M3G classes define both the set and get methods, we extend also them. The reason for this is to force implementing of the Sub interface that enables getting of the object type for the purpose of the Exporter.

2.5 Exporter

The structure of the Exporter is implemented according the M3G file format specification [1]. The Exporter exports the M3G scene graph according the M3G file format, for example into file system, where any M3G application can load it with M3G Loader. We utilize a pool of wrappers to handle the mapping between the M3G API and M3G file format, thus, the serialization of each M3G object, thus, continuous memory allocation from the heap is avoided. For each M3G object there is a wrapper object that retrieves the data from the M3G object and converts it into byte stream. Also the memory for each section is allocated at a time because each object type knows how much space it needs so that continuous copying of data is avoided.

3 SUPPORTED FUNCTIONALITY

Our interest was to enable loading of models into a M3G with the resources we had during the course to have a proof of concept. We were able to demonstrate the Juinness with a real mobile device. The functionality that was implemented for the development purposes during the course consists of the following.

- Support for loading vrml file format, functionality depends of the content of the file
- Support for J3D IndexedTriangleArray geometry, that is, vertices
- Support for normals, colors
- Support for writing almost all of the M3G serializable objects into a M3G file, e.g. animation is not supported yet
- Support for reading all of the M3G serializable objects from a M3G file
- Support for image handling that can be used e.g. for the background

3.1 Postdemo Development

There is some functionality that has been implemented after the demo given on the presentation day. This work can be considered as an extra work because we already have the proof of concept. Thus the Juinness does not currently support everything so some of this functionality is not yet completed but is left as the future work.

The postdemo functionality consists of support for ase file format, J3D TriangleArray geometry and textures. Because the original ase loader downloaded from [11] did not implement the J3D Loader interface we had to modify the ase loader. In addition, the ase loader scales the models in different ways, thus, some of the models are scaled too much or too little. Framework for textures is also developed. Although textures can now be loaded for some models, it still needs further work to be fully capable. An odd thing was to notice that M3G object Texture2D throws the following error shown in the box. Thus, we were not able to extend the Texture2D for the sub object but we needed to extend it directly from the base class of the Texture2D.

```
Exception in thread "main" java.lang.UnsatisfiedLinkError: log2
        at com.nokia.phone.ri.m3g.Math.log2(Native Method)
        at javax.microedition.m3g.Texture2D.<init>(Unknown Source)
        at juinness.m3g.SubTexture2D.<init>(SubTexture2D.java:)
```

4 USING JUINNESS

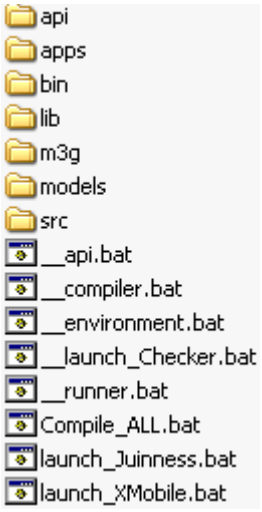


Figure 4.1: Juinness Files

The Juinness has been developed in Windows environment, because the J2ME Wireless Toolkit (WTK) runs on it. You will need to have the J2SE 5.0 installed so that it can be found from the PATH environment variable there is very little J2SE 5.0 specific code so it is easy to use also J2SE1.4 SDK. You will also need to create a WTK project if you want to test the exported m3g file with the emulator. The MIDlet Class Name that we have used is Demoni. In the following chapters we will tell you how to use the development environment and of what it consists of.

4.1 Folders

- api: The API-documentation of the Juinness is generated here
- apps: The Midlet that uses the generated M3G-file. Copy the java-files of this folder to the src-folder of the WTK-project that you have created.
- bin: source code of the Juinness is compiled here
- lib: Contains files for the J3D and M3G so that Juinness can be compiled and run.
- m3g: Default folder for the generated M3G-files
- models: Here is the wrl (vrml) model used to convert. Also the background image used is here.
- src: Contains the source code of the Juinness.

4.2 Bat-files

- __api.bat: Creates the API Documentation of the software to the api-folder

- __compiler.bat: There should be no need to modify this file
- __environment.bat: Adjust here the environment suitable for you. Modify the bolded lines suitable for your environment. Look at the file in the Appendix 1.
- __launch_Checker.bat: Shows the content of the M3G-file
- __runner.bat: There should be no need to modify this file
- Compile_ALL.bat: Compiles all source files in the src-folder and moves the class files to the bin-folder.
- launch_Juinness.bat: Launches the Juinness application
- launch_XMobile.bat: Shows the loaded model in Java 3D

To get started you need probably first do some minor changes to the **_environment.bat**. Then you need to run the **Compile_ALL.bat** and then launch the application with the **launc_Juinness.bat**.

4.3 Notice

You need also download J3D, M3G and J3D loader specific jar, zip and dll files and set them into lib directory although it is enough that they are found during the compilation and running of the Juinness.

The required files are specified with J3D, MOBILE and LOADER variables in **_environment.bat**

FUTURE WORK

There have been attempts to create a converter similar to the Juinness. Since our converter still needs some development to function correctly with different 3D-models, we are going to publish it on the SourceForge-website. We hope that someone can develop it further and utilize the converter in his/her work.

We are also going to publish a Juinness website, where the interested folk can get familiar with it. The website will at least be found in the following address: juinness.maijavilhelmiina.net

5 CONCLUSIONS

We have developed the Juinness file converter, a novel solution that handles the conversion of arbitrary models and scene graphs loadable with J3D into the M3G file format. The Juinness enables richer 3D environments to be created. Despite the fact that the Juinness is able to convert several nodes of the scene graph, such as geometry, it still lacks functionality that is left as the future work for the open source community.

6 REFERENCES

[1] JSR-184, Mobile 3D Graphics API for J2ME
http://jcp.org/en/jsr/detail?id=184, referenced 2004-10-21

[2] JSR-912, Java 3D API 1.3
http://jcp.org/en/jsr/detail?id=912, referenced 2004-11-03

[3] Java Graphics and gaming / Loading OBJ models into M3G
http://fivedots.coe.psu.ac.th/ ad/jg/objm3g/index.html, referenced 2004-11-03

[4] Miklabs
http://www.miklabs.com/, referenced 2004-11-03

[5] Polytrans 3D model converter, OkinoComputer Graphics
http://www.okino.com/conv/conv.htm, referenced 2004-11-03

[6] David Arendash, The Unreal Editor as a Web 3D Authoring Environment,
Proceeding of the eighth international conference on 3DWeb technology, ACM2003

[7] X3D
http://www.web3d.org/, referenced 2004-11-03

[8] Swerve
http://www.superscape.com/products/swerve_studio/ 2004-11-03

[9] The Maya Family
http://www.alias.com/eng/products-services/maya/index.shtml, referenced 2004-11-03

[10] 3D Studio Max
http://www4.discreet.com/3dsmax/, referenced 2004-11-03

[11] File Loader Archives, http://www.j3d.org/utilities/loaders.html

7 APPENDIX 1: __ENVIRONMENT.BAT

```
@echo off
REM ************************
REM lines beginning with REM are comments
REM @echo off disables echoing
REM @echo on enables echoing
REM SET defines a variable
REM ************************

REM ************************
REM note that these are relative not absolute paths because
REM your paths probably differs from mine
REM ************************
SET BASE=..
SET LIB=%BASE%\lib
SET SRC=.\src
SET TARGET=%BASE%\bin
SET BIN=.\bin
SET API_PATH=.\api

SET PACKAGE_JUINNESS=.\juinness\*.java
SET PACKAGE_ABSYN=.\juinness\absyn\*.java
SET PACKAGE_M3G=.\juinness\m3g\*.java
SET PACKAGE_UTIL=.\juinness\util\*.java
SET PACKAGE_FAKE=.\javax\microedition\m3g\FakeTexture2D.java
SET PACKAGE_ASE=.\ta\aseloader\*.java

SET SOURCES=%PACKAGE_FAKE% %PACKAGE_JUINNESS% %PACKAGE_ABSYN%
%PACKAGE_M3G% %PACKAGE_UTIL%
REM %PACKAGE_ASE%

REM ************************
REM jm3d stuff
REM ************************
SET MOBILE=%LIB%\classes.zip

REM ************************
REM j3d stuff
REM ************************
SET
J3D=%LIB%\j3daudio.jar;%LIB%\j3dcore.jar;%LIB%\j3dutils.jar;%LIB%\vecm
ath.jar

//Change someLoader.jar into the actual loader found e.g. from [11]
SET LOADER=%LIB%\someLoader.jar

REM ************************
REM The final 1) compile time and 2) run time CLASSPATH
REM Note that the name CLASSP is used so that we do not mess
REM with the CLASSPATH set with the
ControlPanel/SystemEnvironmentVariables
REM ************************
SET COMP_CLASSP=.;%MOBILE%;%J3D%;%LOADER%
SET RT_CLASSP=.;%MOBILE%;%J3D%;%LOADER%

SET COMPILER=javac

REM ************************
REM Set some other flags to java and compiler(javac/jikes) if needed
REM ************************
SET COMP_FLAGS=-Xlint:deprecation
SET RT_FLAGS=

REM ************************
```

```
REM Should be enough just to modify the following parameters
REM ************************
//Write here the path where your WTK is installed
SET WTK=C:\WTK22
SET PATH=%PATH%;..\lib;%WTK%\bin
SET WTK_APPS=%WTK%\apps
SET WTK_LIB=%WTK%\lib
SET RT_CLASSP=%RT_CLASSP%;%WTK_LIB%/

//If you decide to change the wrl-model write its name here
SET ARG1=../models/coffeepot.wrl

SET ARG2=0
//Create a project called Demoni with the WTK.
SET ARG3=%WTK_APPS%\Demoni\res\test.m3g

//Change here the background
SET ARG4=../models/pyyhe.jpg
```