GL4Java

Implementation Of A Native OpenGL-Interface to Java, X-Window and Windows (95/NT)

Version 2.2.0 Release 0

Sven Goethel

Jausoft - Sven Goethel Software development

29. December 1997 (Diploma Thesis Closing)

7th March 2000 (Last Changes)

May be you want to check the follwing news about GL4Java directly:

- The GL4Java top level homepage at http://www.jausoft.com.
- The JavaDoc package-documentation at packages.html.
- *Download* GL4Java at 2, page 8.
- The *versions* file at 2.1.1, page 8.
- The *changes* file at 7.0.2, page 43.
- The *license* of GL4Java at 7.0.3, page 61.
- The *thanxs* file at 8.1.1, page 62.

Contents

Chapter 1

Introduction

1.1 GL4Java Homepage

Since GL4Java Version 1.0.2 the GL4Java-Homepage has its new top-level homepage.

http://www.jausoft.com contains the top level homepage about GL4Java. Some links to other users are located here.

1.2 About GL4Java

Sorry that this English is not very good ;-) But I guess it's better to write little english for everybody than to write German for only a few. Atilla Kolac and I are working on the implementation of GL4Java[13] for our diploma thesis under the supervision of Prof. Dr. Wolfgang Bunse[12].

Now, i do still maintain GL4Java[13].

The purpose of GL4Java is to use Java as a platform independed programming language for OpenGL applications.

The development of GL4Java may be continued by us or other persons, if there are interesting users.

We would be very lucky, if we get some response, critic, inspirations, bugfixes and participation.

1.3 About this Document

This document was layouted with LaTeX on Linux. We used english as the main language, because this documentation is published on the WWW-Server of the FH-Bielefeld[?]. To convert this LaTeX documentation we used latex2html[14]. The appendix includes the introduction about OpenGL and Java in the german language. We used german for these parts, because there are many documentations in the english language for these purposes [2] [3] [5]. This appendix is not included in the html files.

The paperware we have to produce for the FH-Bielefeld includes a printout of the javadoc-html pages of the base GL4Java classes. We include a link to the javadoc pages in chapter 7.1, page 61.

1.4 Who wrote which part?

The diploma thesis is splitten into two diploma theses:

- Sven Goethel: Implementation Of A Native OpenGL-Interface to Java and X-Window
- Atilla Kolac: Implementation Of A Native OpenGL-Interface to Java and Windows-NT

Because we both worked together in design, implementation and documentation, we publish one documentation and one package.

To help others to distinguish who of us is responsible for which part, we marked Sven Goethel's tasks with a †, we marked Atilla Kolac's part with a ‡, and we marked the parts which are written by both of us with nothing at all. If the chapter is marked, the whole chapter was written by this person. If only some sections are marked, only the specific section was written by this person. If nothing is marked both persons wrote those chapter.

Since Version 1.1.0 Sven Goethel maintains this package.

1.5 The Procedure†

1.5.1 History

The OpenGL interface for Java work was started by Leo Chan[6]. He implements the library with Java 1.0.2 native calls and with an extra window for OpenGL-Rendering.

Leo Chan's works was continued by Adam King[7]. His OpenGL4Java is able to be compiled with Java 1.1 and the OpenGL-Rendering is done in the calling Java-Frame. He still uses the Java 1.0.2 native calls.

Tommy Reilly[8] participated to Adam King's work and the project's title changed to Jogl. Jogl's big points lies in it's powerfull autoconfig and in it's improved X-Window System functions - so mostly all Unices are supported. Another point is the Win32 support. Sources and a precompiled dll are distributed. Jogl still differs from the true OpenGL naming convention.

Because of newly communication results, we are thinking about joining the Jogl project. This does not means, that there will be no more GL4Java ! We still support the GL4Java, with the OpenGL like API (naming conventions) !

1.5.2 Our Work

We started developing GL4Java with Adam King's[7] OpenGL4Java. As a matter of asynchronise development, the changes from Java 1.0.2 to JNI 1.1 native calls were made within the Jogl and GL4Java project parallel - because we did not know of each others development.

Actual results and later changes in Jogl are ported and will be ported to GL4Java. Like Jogl, GL4Java uses the Java 1.1 Java-Native-Interface (JNI). Many incompatible changes to OpenGL4Java/Jogl were made, see the chapter 7.0.2 at page 43.

One big point of GL4Java is the used OpenGL naming convention, support for glu* and glut* functions (glut support in the near future).

GL4Java extends OpenGL's API with an own naming convention. Specialised known windowing functions, like glXSwapBuffers, have the prefix glj, like gljSwap.

Also the GLFrame class adds itself as a ComponentListener, so we have a event-handler in java, like reshape for glut !

If we uses a own created Color-Window as the GL-Window (not usefull for AIX, LINUX AND SOLARIS YET), gljResize will be called if the componentResized is called (ComponentListener) to resize the own created Window !

1.5.3 Status

We can announce the following stats about GL4Java Version 2.1.0.0

- Stable versions with precompiled librarys for Linux, SunOS (Solaris), Windows (95/NT) and Macintosh.
- Runs on Java 1.1.5 and higher, tested on:
  - JDK 1.1.7 (win32 (plus jit), linux (plus jit (tya v3.0)))
  - JDK 1.2 (Win32, Linux (Pre-V1,native,no-jit), Macintosh, ...)
  - Java2 (Java1.2) Plug-In on Netscape 4.5 (Win32)
  - Netscape 4.5 (Win32/Unix)
  - MS-JVM (build 3186) Win32
  - InternetExplorer4.0 with MS-JVM (build 3186) Win32
- AIX stable version with some multithreading difficulties
- Many demonstrations of GL4Java, which also uses many Java GUI features.
- A JavaDoc HTML-Documentation

1.5.4 Targets

If you use Mesa as your OpenGL driver, you should use Mesa Version 3.0 or higher !

The actual implementation of GL4Java supports the following UNICES

- AIX (RS/6000) 4.2 with $\geq$ JDK 1.1.5 (without JIT_COMPILER option)
- Linux (x86) 2.X with Mesa-3.0 and Metro-Link's GL and Xi's Accelerated OpenGL + $\geq$ JDK 1.1.7
- SunOS (Sparc) 5.X with Mesa-2.4 GL with $\geq$ JDK 1.1.5 (green threads)
- SGI-Irix
- Macintosh

but if you know about programming, i guess it only makes little afford to support your unix.

And the following Windows

- Windows NT 4.0 & 95/98 (x86) with MS OpenGL-Library + $\geq$ JDK 1.1.7
- Windows NT 4.0 & 95/98 (x86) with MS OpenGL-Library + $\geq$ MS-JVM (build 3186)

Chapter 2

Installation†

2.1 Obtain GL4Java

You can download GL4Java at http://www.jausoft.com/Files/Java/1.1.X/GL4Java, where you may can find older versions and new demos either.

Then go to this chapter and click on the following list, or just download the appropiate file.

Please use some *Save Link to ...* function of your browser !

It could happen that your browser unzip the package automatically, then 'gunzip' or 'tar xzf' does not work of course - just use 'tar xf' then !

Windows users can use the WinZip utility to extract the tar-gzipped files, but be sure to NOT convert LF/CR characters, because these files are binary !

- The precompiled binaries for all the machines !
- The demos: GL4Java2.2.0.0-demos-v1.zip The demos (sources and classes) for you to enjoy !
- The source: GL4Java2.2.0.0-src.tar.gz All sources no precompiled binaries no demos !
- The docs: GL4Java2.2.0.0-doc.zip This Documentation (html+postscript) plus javadoc-package-documentation !

2.1.1 GL4Java's Version Numbers

Here is a little descriptions what the versions mean.

```
binpkg/gl4javaX.Y.Z.R.zip                (the JAR file)
binpkg/libGL4JavaX.Y.Z.R-<UNIX-TYPE>.tar.gz    (the native libs - unix)
binpkg/libGL4JavaX.Y.Z.R-<WIN32-TYPE>.zip      (the native libs - win32)

        archive/GL4JavaX.Y.Z.R-demos.zip  (demos      - for users :-)
        archive/GL4JavaX.Y.Z.R-src.tgz    (everything - for developers :-)
        archive/GL4JavaX.Y.Z.R-doc.tgz    (everything - for readers :-)

        (where X is the major, Y the minor, Z the bugfix and R the release
 library version number !
 Take a look at VERSIONS.txt for a closer description !
)
```

Version Compatibility

```
All about the GL4Java Version Numbers
=====================================

The GL4Java version-string is X.Y.Z.R:

       X - The major version number,
           this only changes if the architecture has changed significant AND
           some or complete incompatibilities in the Java-API are drawn.
   Synchronisation to other ports needed.

       Y - The minor version number,
           this may change if some changes or add-ons are made
   without incompatibility in the Java-API/ Java-Files.
   Synchronisation to other ports is recommended.

       Z - The bugfix number,
           this may change for bugfix purposes in the
           Java- or Native-Files !
   Synchronisation to other ports is not needed.

       R - The release number,
           for changes in native-libs, packaging, etc.
   Synchronisation to other ports is not needed.

 The reason for synchronisation within X and Y is because
 the GL4Java compatibility !

 A GL4Java X.Y.x.x application with an defined X and Y should run on all
 implementations !

 Another goal is that the gl4java.jar file of version X.Y.xx.xx can be used
 for any machine !

 My proposal is (from now on 2.0.0.1) to being synchronized for X and Y !
 Everybody has to make an GL4Java-RFC to all developers on GL4Java
 (actually three interests :-).

 Please add yourself to the GL4Java mailinglist !
 Look at http://www.jausoft.com/gl4java !
```

2.2 GL4Java's Directory structure

`./`	this is the directory, where you extract this package
`./GL4Java`	GL4Java root directory
`./GL4Java/gl4java`	java-package[a]
`./GL4Java/gl4java/awt`	java-package[a]
`./GL4Java/gl4java/jau/awt`	java-package[a]
`./GL4Java/gl4java/system`	java-package[a]
`./GL4Java/sun/awt/macintosh`	additional java-package[a]
`./GL4Java/sun/awt/motif`	additional java-package[a]
`./GL4Java/sun/awt/windows`	additional java-package[a]
`./GL4Java/CNativeCode`	native code[a]
`./GL4Java/demos`	GL4Java demos[b]
`./GL4Java/demos/natives`	same demos as native implementation[b]
`./GL4Java/mklibs`	shell-scripts to do dynamic libs for different plattforms[a]
`./GL4Java/binpkg`	the precompiled binaries[c]
`./GL4Java/binpkg/gl4javaX.Y.Z.R.zip`	GL4Java Java-Archive[c]
`./GL4Java/binpkg/libGL4JavaX.Y.Z.R-<WIN32-TYPE>.zip`	... for Windows 32[c]
`./GL4Java/binpkg/libGL4JavaX.Y.Z.R-<UNIX-TYPE>.tar.gz`	... for Unices[c]
`./GL4Java/CClassHeaders`	temporaery dir for javah[a]
`./GL4Java/C2J`	the C2J cross-compiler[a] used to create OpenGL and GLU wrappe
`./GL4Java/C2J/manual`	manual wrappers (see above)[a]
`./GL4Java/docs`	documentation target dir[d]
`./GL4Java/docs/html`	the html-documentation[d]
`./GL4Java/docs/html/icons`	icons for latex2html[d]
`./GL4Java/docs/html/images`	icons for javadoc[d]
`./GL4Java/docs-src`	doc source for tex-doc[a]
`./GL4Java/archive`	all archives (*.tar.gz) resides here[e] !
`./GL4Java/Win32BC5`	the target dir for Win32 development (Borland C Version 5.0)[a]
`./GL4Java/Win32VC5`	the target dir for Win32 development (MS Visual C Version 5.0)[a]

[a]within the src-package
[b]within the demo-package
[c]for the binary installation
[d]within the doc-package
[e]not within any package

2.3 Step by Step

Because we do not know much about autoconfiguartion tools, we only can support a *step by step* installation procedure.

2.3.1 Binary Installation

```
WHERE DO YOU GET the actual version of GL4Java ?
================================================

  Unix, Windows, ...
  ===================
        http://www.jausoft.com/gl4java.html

        Go then to GL4Java !

binpkg/png-1.0a-jar.zip                      (the PNG-JAR file)
binpkg/gl4javaX.Y.Z.R-jar.zip               (the JAR file)
binpkg/gl4javaX.Y.Z.R-glutfonts-jar.zip       (the JAR file)
binpkg/gl4javaX.Y.Z.R-classes.zip       (the classes in a ZIP)
binpkg/gl4javaX.Y.Z.R-glutfonts-classes.zip    (the classes in a ZIP)
binpkg/libGL4JavaX.Y.Z.R-<UNIX-TYPE>.tar.gz    (the native libs - unix)
binpkg/libGL4JavaX.Y.Z.R-<WIN32-TYPE>.zip      (the native libs - win32)

        archive/GL4JavaX.Y.Z.R-demos.zip  (demos           - for users :-)
        archive/GL4JavaX.Y.Z.R-src.tgz    (sources no demos - for developers :-)
        archive/GL4JavaX.Y.Z.R-doc.tgz    (everything      - for the patiente :-)

        (where X is the major, Y the minor, Z the bugfix and R the release
 library version number !
 Take a look at VERSIONS.txt for a closer description !
)


   Macintosh
   ===========
    http://members.home.net/gziemski/pages/projects/GL4Java.html



The Documentation
=================

It is recommended to check the documentation (so much work on it ;-).
Just download the file 'GL4JavaX.Y.Z-doc.tgz' and un-gzip and un-tar it
(Windows users may use WinZip here :-)

Now you can read the HTML-Documentation with:
<YourHtmlBrowser> GL4Java/docs/html/GL4Java.html

Or you can read the Postscript version with:
<YourPostScriptViewer> GL4Java/docs/GL4Java.ps
```

```
Special Java2 Installation file:
Java2.txt

Special Netscape Installation file:
Netscape.txt

Special MS-JVM Installation file:
MS-JVM.txt


YOU MUST ALLREADY HAVE THE FOLLOWING INSTALLED:
===============================================

        COMMON:
        =======
            o >= jdk 1.1.5 (for using)
            o >= jdk 1.2   (for developing)


              tested:
               jdk1.1.7 (win32 (plus jit), linux (plus jit (tya v3.0)))
jdk1.2   (win32, linux (Pre-V1,native,no-jit))
Java2 (Java1.2) Plug-In on Netscape 4.5 (Win32)
Netscape 4.5 (Win32)

        UNIX / X11 :
        ============
            o ( GL + GLU ) or ( MesaGL + MesaGLU Version 3.0 or higher)
              AND glut (only for c-demos yet)

              We are looking for libGL.so AND libGLU.so in your library PATH,
              so please create a symbolic link from the Mesa libs,
              to the abstract one's !!

            o X11R6 (XFree86 works fine ;-)

            o Unix standard file-utilities (tar, gzip, ...)

        WINDOWS 32 (NT & 95)
        ====================
            o M$ OpenGL and GLU library - !!!! MUST !!!!

             see if you have opengl32.dll AND glu32.dll
         installed in your library path
         (c:/winnt/system32 OR c:/windows/system)

    o For running GL4Java within MS-JVM:
      Be sure you have the new MS-JVM machine (build 3186) installed:

Microsoft (R) VM for Java, 5.0 Release 5.0.0.3186

            o Unix standard file-utilities (tar, gzip, ...)
              OR WinZip (can extract tar-files ;-)
```

```
Installation procedure for UNIX/WINDOWs BINARY DISTRIBUTION:
============================================================

        UNICE and WINDOWs USERS JAVA ARCHIVE
        =====================================


o Choose a version number X.Y.Z.R !
  Be shure that the version numbers X.Y are the same
  for the gl4java.jar file and the native libraries !
  The closest version number match provides the best compatibility !

o download the gl4java.jar files, which is zipped in

binpkg/gl4javaX.Y.Z.R-jar.zip                (the JAR file)
binpkg/gl4javaX.Y.Z.R-glutfonts-jar.zip        (the JAR file)

o unzip the downloaded gl4javaX.Y.Z.R*-jar.zip files,
  this will result a file called gl4java.jar and gl4java-glutfonts.jar !

o download the png.jar file, which is zipped in

binpkg/png-1.0a-jar.zip                       (the PNG-JAR file)

o unzip the downloaded png-1.0a-jar.zip file,
  this will result a file called png.jar !

o Add gl4java.jar AND png.jar to your CLASSPATH (echo \$CLASSPATH)
  if you use Java 1.1.X !

o If you use JAVA2 or JRE - copy gl4java.jar AND png.jar  to :
  ./jre/lib/ext/.

o If you use JAVA2 Plug-In copy gl4java.jar AND png.jar to :
  ./jre/lib/.
  (looks like a bug, because the Java2-PlugIn does not find
   the gl4java.jar file in the ./jre/lib/ext directory )


        UNICE USERS NATIVE LIBRARY
        ============================

o download the native libraries, which is zipped in

binpkg/libGL4JavaX.Y.Z.R-<UNIX-TYPE>.tar.gz    (the native libs - unix)

o change to a directory which is within your LD_LIBRARY_PATH
  environment variable !
  (Look at 'echo $LD_LIBRARY_PATH').

o unpack the downloaded libGL4JavaX.Y.Z.R-<UNIX-TYPE>.tar.gz,
  this will result some files called libGL4Java*.so* !
  If you want to copy the extracted library files,
  be sure to use 'cp -a' to keep the symbolic links alive !

o If you use JAVA2, JAVA2-Plug-In or JRE - copy the libs to :
  ./jre/lib/<machine>/.
e.g. linux:
  ./jre/lib/i386/.

  or

  ./jre/lib/i386/green-threads/.

  Be sure to use 'cp -a' to keep the symbolic links alive !

o If you want to use Netscape 4.5 or above,
  please read Netscape.txt



        WINDOWs USERS NATIVE LIBRARY
        ============================

o download the native libraries, which is zipped in

binpkg/libGL4JavaX.Y.Z.R-<WIN32-TYPE>.zip      (the native libs - win32)

o unzip the downloaded libGL4JavaX.Y.Z.R-<WIN32-TYPE>.zip,
  this will result some files called GL4Java*.dll !

o Then copy the libraries
  to c:/winnt/system32 (WINNT), c:/windows/system (WIN9X)
  or where the other's *.dll files exits !

  You can also copy the files (better) to /java2/jre/bin if using java2 !

o If you use JAVA2, JAVA2-Plug-In or JRE - copy the libs also to :
  ./jre/bin/.

o If you want to use Netscape 4.5 or above,
  please read Netscape.txt

o If you use MS-JVM (MS-InternetExplorer >=4.0)
  please read MS-JVM.txt


        UNICE and WINDOWs USERS
        =======================

o You can check the installation and the used library versions with:
java gl4java.GLContext

o You can now download the demos-archive GL4JavaX.Y.Z.R-demos.zip
  This will create the directory GL4Java/demos !
  Just try the demos while typing:
        cd demos
```

```
          java glDemosCvs

  This will invoke the demo-manager !

o To allow JAVA2's appletviewer using JNI native libs,
  just look at GL4Java/demo/Java2Applet.bat
  (You have to change the gl4java.policy file) !

  You can - of course - use your 'policytool',
  which is shipped with JAVA2 !

  To use the demo Applets for Java2-Plug-In from www.jausoft.com,
  please add the following lines to your java.policy file:

grant codeBase "http://www.jausoft.com/Files/Java/1.1.X/GL4Java/demos/-" {
  permission java.security.AllPermission;
};


o You can check JAVA2-Plug-In with the Java-Applets-Html-File's
demos/glLogoCvsAppletJ2P.html
demos/glOlympicCvsAppletJ2P.html
demos/testTextPPM1J2P.html

o You can check Netscape Win32-JVM with the Java-Applets-Html-File's
demos/glLogoCvsApplet.html
demos/glOlympicCvsApplet.html
demos/testTextPPM1.html (copy swingall.jar of Swing1.1 to
                 ./Netcape/Communicator/Program/Java/classes )

o You can check MS-InternetExplorer >= 4.0 with the Java-Applets-Html-File's
demos/glLogoCvsApplet.html
demos/glOlympicCvsApplet.html
demos/testTextPPM1.html (extract swingall.jar of Swing1.1 to
    C:\WINDOWS\Java\TRUSTLIB - and delete
    C:\WINDOWS\Java\TRUSTLIB\META-INF !)

o you will find further documentations in docs/html/GL4Java.html
  or docs/GL4Java.ps !
  (Download GL4JavaX.Y.Z.R-doc.tgz,
   or check it online - search for it at www.jausoft.com :-)


have a lot of fun, responses and ideas are welcome !

Sven Goethel

6th July 1998
22 April 1999
1st June 1999
2nd September 1999
16th Novemeber 1999
7th March 2000
```

```
---------------------
```

Java2 Installation

```
With GL4Java Version 2.0.0 Release 1,
GL4Java do supports the Java2 plattform !

Please read INSTALL.txt first !

Prerequisites
==============

Be sure to have:

  ./jre/lib/ext/gl4java.jar       : for Java2 or JRE
  ./jre/lib/ext/gl4java-glutfonts.jar : for Java2 or JRE
          ./jre/lib/ext/png.jar          : for Java2 or JRE

  For Windows32: Java2-Plug-In, Java2, JRE

  ./jre/bin/GL4JavaJauGljJNI.dll
  ./jre/bin/GL4JavaJauGLJNI.dll
  ./jre/bin/GL4JavaJauGLUJNI.dll

  For Unix: Java2-Plug-In, Java2, JRE

  ./jre/lib/<machine>/libGL4JavaJauGljJNI.so*
  ./jre/lib/<machine>/libGL4JavaJauGLJNI.so*
  ./jre/lib/<machine>/libGL4JavaJauGLUJNI.so*

  * => all symbolic links and the file itself,
       use "cp -a" to keep the symbolic links alive !

  e.g. linux:

  ./jre/lib/i386/green-threads/.

  Be sure to copy the linked lib-files also !


Java2-Plug-In, Appletviewer, Security:
======================================
  To allow JAVA2's appletviewer and Plug-In using JNI native libs,
  just look at GL4Java/demo/Java2Applet.bat
  (You have to change the GL4Java.policy file) !

  You can - of course - use your 'policytool',
  which is shipped with JAVA2 !

  You can check JAVA2-Plug-In with the Java-Applets-Html-File's
demos/glLogoCvsAppletJ2P.html
```

```
demos/glOlympicCvsAppletJ2P.html
demos/testTextPPM1J2P.html

  To use the demo Applets for Java2-Plug-In from www.jausoft.com,
  please add the following lines to your java.policy file:

grant codeBase "http://www.jausoft.com/Files/Java/1.1.X/GL4Java/demos/-" {
  permission java.security.AllPermission;
};


TODO:
=====

Test other plattforms :-)
```

Netscape Installation

```
With GL4Java Version 2.0.1 Release 2,
GL4Java do supports the Netscape 4.5 Win32-JVM and Unix plattform !

<Version 2.0.0 Release 1 and Version 2.0.1 Release 1
 has a bug within the capabilities call and the JAR package >

Please read INSTALL.txt first !

WINDOW 95/98/NT:
=================

Be sure to have:

  ./Netcape/Communicator/Program/Java/classes/png.jar
  ./Netcape/Communicator/Program/Java/classes/gl4java.jar
  ./Netcape/Communicator/Program/Java/classes/gl4java-glutfonts.jar

  ./Netcape/Communicator/Program/Java/bin/GL4JavaJauGljJNI.dll
  ./Netcape/Communicator/Program/Java/bin/GL4JavaJauGLJNI.dll
  ./Netcape/Communicator/Program/Java/bin/GL4JavaJauGLUJNI.dll


UNIX (Linux/Solaris/...):
=========================

Be sure to have:

  /opt/netscape/java/classes/gl4java-glutfonts.jar
  /opt/netscape/java/classes/png.jar

The nativ libraries must be installed on a directory,
which is in your LD_LIBRARY_PATH environment !!
E.g.:
/usr/lib/libGL4JavaJauGLJNI.so
```

```
/usr/lib/libGL4JavaJauGLJNI.so.2
/usr/lib/libGL4JavaJauGLJNI.so.2.1
/usr/lib/libGL4JavaJauGLJNI.so.2.1.2
/usr/lib/libGL4JavaJauGLJNInf.so
/usr/lib/libGL4JavaJauGLJNInf.so.2
/usr/lib/libGL4JavaJauGLJNInf.so.2.1
/usr/lib/libGL4JavaJauGLJNInf.so.2.1.2
/usr/lib/libGL4JavaJauGLUJNI.so
/usr/lib/libGL4JavaJauGLUJNI.so.2
/usr/lib/libGL4JavaJauGLUJNI.so.2.1
/usr/lib/libGL4JavaJauGLUJNI.so.2.1.2
/usr/lib/libGL4JavaJauGLUJNInf.so
/usr/lib/libGL4JavaJauGLUJNInf.so.2
/usr/lib/libGL4JavaJauGLUJNInf.so.2.1
/usr/lib/libGL4JavaJauGLUJNInf.so.2.1.2
/usr/lib/libGL4JavaJauGljJNI.so
/usr/lib/libGL4JavaJauGljJNI.so.2
/usr/lib/libGL4JavaJauGljJNI.so.2.1
/usr/lib/libGL4JavaJauGljJNI.so.2.1.2

All symbolic links and the file itself,
use "cp -a" to keep the symbolic links alive !

Also you should be sure to have a CLASSPATH for Netscape set
like this:
CLASSPATH=/opt/netscape/java/classes:.
for i in /opt/netscape/java/classes/*.jar ; do
CLASSPATH=$CLASSPATH:$i
done
echo $CLASSPATH
export CLASSPATH

or just make the default whith:

CLASSPATH=
export CLASSPATH

Applets/Security:
=====================
You can check Netscape with the Java-Applets-Html-File's
demos/glLogoCvsApplet.html
demos/glOlympicCvsApplet.html
demos/testTextPPM1.html (copy swingall.jar of Swing1.1 to
                 ./Netcape/Communicator/Program/Java/classes )

You will be asked to grant the asked privilege,
so trust GL4Java and say yes !

TODO:
=====

Test other plattforms :-)
```

Microsoft JVM Installation

```
With GL4Java Version 2.1.0 Release 0,
GL4Java do supports the MS-JVM plattform,
which means GL4Java-Applets runs under the MS InternetExplorer >= 4.0 !

Please read INSTALL.txt first !

Prerequisites
==============

Be sure to have:

The new MS-JVM machine (build 3186) installed:

Microsoft (R) VM for Java, 5.0 Release 5.0.0.3186

(The following version does NOT Work:
Microsoft (R) VM for Java (tm), 4.0 Release 4.79.0.2424)

You can get it from the Microsoft-Webside's:
http://www.microsoft.com/java/
http://www.microsoft.com/java/vm/dl_vm40.htm

Extracted the gl4java.jar, gl4java-glutfonts.jar AND png.jar archive
(with WinZip or the jar command-line-tool)
to: C:\WINDOWS\Java\TRUSTLIB,
so you have the new directories:

  C:\WINDOWS\Java\TRUSTLIB\com\sixlegs
  C:\WINDOWS\Java\TRUSTLIB\gl4java
  C:\WINDOWS\Java\TRUSTLIB\sun
  C:\WINDOWS\Java\TRUSTLIB\META-INF

Delete the new directory C:\WINDOWS\Java\TRUSTLIB\META-INF !

Then copy the native library (DLL) GL4JavaGljMSJDirect.dll
to C:\WINDOWS\SYSTEM[32]

  C:\WINDOWS\SYSTEM32\GL4JavaGljMSJDirect.dll (WINNT)

  or

  C:\WINDOWS\SYSTEM\GL4JavaGljMSJDirect.dll   (WIN9X)


Notes:
======

Be sure to have a MS-JVM installed on your system,
e.g. the one installed within MS-IE 4.0 !

Applets, Security:
```

```
======================================
  Because now you have installed the GL4Java system on your local
  system. All Applet's using GL4Java can run !

  If not - please try to modify your MS-IE security setup !

This Port to the MS-JVM is written by Ron Cemer
and merged into the GL4Java official source tree by Sven Goethel.

TODO:
=====

Support a better window update for the GL4Java OpenGL windows !

Support query Double-Buffer and STEREO, like X11 !

An Easy Installation Tool :-) That's Ron's sweating either.
```

2.3.2 Source Installation

Prerequisites

You must allready have the following installed:

1. Common
   (a) JDK $\geq$ 1.1.7
   (b) JavaCC[9] (chapter 4, page 29)
   (c) bash (chapter 4, page 29)
   (d) GNU make (chapter 4, page 29)
   (e) GNU tools (chapter 4, page 29)
   (f) Netscape Communicator 4.5 (optional)
2. UNIX/X11
   (a) (GL + GLU) or (MesaGL + MesaGLU Version 2.4 or higher) AND glut (only for c-demos yet)
   We are looking for libGL.so AND libGLU.so in your library PATH, so please create a symbolic link from the Mesa libs, to the abstract one's !!
   (b) X11R6 (XFree86 works fine ;-)
   (c) Unix standard file-utilities (tar, gzip, ...)
   (d) LaTeX and LaTeX2Html[14] (to generate the documentation)
3. Windows 32 (NT & 95)
   (a) MS-OpenGL and GLU library installed - **must** ! See if you have opengl32.dll AND glu32.dll installed in your library path (c:/winnt/system32 OR c:/windows/system)
   (b) WinZip (to extract zip and tar-gzip files), be sure to disable the option LF/CR conversion in TAR-Archives ! You can also use the jar command-line tool of the JDK for zip files !
   (c) Unix standard utilities (make, ksh (bash), tar ...) E.g. GNU-Win32 Project from Cygnus[11], make sure to have zip installed also !
   (d) LaTeX and LaTeX2Html[14] (not testet - to generate the documentation)
   (e) For compiling GL4Java for the MS-JVM, be sure you have the new MS-JVM machine (build 3186) installed.

Compilation

UNICEs and WINDOWs users
You CAN (not a must) follow the Installation procedure above, just be sure to copy the libs to your LD_LIBRARY_PATH which should be the path in symbols.mak 'HOME_LIB_DIR' see below (UNICEs) !! Also you should use a symbolic link OR edit CLASSPATH to make 'GL4Java' and 'sun' visible in you CLASSPATH (copy is not recommended - see below !!!) !

You MUST follow the Installation procedure above, if you want to USE GL4Java with JAVA2-JRE, JAVA2-Plug-In, Netscape 4.5 (Win32/Unix) JVM or MS-JVM/InternetExplorer.

Just procede the Installation procedure AFTER the compilation !

Login as root (or any other), obtain the source archive for GL4Java.

copy the .tar.gz file to a directory such as /usr/local and cd to that directory, then unpack the file:

```
cd /usr/local ( or where ever you placed the file )
gunzip GL4JavaX.Y.Z-Rn-src.tar.gz ; tar xvf GL4JavaX.Y.Z-Rn-src.tar
gunzip GL4JavaX.Y.Z-Rn-doc.tar.gz ;
tar xvf GL4JavaX.Y.Z-Rn-doc.tar (optional :-)
```

X is the major, Y the minor and Z the bugfix library version number, and Rn (e.g. R2) is the release number, where no changes in the libs where made !

A new directory will be created called GL4Java.

Go to the directory GL4Java, which is the root-directory of GL4Java - so we stay here for the following procedures !

You will see some 'symbols.mak.<machine>' files. Choose the right 'symbols.mak.<machine>' file, or create one. Some older actual non-maintained files can be found under symbols.mak-old ! Edit your choosen or created 'symbols.mak.<machine>', so it will fit to your OS configuration - all macros ! Copy the file 'symbols.mak.<your-machine>' to 'symbols.mak'. Do not forget to add your 'HOME_LIB_DIR' (set in symbols.mak) to your LD_LIBRARY_PATH (UNICEs). The created lib will be copied to 'HOME_LIB_DIR' by automatic (via tar - symbolic links are still alive ;-) !

Next - you have to be sure, that the root-directory of GL4Java, where the directories *gl4java* and *sun* exists, is in your CLASSPATH. E.g.: Assume you have expanded the package in the directory */usr/local.*

```
For sh/ksh:    CLASSPATH=$CLASSPATH:/usr/local/GL4Java
   or
for csh: set CLASSPATH=(/usr/local/GL4Java $CLASSPATH)
  or
for WINDOW's  check out your GUI's
```

This is evt. obsolete, if you have added the THISDIR path in the CLASSPATH of the symbols.mak's JAVAC definition, look into these files ! You must also use the macro 'DEST_CLASSES_DIR' (symbols.mak), where all generated class-files will be copied into, when created. You can also invoke make classcpy to force copy all class-files to 'DEST_CLASSES_DIR'. Also the gl4java.jar file is generated there !

Because GL4Java supports Netscape's JVM with JNI, we do try to ask for a privilege to run native DLL's. Because this ask - the 'netscape' package is included in the file *capsapi_classes.zip* which is used in the *JAVAC* macro definition in *symbols.mak.*

This is evt. obsolete, if you have entered the file 'capsapi_classes.zip' in the CLASSPATH of the symbols.mak's JAVAC definition, look into these files !

UNICEs users

Next create the library, class-files and all is needed with (for Unix/X11):

```
make x11
```

The Unix makefile-action does the complete creation :-) The default shared-library files for Unice's are :

```
   GLContext: libGL4JavaJauGljJNI.so
GLFuncJauJNI: libGL4JavaJauGLJNI.so
GLUFuncJauJNI: libGL4JavaJauGLUJNI.so

This *.so files stands for the versions files
with all symbolic links, e.g.:

  libGL4JavaJauGljJNI.so     -> libGL4JavaJauGljJNI.so.2.1
  libGL4JavaJauGljJNI.so.2.1 -> libGL4JavaJauGljJNI.so.2.1.2
  libGL4JavaJauGljJNI.so.2.1.2
```

They are moved to the HOME_LIB_DIR (see above) or copied with the "-a" option to keep the symbolic links alive !

WINDOWs users

Next create the class-files and all preparations with (Win32):

```
 make w32
```

The Windows makefile-action only creates the tool (gljni) and all Java classes and C-wrappers. An additional C compiler invocation must be done (see 2.3.3, page 23) to create the default shared-library files:

```
  GLContext     (SUN-JNI): GL4JavaJauGljJNI.dll
GLFuncJauJNI  (SUN-JNI): GL4JavaJauGLJNI.dll
GLUFuncJauJNI (SUN-JNI): GL4JavaJauGLUJNI.dll
  GLContext     (MS-JVM) : GL4JavaGljMSJDirect.dll
```

To run GL4Java within MS-JVM and InternetExplorer, please see 2.3.1, page 19.

UNICEs and WINDOWs users

To run GL4Java within Netscape, please 2.3.1, page 17.
For a complete description of the makefile invokations see 5.1.1, page 30.
Next create the demos with.

```
cd demos
make
```

If all goes well, just type e.g.: 'java glDemosCvs' in this directory, and you should see a demo manager for all the GL4Java demos we have written.

To allow JAVA2's appletviewer or Java2-Plug-In using JNI native libs, just look at GL4Java/demo/Java2Applet.bat (You have to change the gl4java.policy file) !

You can - of course - use your 'policytool', which is shipped with JAVA2 !

To use the demo Applets for Java2-Plug-In from www.jausoft.com, please add the following lines to your java.policy file:

```
grant codeBase "http://www.jausoft.com/Files/Java/1.1.X/GL4Java/demos/-"
{
  permission java.security.AllPermission;
};
```

You can also compare the totally native glut version against the java version. The glut versions do exist in the demos path also (under native).

2.3.3 Windows Source Installation‡

To compile GL4Java, we still use MS Visual C++ 6.0 ! We also assume that Windows is installed under c:/WIN_NT ! The compiler flags are set to Pentium Pro (686 / PII) with the optimizimg Intel compiler !

Also Java2 SDK is installed under c:/java2 ! Also MS Java SDK is installed under c:/MSJAVASDK !

Please check all location in the makefile !

1. If you have installed cygwin32 and it's bash and make, etc... AND set it up very well (CLASSPATH, PATH, ...) you can invoke the makefile[b] with `make w32` to create all java-depended stuff and the C-wrappers.
2. Befor going any further, check if the files opengl32.dll glu.dll[a], glu32.dll exist in c:/WIN_NT/system32 OR c:/windows/system ! IF not get MS-OpenGL Lib (opengl32.dll, glu32.dll) !
3. Go with the Explorer to the directory Win32VC6 and open the workspace Win32VC6.dsw !
4. Now, you should fix some option in the project file, we used:

   The order of this path is important !

   INCLUDEPATH:

   ```
   c:/projects/java-1.1.X/GL4Java/CClassHeaders; \
   c:/projects/java-1.1.X/GL4Java/CNativeCode; \
   c:/java2/include; \
   c:/java2/include/windows \
   <the MSVC60 SDK PATH's ...>
   ```

   LIBPATH:

   ```
   c:/WIN_NT/system32; \
   <the MSVC60 SDK PATH's ...>
   ```

   WHERE: Java2 is installed under `c:/java2` GL4Java is installed under `d:/projects/java-1.1.X/GL4Java`
5. Now just compile each project and the targets are placed in the lib directory !
6. Copy the generated dll's (in the libs-directory) to your windows system32 directory, or (better) to your /java2/jre/bin directory .

[b]in the GL4Java directory, the main one

2.4 Hacking

For a complete description of the makefile invokations see 5.1.1, page 30.

You can call 'make x11' for Unix/X11, or 'make w32' for Windows or 'make mac' for Macintosh to create all wrapper classes, Java classes, etc. For Unix/X11 users the native libraries are created also.

If you want to modify the sources, you have to edit the .skel files !!! E.g.: You should modify 'gl4java/GLContext.java.skel' instead of 'gl4java/GLContext.java', because 'gl4java/GLContext.java' will be generated !!!! Check the 'makefile' for more details !

Again: Look for *.skel files and modify these, because they will be taken to create the target source files without the prefix .skel !!!

Documentation for sources will be created later !! Check gl4java.GLContext.html.

For a *step by step* instruction, please see chapter 2.3.2 on page 20.

Chapter 3

Design†

Since Version 2.0.0 R1 the object model of GL4Java is completly changed ! But it is very easy to port existing 1.X.X RX GL4Java stuff to the new object model. The script ../../demos/change2GL4JavaV2.sh will help.

GL4Java is designed to use the native OpenGL library, which should be installed on each platform. Instead of writing an own OpenGL library, we just write a wrapper to access the nativ platform dependend OpenGL library.

3.1 Native-Call to OpenGL

Native programs are written in a programing language, where the compiler is able to produce binary code for the platforms processor. These OpenGL programs will use the platforms OpenGL library. *Use* means, that they load the library and call the procedures which exists in the loaded library.

3.2 Java-Call to OpenGL

The java program should act like a native program. Java programs should use the native OpenGL library on the specific platform. The java program must load the wrapper library, which will load the OpenGL library. Like the OpenGL library, the wrapper library is platform depended and will be load in runtime, a so called dynamic library. The platform independed java OpenGL application needs the platform depended wrapper library and all librarys the wrapper library needs, e.g.: the OpenGL library etc. . To run a java OpenGL application, the user must have the OpenGL library installed and must install the wrapper library - that's all. The java OpenGL applications have the performance like an native OpenGL application. The only reason for a performance less than the native one are the costs for the java procedures (to modify data etc.). The passing through of data from java to OpenGL and vice versa needs no special converting features, thus the passing does not consume much time.

3.2.1 JNI to OpenGL

Data Type Mappings

Because java should pass its data without any time consuming convertion, we need a data type mapping style, where the OpenGL data types[2] fits in the java types and vice versa. The first and important criteria for a type match is the size of bytes. The secound and convinient criteria is to find aequivalent types in java, where less or no data converting must be done by the wrapper library and the java OpenGL programmer himself. Except for the marked OpenGL types we found a fine type matching which fullfill our needs. All *unsigned* types must match in a byte-size java aequivalent, so a special java-casting must be done by the java OpenGL programmer. Because java programmer uses arrays, we generate the GL4Java function for all Java-Array-Types, to achieve compatibility with the *GLvoid ** (see below).

OpenGL Data Types Mapping to Java Types.

Data Type	C-Type	JNI-Type	Java-Type	OpenGL Type
8-bit integer	unsigned char	jboolean	boolean	GLboolean
8-bit integer	signed char	jbyte	byte	GLbyte
8-bit unsigned integer	unsigned char	jbyte[a]	byte[a]	GLubyte,
16-bit integer	short	jshort	short	GLshort
16-bit unsigned integer	unsigned short	jshort[a]	short[a]	GLushort
32-bit integer	int or long	jint	int	GLint, GLsizei
32-bit unsigned integer	unsigned int or unsigned int	jint[a]	int[a]	GLuint, GLenum, GLbitfield
32-bit floating-point	float	jfloat	float	GLfloat, GLclampf
64-bit floating-point	double	jdouble	double	GLdouble, GLclampd
32-bit integer	void *	jbyteArray[b]	byte[][b]	GLvoid *
		jshortArray[b]	short[][b]	GLvoid *
		jintArray[b]	int[][b]	GLvoid *
		jfloatArray[b]	float[][b]	GLvoid *
		jdoubleArray[b]	double[][b]	GLvoid *
		jbooleanArray[b]	boolean[][b]	GLvoid *
		jlongArray[b]	long[][b]	GLvoid *

[a]No Java matching for unsigned, using extra conversion
[b]No Java matching for pointers, using any arrays

JNI Generation

The wrapper library which is called by the java application is specified by the Java Native Interface (JNI)[5]. All native functions, which should be called by the java application must be declared as a native function in a java class. E.g.:

```
public native void glClear ( int mask ) ;
```

The JNI compiler creates a header file for the C programming language, e.g. *foo.h*. This header file can be copied as an template to the C definition file, e.g. *foo.c*. This definition file can modified to add the needed function bodies. The function bodies, the functions, only has to call its C function out of the OpenGL library with a standard data type convertion.

In the begining we wrote some of the OpenGL wrapper functions by ourself, but later we learned that we can not guarante that all functions are wrapped and that all functions were almost bug free.

To simplify the writing of the apropiate wrapper functions, we created a C-Header-File to Java-Native-Declarations plus C-Wrapper-Functions generator. We named this generator *C2J*.

Details see chapter 5.2 on page 31.

3.3 OpenGL's Connection to the Windowing-System

OpenGL is specified as a 3D rendering machine. To view the rendering results, we need an visual output device, e.g. the monitor ;-). To simplify the monitor output OpenGL uses an existing operating system, which allows graphical output on the screen. The X-Window-System or MS-Windows is a well known an often distributed graphical interface, where the X-Window-System is avaiable on many platforms. The common tasks, means the minimum need, of a library which connects the OpenGL machine to the Windowing-System are:

- connect the OpenGL machine to a window
  - create a window
  - connect the window handel to the OpenGL handle/context
  - set up the window properties, e.g. colors
  - close the window
- re-rendering with OpenGL when the window must be refreshed

See the java class implementation in chapter 5.13 on page 34 to find the platforms window handle.

3.3.1 X-Window-System

OpenGL's interface to the X-Window-System (X)[4] is the GLX library. The GLX library[2] can connect the OpenGL machine[a] to X. GLX supports all criteria specified above.

3.3.2 MS-Windows

OpenGL's interface to MS-Windows (MSW) is the WGL library. The WGL library[3] can connect the OpenGL machine[b] to MSW. MSW supports all criteria specified above.

3.4 OpenGL's Connection to the AWT

The Abstract Window Toolkit (AWT) is the windowing system of the java machine. The AWT is included in the standard java package. The AWT for Unix and MS-Windows uses native functions of the windowing system, to take advantages of the existing windowing system. To achieve the platform window-handle, we use the (since GL4Java Version 1.1.0) the following classes:

- sun.awt.DrawingSurface

[a] a instance of the OpenGL machine, the OpenGL context - in detail

[b] a instance of the OpenGL machine, the OpenGL context - in detail

- sun.awt.DrawingSurfaceInfo
- sun.awt.X11DrawingSurface (for X11/motif)
- sun.awt.Win32DrawingSurface (for Windows 32-bit)

Of course, these classes are not explicit for user purposes, but they do provide us with interfaces to achieve the needed native window handle. We got this technique from Java3D's Canvas3D (decompiled it ...). You can see this implementation in:

- sun/awt/motif/X11HandleAccess.java
- sun/awt/window/Win32HandleAccess.java

3.5 OpenGL Mapping to Java

The JNI and data-type mapping (see 3.2.1 on page 26) is allready specified above.

What still need to be specified is the place of the java OpenGL native function declarations and which OpenGL librarys should be wrapped.

3.5.1 Convinient Placement

Because OpenGL is not an object-orientated API, OpenGL programmers does not use methods. To support the same OpenGL environment, we will use a java class, which contains:

- window control
- OpenGL functions

3.5.2 Keeping the OpenGL C-API

The Naming Convention

The GL4Java interface supports the OpenGLFrame class, which contains all OpenGL and GLU functions (glut in the near future :-) ! The naming convention of all gl* and glu* functions in the OpenGLFrame class is exactly the same as in the original, so there are no problems in porting C code. Datatypes java does not support will by supported by binary compatible types, e.g. for *unsigned byte* in *glColor4ubv*, GL4Java uses a *byte*-array !

Chapter 4

Development Environment

We use a directory structure as described above (see 2.2, page 10). Many well known Unix tools will be used, included some java tools.

4.1 Unix Tools†

We use the follwing standard tools as our develeopment environment:

1. c-compiler and linker (GNU-C, ...)
2. debugger (gdb, xxgdb)
3. JavaCC[9] (a java lex & yacc parser generator)
4. bash, make (GNU or other)
5. tar & gzip as our archive toolkit

4.2 Windows Tools‡

All above named Unix tools are avaiable under MS-Windows 32 with the GNU-Win32 project by Cygnus[11]. We need an additional MS-Windows compiler, MC-Visual-C++ or Borland-C++ are appropiate MS-Windows compilers. We took the Borland compiler, because of it's accessible, but we guess it does not make a different.

Chapter 5

Implementation

5.1 Makefile†

To manage the project, we wrote a makefile (used with make). Our makefile is able to generate the complete code, the demos, the documentation and the archived packages. The makefile resides in the projects root directory and is splitten into two files:

- makefile (where all common rules resides)
- symbols.mak (where all platform depended rules and macros resides)

A symbols.mak file must be created or copied & edited for each platform and linked to symbols.mak. We support some symbols.mak files like:

- symbols.mak.aix
- symbols.mak.linux
- symbols.mak.solaris
- symbols.mak.win32

5.1.1 Makefile Invokation

Here is the part of the makefile, which describes the different actions, which are possible !

```
#
# for general creation (java + native-lib) invoke:
#
# make x11 : create java and native lib for unix/x11
# make w32 : create java and native lib for windows32
# make mac : create java and native lib for Macintosh
#
#
# to copy the class-files to DEST_CLASSES_DIR, invoke:
#
# make classcpy
#
#
# to save the native-library invoke after general creation:
```

```
#
# make unix2binpkg : put the created unix-lib to the binpkg-dir
# make win2binpkg : put the created win-lib to the binpkg-dir
# make mac2binpkg : put the created macintosh-lib to the binpkg-dir
#
#
# to create the complete html documentation invoke
#
# make htmldoc : unix
# make javadoc : unix (javadoc only)
# make htmldocw32 : win32
# make javadocw32 : win32 (javadoc only)
#
#
# to put all together as an tar-gzip archive in the archive-dir, invoke:
#
# make archiv
#
#
# for cleanup (without archive and binpkg !!) invoke:
#
# make clean : only temp-files (java, native)
# make cleannative : only temp-files (native)
# make cleanall : all temp-,java-,and native-files
#
```

5.1.2 Make-Scripts to generate the librarys†

To support a platform independed creation for the wrapper libraries for the different Unix platforms, we wrote a shell script therefor, which will be invoked by the makefile.

The following files contains the apropiate scripts:

```
mklibs/mkexp.aix
mklibs/mklib.aix
mklibs/mklib.linux
mklibs/mklib.solaris
mklibs/mkslib.aix
mklibs/mkslib.linux
mklibs/mkslib.solaris
```

5.2 C2J Version 1.0†

As decribed in chapter 3.2.1 on page 26, we use C2J as our OpenGL wrapper functions generator. C2J is our JNI and MSJDirect file generator. C2J creates the java native function declarations and the C functions. C2J gets a modified Mesa-OpenGL header file as it's input and generates the java native declarations in one file, and the complete C functions in another file. C2J is written in an Extended Backus Naur Form (EBNF) of the JavaCC[9]parser generator. We modified the C parser provided with the JavaCC to write C2J.

The C2J syntax can seen in GL4Java/C2J/C2J.jj !

The C2J output files are glued with the other parts needed by the java class and the C definition-file to the complete sources.

The new C2J Version supports the following new features:

1. Only the glGetString, gluGetString and gluErrorString functions must be written manualy now !!!!

2. Better compatibility with *GLvoid ** arguments. This means

   ```
   "byte[]", "short[]", "int[]",
   "float[]", "double[]", "boolean[]", "long[]"
   ```

   functions are generated !

3. Better MS-JDirect (MS-JVM) integeration !

The source files are:

`C2J/C2J.jj`	the parser definition (input for JavaCC)
`C2J/makefile`	the lokal makefile
`C2J/gl-enum-auto.orig`	the OpenGL enumeration part, C2J input
`C2J/gl-proto-auto.orig`	the OpenGL declaration part, C2J input
`C2J/glu-enum-auto.orig`	the GLU enumeration part
`C2J/glu-proto-auto.orig`	the GLU declarartion part
`C2J/manual/gl-manualCoded.orig`	the GL manual coded C2J input
`C2J/manual/glu-manualCoded.orig`	the GL manual coded C2J input part
`C2J/manual/gl-enum-manualCoded.java`	the GL manual coded java part
`C2J/manual/glu-enum-manualCoded.java`	the GLU manual coded java part
`C2J/manual/glu-manualCodedImplJNI.c`	the GLU manual coded c part
`C2J/manual/glu-manualCodedImplJNI.h`	the GLU manual coded c part
`C2J/manual/glu-manualCodedImplJNI.java`	the GLU manual coded java part
`C2J/manual/glu-manualCodedImplMSJVM.java`	the GLU manual coded java part
`C2J/manual/glu-manualCodedVirt.java`	the GLU manual coded java part
`C2J/manual/gl-manualCodedImplJNI.c`	the GL manual coded c part
`C2J/manual/gl-manualCodedVirt.java`	the GL manual coded java part
`C2J/manual/gl-manualCodedImplJNI.java`	the GL manual coded java part
`C2J/manual/gl-manualCodedImplMSJVM.java`	the GL manual coded java part

5.3 GLEnum

Since Version 2.0.0, GLEnum is the interface, where all C-API OpenGL enumerations are defined as static final. This class is generated, for details see chapter 5.2 on page 31.

5.4 GLUEnum

Since Version 2.0.0, GLUEnum is the interface, where all C-API GLU enumerations are defined as static final. This class is generated, for details see chapter 5.2 on page 31.

5.5 GLFunc

Since Version 2.0.0, GLFunc is the interface, where all C-API OpenGL functions are specified. This class is almost generated, for details see chapter 5.2 on page 31.

5.6 GLFuncJauJNI

Since Version 2.0.0, GLFuncJauJNU is one implementation of the interface GLFunc. This class is almost generated, for details see chapter 5.2 on page 31.

5.7 GLUFunc

Since Version 2.0.0, GLUFunc is the interface, where all C-API GLU functions are specified. This class is almost generated, for details see chapter 5.2 on page 31.

5.8 GLUFuncJauJNI

Since Version 2.0.0, GLUFuncJauJNU is one implementation of the interface GLU-Func. This class is almost generated, for details see chapter 5.2 on page 31.

5.9 GLContext

Since Version 2.0.0, GLContext is the central java class for the OpenGL bindings and it is defined as a package member of the java package gl4java. GLContext trys to connect the OpenGL machine to the AWT window of the given Component, which is represented by the platform specific window.

GLContext also can loads all the needed libraries and GLFunc and GLUFunc implementations.

These are the basic source files, look at 5.12, page 34.

5.10 GLFrame

Since Version 2.0.0, GLFrame is moved out !

5.11 GLCanvas

Since Version 2.0.0, GLCanvas is one AWT toolkit implementation for the OpenGL bindings and it is defined as a package member of the java package gl4java.awt. GLCanvas is derived from AWT's Canvas. While creation GLCanvas trys to create a GLContext instance to connect the OpenGL machine to the AWT window, which is represented by the platform specific window.

Also the refresh signal is routed to OpenGL, so OpenGL can re-render.

These are the basic source files, look at 5.12, page 34.

5.12 GLAnimCanvas†‡

Since Version 2.0.0, GLAnimCanvas is one AWT toolkit implementation for the OpenGL bindings and it is defined as a package member of the java package gl4java.awt. GLAnimCanvas is derived from GLCanvas. GLAnimCanvas addds thread support for animation, thats where the Anim stands for in GLAnimCanvas.

`gl4java/GLEnum.java`	The OpenGL Enumerates as an interface.
`gl4java/GLUEnum.java`	The GLU Enumerates as an interface.
`gl4java/GLFunc.java`	The OpenGL functions interface.
`gl4java/GLUFunc.java`	The GLU functions interface.
`gl4java/GLFuncJauJNI.java`	One OpenGL functions implementation.
`gl4java/GLUFuncJauJNI.java`	One GLU functions implementation.
`gl4java/GLContext.java`	The OpenGL-Context Window service, and lib and class loa
`gl4java/GL4JavaInitException.java`	Our Exception
`gl4java/awt/GLCanvas.java`	One OpenGL AWT binding
`gl4java/awt/GLAnimCanvas.java`	Another OpenGL AWT binding
`gl4java/jau/awt/WinHandleAccess.java`	Interface to the platform's windowing system

Java OpenGL programmers can derive their classes either from GLCanvas. See the annotated Example in chapter 6.1 on page 6.1.

See the Html-Source documentation for details (see 7.1 on page 61).

5.13 Extensions for the sun-package†‡

Like described in chapter 3.3 on page 27, we need to find out the platforms window handle to connect it with the OpenGL machine.

The used motif and windows sub-packages are not defined as a standard, but they exist as long Java 1.0.2 exist and they are the only way to find out the unique AWT window's platform-window-handle from the java-side.

See the Html-Source documentation for details (see 7.1 on page 61).

Here are the apropiate source files:

`sun/awt/motif/X11HandleAccess.java`	X-Window-System glue
`sun/awt/windows/Win32HandleAccess.java`	MS-Windows glue
`sun/awt/macintosh/MacHandleAccess.java`	Macintosh glue

See also 3.4 on page 27.

5.14 The Native Wrapper-Library Code†‡

As described in chapter 3.3 on page 27 we need a connection between the platform depended window system and OpenGL.
Here are the apropiate source files:

`CNativeCode/OpenGL_Win32.c`	MS-Windows window access
`CNativeCode/OpenGL_X11.c`	X-Window-System window access
`CNativeCode/OpenGL_misc.c`	common miscelaneos sources
`CNativeCode/OpenGLU_funcs.c.skel`	common GLU source-part
`CNativeCode/OpenGLU_funcs.c`	glued source from the *.skel and C2J output
`CNativeCode/OpenGL_funcs.c.skel`	common GL source-part
`CNativeCode/OpenGL_funcs.c`	glued source from the *.skel and C2J output
`CNativeCode/gl.aix.not.c`	Dummy GL functions not defined in AIX OpenGL library
`CNativeCode/gl.aix.not.h`	Dummy GL functions not defined in AIX OpenGL library
`CNativeCode/gl.win32.not.c`	Dummy GL functions not defined in MS-Windows OpenGL library
`CNativeCode/gl.win32.not.h`	Dummy GL functions not defined in MS-Windows OpenGL library
`CNativeCode/winstuff.h`	Miscelaneous declarations missing in MS-Windows

See the Source documentation for details.

Chapter 6

Examples†‡

6.1 The olympicCvs GLAnimCanvas derivation

```
/**
 * @(#) olympicCvs.java
 */

import gl4java.awt.GLAnimCanvas;
import java.awt.*;
import java.awt.event.*;
import java.lang.Math;


class olympicCvs extends GLAnimCanvas
{
/**
 * Instead of using suspend (JAVA2)
 *
 * @see run
 */
protected boolean threadSuspended = true;

static final double M_PI= 3.141592654;
static final double M_2PI= 2*3.141592654;

static final int
XSIZE=   100,
YSIZE=   75,
RINGS= 5,
BLUERING= 0,
BLACKRING= 1,
REDRING= 2,
YELLOWRING =3,
GREENRING =4,
BACKGROUND =8,
BLACK = 0,
RED=1,
GREEN=2,
YELLOW=3,
BLUE=4,
MAGENTA=5,
CYAN=6,
WHITE=7;

byte rgb_colors[][];
int mapped_colors[];
float dests[][];
float offsets[][];
float angs[];
float rotAxis[][];
int iters[];
int theTorus;

        float lmodel_ambient[] = {0.0f, 0.0f, 0.0f, 0.0f};
        float lmodel_twoside[] = {0.0f, 0.0f, 0.0f, 0.0f};
        float lmodel_local[] = {0.0f, 0.0f, 0.0f, 0.0f};
        float light0_ambient[] = {0.1f, 0.1f, 0.1f, 1.0f};
        float light0_diffuse[] = {1.0f, 1.0f, 1.0f, 0.0f};
        float light0_position[] = {0.8660254f, 0.5f, 1f, 0f};
        float light0_specular[] = {1.0f, 1.0f, 1.0f, 0.0f};
        float bevel_mat_ambient[] = {0.0f, 0.0f, 0.0f, 1.0f};
        float bevel_mat_shininess[] = {40.0f, 0f, 0f, 0f};
        float bevel_mat_specular[] = {1.0f, 1.0f, 1.0f, 0.0f};
        float bevel_mat_diffuse[] = {1.0f, 0.0f, 0.0f, 0.0f};

public olympicCvs (int w, int h,
   String glClass, String gluClass )
{
super(w, h, glClass, gluClass );
}

        public void init()
{
rgb_colors=new byte[RINGS][3];
mapped_colors=new int [RINGS];
dests=new float [RINGS][3];
offsets=new float[RINGS][3];
angs=new float[RINGS];
rotAxis=new float[RINGS][3];
iters=new int[RINGS];

int i;
float top_y = 1.0f;
float bottom_y = 0.0f;
float top_z = 0.15f;
float bottom_z = 0.69f;
float spacing = 2.5f;

ReInit();
for (i = 0; i < RINGS; i++) {
  rgb_colors[i][0] = rgb_colors[i][1] = rgb_colors[i][2] = (byte)0;
}
rgb_colors[BLUERING][2] = (byte)255;
```

```
rgb_colors[REDRING][0] = (byte)255;
rgb_colors[GREENRING][1] = (byte)255;
rgb_colors[YELLOWRING][0] = (byte)255;
rgb_colors[YELLOWRING][1] = (byte)255;
mapped_colors[BLUERING] = BLUE;
mapped_colors[REDRING] = RED;
mapped_colors[GREENRING] = GREEN;
mapped_colors[YELLOWRING] = YELLOW;
mapped_colors[BLACKRING] = BLACK;

dests[BLUERING][0] = -spacing;
dests[BLUERING][1] = top_y;
dests[BLUERING][2] = top_z;

dests[BLACKRING][0] = 0.0f;
dests[BLACKRING][1] = top_y;
dests[BLACKRING][2] = top_z;

dests[REDRING][0] = spacing;
dests[REDRING][1] = top_y;
dests[REDRING][2] = top_z;

dests[YELLOWRING][0] = -spacing / 2.0f;
dests[YELLOWRING][1] = bottom_y;
dests[YELLOWRING][2] = bottom_z;

dests[GREENRING][0] = spacing / 2.0f;
dests[GREENRING][1] = bottom_y;
dests[GREENRING][2] = bottom_z;

theTorus = gl.glGenLists(1);
gl.glNewList(theTorus, GL_COMPILE);
FillTorus(0.1f, 8, 1.0f, 25);
gl.glEndList();

gl.glEnable(GL_CULL_FACE);
gl.glCullFace(GL_BACK);
gl.glEnable(GL_DEPTH_TEST);
gl.glClearDepth(1.0);

gl.glLightfv(GL_LIGHT0, GL_AMBIENT, light0_ambient);
gl.glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse);
gl.glLightfv(GL_LIGHT0, GL_SPECULAR, light0_specular);
gl.glLightfv(GL_LIGHT0, GL_POSITION, light0_position);
gl.glEnable(GL_LIGHT0);

gl.glLightModelfv(GL_LIGHT_MODEL_LOCAL_VIEWER, lmodel_local);
gl.glLightModelfv(GL_LIGHT_MODEL_TWO_SIDE, lmodel_twoside);
gl.glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
gl.glEnable(GL_LIGHTING);

gl.glClearColor(0.5f, 0.5f, 0.5f, 0.0f);

gl.glMaterialfv(GL_FRONT, GL_AMBIENT, bevel_mat_ambient);
gl.glMaterialfv(GL_FRONT, GL_SHININESS, bevel_mat_shininess);
gl.glMaterialfv(GL_FRONT, GL_SPECULAR, bevel_mat_specular);
gl.glMaterialfv(GL_FRONT, GL_DIFFUSE, bevel_mat_diffuse);

gl.glColorMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE);
gl.glEnable(GL_COLOR_MATERIAL);
gl.glShadeModel(GL_SMOOTH);

gl.glMatrixMode(GL_PROJECTION);
glu.gluPerspective(45f, 1.33f, 0.1f, 100.0f);
gl.glMatrixMode(GL_MODELVIEW);
                glj.gljCheckGL();
}

public void display()
{
  int i;

  /* Standard GL4Java Init */
  if( glj.gljMakeCurrent(true) == false )
  {
System.out.println("problem in use() method");
System.exit(0);
return;
  }

  gl.glPushMatrix();

  gl.glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glu.gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);

  for (i = 0; i < RINGS; i++) {
    gl.glColor3ubv(rgb_colors[i]);
    gl.glPushMatrix();
    gl.glTranslatef(dests[i][0] + offsets[i][0], dests[i][1] + offsets[i][1],
      dests[i][2] + offsets[i][2]);
    gl.glRotatef(angs[i], rotAxis[i][0], rotAxis[i][1], rotAxis[i][2]);
    gl.glCallList(theTorus);
    gl.glPopMatrix();
  }

  gl.glPopMatrix();
  glj.gljSwap();
  glj.gljCheckGL();
  glj.gljFree();
}

public void animationCalc()
{
          int i, j;

  for (i = 0; i < RINGS; i++) {
    if (iters[i]!=0) {
      for (j = 0; j < 3; j++) {
```

```
offsets[i][j] = Clamp(iters[i], offsets[i][j]);
      }
      angs[i] = Clamp(iters[i], angs[i]);
      iters[i]--;
    }
  }
          if (iters[0]==0)
  {
  shallWeRender=false;
  }
}

public void ReInit()
{
  int i;
  float deviation;

  deviation = MyRand() / 2;
  deviation = deviation * deviation;
  for (i = 0; i < RINGS; i++) {
    offsets[i][0] = MyRand();
    offsets[i][1] = MyRand();
    offsets[i][2] = MyRand();
    angs[i] = (float) (260.0 * MyRand());
    rotAxis[i][0] = MyRand();
    rotAxis[i][1] = MyRand();
    rotAxis[i][2] = MyRand();
    iters[i] = ( int ) (deviation * MyRand() + 60.0);
  }
}

public void FillTorus(float rc, int numc, float rt, int numt)
{
  int i, j, k;
  double s, t;
  float x, y, z;

  for (i = 0; i < numc; i++) {
    gl.glBegin(GL_QUAD_STRIP);
    for (j = 0; j <= numt; j++) {
      for (k = 1; k >= 0; k--) {
s = (i + k) % numc + 0.5;
t = j % numt;

x = (float) Math.cos(t * M_2PI / numt) * (float) Math.cos(s * M_2PI / numc);
y = (float) Math.sin(t * M_2PI / numt) * (float) Math.cos(s * M_2PI / numc);
z = (float) Math.sin(s * M_2PI / numc);
gl.glNormal3f(x, y, z);

x = (rt + rc * (float) Math.cos(s * M_2PI / numc)) * (float) Math.cos(t * M_2PI / numt);
y = (rt + rc * (float) Math.cos(s * M_2PI / numc)) * (float) Math.sin(t * M_2PI / numt);
z = rc * (float) Math.sin(s * M_2PI / numc);
gl.glVertex3f(x, y, z);
      }
    }
    gl.glEnd();
  }
}

public float Clamp(int iters_left, float t)
{

  if (iters_left < 3) {
    return 0.0f;
  }
  return (iters_left - 2) * t / iters_left;
}

public float MyRand()
{
  // return 10.0 * (drand48() - 0.5);
  return (float) ( 10.0 * (Math.random() - 0.5) );
}

}
```

6.2 Further Examples†‡

Please look in the directory demos to look at the samples. You can start the glDemosCvs (gl4java.awt.GLAnimCanvas) java application to see them all running.

Chapter 7

Further Documentation†

7.0.1 Readme

```
I modified Adam King's version of his INSTALL guide to fit the needs
to my version.

I added my CHANGES to Adam King's CHANGES.

I would be very lucky, if i get some response, critic, inspirations,
bugfixes and participation.

I changed (see CHANGES) the naming convention for a better
transparence and porting-ability (sorry for my english)
- so i changed the lib-name to GL4Java !

<Please forgive me for so many I's :>

The following Unices were succesfully testes:

o Linux (x86) 2.X
o SunOS (Sparc) 5.X
o AIX 4.2 (RS/6000)
o SGI Irix

o Macintosh (check http://www.jausoft.com/gl4java.html)
o Windows NT 95/98 (x86) 4.0 (sun/netscape)
o Windows NT 95/98 (x86) 4.0 (msjvm/ie40)

May be - WE can find a way for a standard OpenGL for Java API !

As you can see, OpenGL-Rendering can be integrated in Java-Programs
very well. You can use the normal java paint and event - functions
as you used display and reshape in glut !

See also LICENSE.txt for the license of GL4Java !

Sven Goethel
January 1999
April 1999
```

```
September 1999

mailto:sgoethel@jausoft.com
www   : http://www.jausoft.com
voice : +49-0521-2399440
fax   : +49-0521-2399442
```

7.0.2 Changes

```
Last Changes: 6. March 2000 (Version 2.2.0 - Release 0 )

-----------------
TOP  = old
DOWN = new
-----------------

July 24, 1997:

        Added genLists(), callList(), newList()
        Fixed a couple of minor bugs

------- Sven Goethel - September 1997

August/September 1997:

        o renamed the directory to OpenGL4Java
        o renamed the library to GL4Java !
        o created a new makefile
          (now just one for the lib, and one for the demos)
        o make support about the symbols.mak.<machine> descriptor-files
          for linux(gl and mesa), solaris(mesa), aix
        o portet all 1.0.2 native calls to 1.1.X JNI calls
        o changed the naming convention to the strict convention
          of OpenGL. IE. glColor3f, gluPerspective, GL_DEPTH, etc.
        o renamed most of all java and c overheads in the wrapper
          functions - and passing all arguments straight to the
          OpenGL-Library.
        o renamed use and swap to gljUse and gljSwap,
          according to a correct naming convention
          - with glj for gl java !
        o moved all gl-functions to OpenGLFrame for a more easier porting.
          no class instance prefix must be added !
          now gljInit creates the OpenGL initialisation,
          and gljIsInit checks for OpenGL initialisation !
        o added some glu* and gl* functions
        o added some demos in the directory demos
          also added a c-version for the demos to be able to
          compare the GL4Java with OpenGL.
        o added X-Error functions to be informed about X11 errors.
        o added more selective X procedures to get the true
          java-window. this is done while porting GL4Java to aix.
          this porting is not successfully done.
```

```
10. SEPT 1997
        o taken some improvements from jogl v 0.5 for the
          X-Window System functions
        o renamed the java-package an the top-level directory to 'GL4Java'
          to avoid name-concurrency with other implementations
        o the actual top-level directory is now GL4Java,
          no more java-1.1.X and mklib* links were archived
          (to avoid confusion).
        o changed the 'javac' debug-flag '-g' to optimization '-O'
        o openOpenGL now greps the correct windows width and height
          from the JavaWin-Size
        o added gljResize to resize the GL-Window
          This is done with the native call XResizeWindow.
        o GLFrame is a ComponentListener,
          so a Resize _IS_ be tracked and passed to gljResize.
          If the java win is resized, the GL-Win is resized either !
          Your derived GLFrame-Method componentResized
          should call super.componentResized(e) to be shure
          that the super class makes the resize !
          Your derivation must NOT addComponentListener, because super does ..
        o improved cube demo with eyes control panel
        o actual dummyGL.java skelleton for your GL4Java portings ..

14. SEPT 1997

        o spended very much time to port GL4Java to AIX 4.2.
        o CHANGES for AIX:
                o must use X11 lib in : '/usr/lpp/X11/lib'
                  - and so i used the GL lib in : '/usr/lpp/OpenGL/lib' either.
                  The X11 lib located in '/usr/lib' does not work with
                  the JDK :-(.
                o must use JDK 1.1.1 OR
                  JDK 1.1.2 without JIT (export JAVA_COMPILER=).
                  The JIT forces the program to exit.
                o must use the Java-Window itself, instead of
                  creating an own child-win !
                  Because the color's are not established in the
                  own-win.
        o NOW GL4Java runs on AIX 4.2 :-)))
        o corrected skelleton 'demos/dummyGL.java'
        o modified GLFrame class - initialisation will be done
          in the constructor. so the constructor takes now two more
          arguments, the width and the height of the Window !

16. SEPT 1997 (Version 0.3)

        o added compiler FLAGS :
                o __CREATE_OWN_WINDOW__
                  NOT to use the original Java Window,
                  so a color-window created by our own will be used  !
                  This FLAG is actually NOT used for AIX, LINUX and SOLARIS !

03. SEPT 1997 (Version 0.4)
```

```
o created mklibs/mklib.<os> -> mklibs/mkslib.<os>
  to create * ONLY * shared librarys.
  This is a step for binary distribution !
  Also all symbols.mak.<os> uses for other shared librarys
  (MesaGL, m, MesaGLU, ...) * ONLY * the '-l<BaseLibName>'
  construct - so the created shared library will find the other
  librarys via the LD_LIBRARY_PATH. E.g.: '-lMesaGL' !
  I must say i am not that sure about all the shared dynamic
  library stuff - but i testet it for solaris, ***
  - and it works. So we can download the 'libGL4Java.so' library
  for each target/os - check whether all libs are in
  the LD_LIBRARY_PATH - and it's done.
o checked in demos if we can use 'Thread.yield()' while
  the thread-running-loop after direct re-rendering
  (direct -> not via 'repaint' - via direct method call) !
  This works fine for: aix, solaris, ***!
o setup the homepage at
  <http://parallel.fh-bielefeld.de/pv/news/gl4java.html>.
  Some syntax fixes ;-) and * BINARY * support for
  solaris, *** !
o Preparation for Win32.
  We just could do it with BC++ 5.0, instaed of using
  gnuwin32 (cygnus32/mingw32).
  We tried the GNU compiler much, but we could not compile
  a OpenGL demo, because we could not link with the DLLs :-(((.
o removed the single concatenated list of a data structur,
  which holds the gl-context, win-handle.
  Added those handles in GLFrame (direct ;-).
  This is mainly don for Win32 support !
o Added glViewport direct into GLFrames componentResize
  callback-function. So derivations do not need to do that !
o Added glGetError and gljCheckGL, where the last performs
  a glGetError AND throws - catches an exception to print the
  source lines - where the check was performed
o All glGet*v does resturn their values back to java - now ;-)
o Code Clean Up :
        * no more java actions in wrapper class
        * just a passthrough of datas in native class
        * because of the non-data-parsing (speedup),
          we will check all GL <-> JNI type mappings at startup once.
o Added Jogl's sun.awt.motiv.* sun.awt.windows.* classes,
  to support window handle grabbing by java !
  This makes it possible to show the native method's the window
  handle, instead of using the window-title to identify.
  Now it is possible (theoretical) to use a canvas (like Jogl)
  as a OpenGL drawable component instead of a frame. BUT we believe
  that this is not that usefull - or is it ?!
o GLFrame has the public method display (like glut),
  which will be called automatical by GLFrame's paint !
  So the derivation can easy use display.
  Also paint is needed for grabbing the window-handle by java,
  befor GLFrame initializes the OpenGL-Context !
o All demos and the 'dummyGL.java' are updated !
08. OKT 1997 (Version 1.0 beta1 )

        o Because of the lower described Major changes,
          we moved to version 1.0 !

        o Nearly *ALL* GL and GLU functions are supported.
          All JNI interfaces are no created by cross-compilation
          of manually fixed MESA Header-Files with the
          cross-compiler C2J !
          C2J is a derivation of the BNF C-Parser delivered with
          the SUN's javaCC 0.6.1 !
          So, C2J (C2J.jj) can be compiled with jdk and javacc.
          C2J creates with Mesa's prototypes (a bit manipulated,
          so they will fit C2J) the needed java-prototypes AND
          the c-interface for the JNI-library !
          C2J is shipped within this package !
          The JNI-Interface just mapps the function.
          Neither the java class nor the JNI-Lib checks arguments !
          We perform a type mapping check to see if the
          suggested GL-types fits to JNI-types at GL-Initialisation !

        o The sun.awt.motiv.* and sun.awt.windows.* classes (java)
          are taken from Jogl V 0.7 (THANXS).
          We renamed 'em a bit, so NO naming confusion will occure,
          while using GL4Java AND Jogl in the same CLASSPATH !

        o GL4Java takes the Window-Handle direct from java
          (also taken from Jogl -- puhh).
          Because the use of the sun.* package, we do not know,
          if the platform specific classes do exist tommorow.
          SUN said "DO NOT USE THE sun.* classes ;-))".
          Anyway - we still can rewind the procedure to get the window handle.
          Ooops - I forgot - SO it's poosible now to have many frames
          with the same NAME (Yes we still using Frames) !

        o The Windows port was taken from Jogl V 0.7 (THANXS)
          But still not compiled ....

        o The function gljIsInit  is obsolete,
          use gljUse now !

        o All demos are updated AND wave is added !
          (Textures and GLU functions do work ;-))

        o Yes - We added some HTML documentation !


10. OKT 1997 (Version 1.0 beta2 )

        o Solaris SunOs 5.5 is created and all the following
          changes  and bugfixes were found while porting to Solaris !
          The bugs could be found, because Solaris has a multithreading
          X11 Server ! The development under linux does not show
          those reentrance-problems all the time.

        o 'componentResize' won't call glViewport direct.
          It will set a resize flag, and the next paint invocation
          will doing the resize-action ...

          o paint now invokes 'sDisplay' instead of invoking
          'display' direct.
          'sDisplay' has a semaphore, which avoids reentrance of
          the 'display' method !
          If 'sDisplay' has the semaphore, it will calls 'display' !
          Also sDisplay is a synchronized method,
          which will push the 2nd thread to the wait-list
          and notify the one after completion !

        o All demo's are now updated
          (invokation of 'sDisplay' in run-method, instead of 'display')
          AND are WORKING well (as green- and native-threads !).

15. OKT 1997 (Version 1.0 beta5 )

        o Added void reshape (int,int) for the resize.
          This is not like the deprecated 1.0 API,
          but like gluts reshape callback function !
          reshape will be invoked, when sDisplay
          check the flag mustResize and its true.
          mustResize is set by componentResize !
          GLFrame.reshape implements the default GL behavior,
          but if you want to - you can overload it now ;-)))

        o Added GLFrame as a windowListener
          to act for windowClosing !

          windowClosing calls gljDestroy (see lower) AND dispose !

        o Added gljFree, gljDestroy .
          gljFree releases the GL Context AND
          gljDestroy destroys it !

          gljFree is always invoked after sDisplay called display !
          This is needed while having some GL Context problems in
          glXMakeCurrent (or wglMakeCurrent ;-) for some plattforms.
          The Java Event Thread wants to set gc as current, but its allready
          has a gc set current. So we just release (NOT destroy) the GC !

          gljDestroy is invoked by windowClosing ;-)))

        o setVisible MUST be invoked by the subclasss constructor
          after calling the super-class constructor at the end.
          We have saw that the class global arrays with agregat-initializing
          were not initialized yet :-( !

        o updated all demos AND added glLogo and glDemos.
```

```
          glDemos manages invocation of all demos ;-))
          Incl. multi-threading :-)))

19. Okt 1997 (Version 1.0)

          o 'sDisplay' changes.
          'sDisplay' now has NO more semaphore's.
          'sDisplay' actually uses 'gljFree' (see above) to avoid
          GL Context problems.
          'sDisplay' sets the actual Thread to highes priority before
          calling 'display' AND resets its priority after 'display'.
          We now do not have to care about another reentranced thread.
          We beleave (;-) 'gljFree' and 'priority settings' are enough.
          'sDisplay' is still a 'synchronized' method !

        o Multi Threading  works

        o X11 (Linux) works

        o X11 (Solaris, Aix) still have to be checked !
          (but we guessing that they will work ;-)

        o Windows 32 (Windows NT, Windows 95) works
          (We use SGI DLL's !)


21. Okt 1997 (Version 1.0.0)

        o X11 (SunOS 5.X - Solaris) works with green threads

        o Found out that if we use Mesa, we need to use Version 2.4 or higher

        o Added the BUGFIX Version Number !


03. Nov 1997 (Version 1.0.1)

        o fixed mkslib.<OS> scripts for all OS !

        o tested AIX V 4.2 port - it works - but should work better :-))

23. Feb 1998 (Still Version 1.0.1)

        o fixing the documentation (LaTeX stuff)

        o using JavaCC Version 0.7.1, modified C2J for this purpose

24. JUNI 1998 (Version 1.0.2)

        o using tar archives in binpkg for the unix-libs ..
          no symbolic-links are needed anymore, so we can put this
          project on a vfat filesystem for convinient crossdevelopment

        o changed all demos.
```

```
          now, the demos (included the dummyGL.java)
          using a frame-per-secound method with repaint and sleep
          to create animation !

        o if the native library GL4Java is not found,
          GL4Java32 will be loaded.
          GL4Java32 is only for Windows32 users,
          and uses only MS native libs: opengl32.dll and glu32.dll
          (see chapter Binary Installation in the documentation) !!!

        o improvements in the makefile,
          better distinguish of win32 and unix users !
          Now win32-users can REALY use the makefile :-)

        o LaTeX-documentation update, with a better
          installation description for unix & win32

        - This work is done to prepare for rework GL4Java.
          GL4Java should support canvas :-)

06. JULI 1998 (Version 1.1.0)

        o Creating new GL4Java top class GLCanvas,
          which support and is derived from Canvas.
          Now it is possible, to use a canvas to render OpenGL :-)
          New Files:
                o GL4Java/GLCanvas.java
                o LIBRARIES: GL4JavaCanvas AND GL4JavaCanvas32

        o debugging of some JNI functions - thanxs to Java 1.2 beta 3,
          which shows me the errors :-)

        o Using a new way to achieve the native Window-Handle.
          Fetched the methods from the Canvas3d Class outta Java3d-Package :-)
          Now GL4Java runs with Java 1.1.6 and higher (also Java 1.2 beta 3)
           - changed sun.awt.motif.*DataAccess -> sun.awt.motif.X11HandleAccess
           - changed sun.awt.windows.*DataAccess -> sun.awt.motif.Win32HandleAccess
           - changed GL4Java.*DataAccess -> sun.awt.motif.WinHandleAccess

        o Some rework for makefiles :-)

        o LaTeX Documentation-Update :-)
           - Now we do support a postscript version !

        o GLCanvas examples (check glDemosCvs)

        o The precompiled Win32 versions are made with:
                - BC5, Intel-Compiler, Optimisations, Pentium-Optimisations !

        o added a MSVC 5.0 Project directory WIN32VC.
          i used this one to perfom debugging :-)


15. JAN 1999 (Version 1.1.0 - Release 2 )
```

```
o added a SunOS 5.X patch for Suns OpenGL library
  included a new define _GL_SOLARIS_ with its
  c-stubs in CNativeCode directory

o Linux: changed to glibc and Mesa3.0 with pthreads

o added precompiled SunOS lib linked against mesa

o the demos/native/x11 are now compiler-clean
  please check the makefile

o changed the license to lgpl, see LICENSE.TXT

o documentation changes:
- german OpenGL intro is not more included in the html-version
- now using LaTeX2Html Version 98.2 beta8

25. JAN 1999 (Version 1.1.0 - Release 3 )

o added glXWaitX after glXWaitGL in OpenGL_X11.c.skel

o tested with MetroX Metro-Extreme-3D (OpenGL - Hardwareaccelerated)
  this works, but there is flickering while animation,
  and i do not know why - checked it with native/x11/glX*
  dem testGL2 where no flickering is :-(
  Also Metro3D does not have any GL-EXT funcs ...

o merged Leung Yau Wais Win32 patch.
  he checked minimized the "not implemented Win32" functions :-)
  Thanxs Leung !
  The problem was to use BC5 OpenGL include files,
  which are not complete ;-(
  So i put the SGI-OpenGL-Include-Path to the first place :-)))

18. APRIL 1999 (Version 1.2.0 R1)

o Added Netscape 4.5 (Win32) support !
  Because Netscape JVM supports JNI, we do try
  to ask for a privilege :-) to run native DLL's !

  Please read the chapter <Source Installation>,
  <Binary Installation> in the given documentation, or the
  INSTALL.txt file for changes !

  This task is done for Eloi Maduell
  who asked for it :-) - thanxs !

o Docs supports now JAVA2

o boolean gljIsInit();
  is added - to query if the users init function is allready done !
  Now - we can call start and stop while doing animations
  directly from the applets class.
  Look in the demo directory at:

   glOlympicCvsApplet and glLogoCvsApplet

  Also gljIsInit is used to query we are able to render,
  better than just check if we just got the gl-context !

o Updated dummyGL.java and dummyGLCvs.java, also
  olympicCvs.java and glLogoCvs.java and
  glOlympicCvsApplet, glLogoCvsApplet !

  The above files are now JAVA2 and applet clean !

o Added Java2Applet.bat and Java2AppletB.bat with gl4java.policy
  to use JAVA2 appletviewer !
  (you have to change the gl4java.policy file) !

o make creates now a GL4Java.jar file - also :-)

o added Elois Maduell Texture Test JApplet which uses PPM.
  Moddified his one to use JAVA2 Swing, and Netscape !

19. APRIL 1999 (Version 1.2.0 R2)

o Netscape 4.5 (Win32) PrivilegeManager usage only if we are
  running the Netscape JVM !

o Java-PlugIn for Browser works.
  Added Java2 Plugin HTML-Pages in demos (demos/*J2P.html) !
  Tested with Win32,Netscape,Java2 !
  See INSTALL.txt for docu.

o Added static main function in GLCanvas and GLFrame,
  to just checking the library loading !

o supporting explicit library loading with static function
  loadNativeLibrary in GLCanvas and GLFrame

o debugged gljDispose: using topLevelWindow only if != null :-)

20. APRIL 1999 (Version 2.0.0 - Release 1)

o This new Major Release announces a new implementation !

     GL4Java V 2.X.X.X uses a new object model, which allows
     separation of GLEnum, GLUEnum, GLFunc*, GLUFunc* and GLContext !

  GLFunc*
   implements GLEnum

  GLUFunc*
   implements GLUEnum

  GLContext
```

```
   has_a    GLFunc*
   has_a    GLUFunc*

     For compatibility issues,
     i do provide the class gl4java.awt.GLCanvas,
     which:
   implements  GLEnum
   implements  GLUEnum
   has_a GLFuncJauJNI
   has_a GLUFuncJauJNI
   has_a   GLContex
   is_a Canvas

     The little changes the user must do in his sources are:
   change: GL4Java.GLCanvas   -> gl4java.awt.GLCanvas
   change: gljGetFpsCounter   -> cvsGetFpsCounter
   change: gljResetFpsCounter -> cvsResetFpsCounter
   change: gljGetWidth        -> cvsGetWidth
   change: gljGetHeight       -> cvsGetHeight
   change: gljDispose         -> cvsDispose
   change: gljUse()     -> glj.gljMakeCurrent(true)
   change: glj*     -> glj.glj*
   change: gl*     -> gl.gl*
   change: glu*     -> glu.glu*
change: GLCanvas.glClassDebug  -> GLContext.gljClassDebug
change: GLCanvas.glNativeDebug -> GLContext.gljNativeDebug
change: glClassDebug           -> GLContext.gljClassDebug
change: glNativeDebug          -> GLContext.gljNativeDebug


   The little shell script change2GL4JavaV2.sh
   will do this for you while using sed !

The function 'GLContext.loadNativeLibraries(null,null,null)'
is called in the static part of GLCanvasV2 !
If you do not use GLCanvasV2, call it by yourself !

If the given paramter 'libName's' is zero
by calling GLContext.loadNativeLibraries,
the default library-names are used - see below !

     Look in the new documentation and the demos,
     which explain the new model !

     So far ...

o Added the sun.awt.macintosh.MacHandleAccess.java class,
  and the appropriate Code to GLContext for
  fetching the WinHandle and the library !
  This is done to support Gerard Ziemski's Macinstosh-Port !

  The default native library for Win32 and Unice's is :
   GLContext: GL4JavaJauGljJNI
GLFuncJauJNI: GL4JavaJauGLJNI
```

```
GLUFuncJauJNI: GL4JavaJauGLUJNI

  The default native library for Macintosh is :
   GLContext: GL4JavaMacGZGljJNI
GLFuncJauJNI: GL4JavaMacGZGLJNI
GLUFuncJauJNI: GL4JavaMacGZGLUJNI

   These lib-names are used,
   if 'GLContext.loadNativeLibraries( String gljLibName,
   i       String glLibName,
          String gluLibName )'
   is called with an null argument !

o updated documentation
  now using java2's javadoc ;-)

o now, only a MS OpenGL binding is supported,
    because SGI does not support their library anymore :-( !

18. MAY 1999 (Version 2.0.1 - Release 1)

o Netscape 4.5 (Win32) PrivilegeManager usage was incorrect
  (used 30caps...), now using "UniversalLinkAccess" !

  This now works for me within Win95 and WinNT !

o Made some bugfixes within the source code include-file-names
  and the gl.solaris.not.* files and the makefile
  (thanxs Odell Reynolds)

o added support for the Win32 OpenGL-EXT functions,
  wich are now loaded via 'wglGetProcAddress' the very first time !

o added 'glColorTableEXT' support within GLFunc and its implementations.

01. JUNE 1999 (Version 2.0.1 - Release 2)

o Added files:
- VERSIONS.txt (describes the version numbers)
- THANXS.txt   (for all the helping hands ...)

  o The gl4java.jar package of the previous version included
  encryption information which where wrong.
  This bug diabled Netscape from using the JAR archive :( !
  This is now fixed :) !

o Tested GL4Java with jdk1.2-pre-v1 on Linux, it works !
  You must disable JIT and enable NATIVE THREADS !

o Removed native-libs other than
- libGL4Java-Linux-glibc-2.X-mesa3.0pthreads-x86
- libGL4Java-Win32-x86

        o Now having UNIXTYPE/WIN32TYPE defined
```

```
          instead of UNIXLIBDIR/WIN32LIBDIR in symbols.mak,
          where only the OS is specified - not the GL4Java version !
          Now - a tar-gz/zip file is generated for the specific
          OS-type - not a directory. The filename contains the
          GL4Java name+version and the OS type information !
  This is also done for the gl4java.jar file !

o No more GL4Java*bin archive exists !
  Now all needed files for a binary-installation
  (the native-libs and gl4java.jar) moved to the 'binpkg' directory !
  Have a more closer look at INSTALL.txt !

11. JUNE 1999 (Version 2.0.2 - Release 1)

o Fixed gl4java.awt.GLAnimCanvas !

  - Now the fps rate is adapted for every frame - all the time.
    No more stocking animations !!!

  - Set the default fps value to 20 :-) !

o The X11 native function gljSwap()
  does not call glXWaitX and glXWaitGL anymore (-> faster ?!) !
  I hope this is not needed - please tell if so !

o The gl.solaris.not.c file is repaired !

        o Now the makefile uses the zip-utility again.
  I registered, that the java-jar zip-files
  could not be read by WinZip :-( !

o all zip archives should be readable for win32/tools.

30. AUGUST 1999 (Version 2.1.0 - Release 1 )

o Added symbols.mak and gl.not.c for NVIDIA Mesa GLX Module

o X11: Fixed XVisualInfo query, while just setting a prefix now,
  instead of using the found XVisualInfo (RGBA, DOUBLEBUFFER) !
  This fixes the double buffering problem with
  some hardware accelerated OpenGL driver !

o Merged MSJVM port
- Ron Cemers port to the MS-JVM is merged.
  The following MS-JVM (build 3186) is needed:
  "Microsoft (R) VM for Java, 5.0 Release 5.0.0.3186"
  New classes for MS-JVM:
. gl4java.system.GljMSJDirect
. gl4java.GLFuncMSJDirect
. gl4java.GLUFuncMSJDirect
. sun.awt.windows.MSWin32HandleAccess
  New native library for MS-JVM:
   . GL4JavaGljMSJDirect
  New documentation for MS-JVM:
   . MS-JVM.txt
. README.RonCemer (updated original)

- The MS-Java-SDK >= 2.0 is needed
    (A SUN std. JDK is still needed)
- To give the port a better distinguish,
  I droped the JDirect stuff outta gl4java.GLContext
  to gl4java.system.GljMSJDirect
- gl4java.system.GljMSJDirect,
  gl4java.GLFuncMSJDirect,
  gl4java.GLUFuncMSJDirect and
  sun.awt.windows.MSWin32HandleAccess is compiled ONLY
  with the MS-Java-Compiler ;-) !
- gl4java.system.GljMSJDirect,
  gl4java.GLFuncMSJDirect,
  gl4java.GLUFuncMSJDirect do have the MS-Java state:
  "@security(checkDllCalls=off)", so if you have
  the classes and dll installed, you can run
  the applet within the MS-IE 4.0 :-) without cab's !
- The old glj* native methods and the new glj*JDirect
  native methods are improved:
- function arguments, instead of object retrieving
  (for the old ones, like Ron's -> faster)
- straight usage of "static" "final", if usabel !
- The method "doCleanup" and the paranoia implementation
  of "cvsDispose" is taken over from Ron
  within gl4java.awt.GLCanvas
- glj.gljResize() is now called within the paint method,
  instead of the reshape method, which is for
  users-overwrite purpose - so you cannot forget it.
  This is important - especially for the MS-JVM, to
  resize the own created native window !

o Prepared for merging Macintosh port
- I hope gl4java.GLContext and the other std. java-classes
  must not be changed ...

o Switched to MS-Visual-C++ 6.0 (Borland deleted)

o Moved glDemosCvs to an Applet ;-)
  Hmm - MS-IE is faster than Netscape (under Win32) with GL4Java ...

o Added visual properties in gl4java.GLContext:
- doubleBuffer
- stereoView
  These can be set with the constructor now
  and are updated after creating the OpenGL-Context,
  while quering with glXGetConfig/getPixelFormat../... !
  If one of these properties is invalid for the hardware,
  the native stuff in GL4Java has to "fall back" and set the
  java flags correct !
  These states can be queried with gl4java.GLContext !

o Added gljSetEnabled()/gljIsEnabled() to gl4java.GLContext !
```

```
o Added doubleBuffer, stereoView to
  gl4java.awt.GLCanvas as transparent params (visual-properties)
  to use for the gl4java.GLContext constructor !

o Added preInit() to gl4java.awt.GLCanvas to give the user a chance
  to modify the default visual-properties !

o Now using Ron's "capsapi_classes.zip" within compilation
  of GL4Java, to prevent the users to add some
  netscape or other java-api's to his classpath !
  Look at symbols.mak !

14. Oktober 1999 (Version 2.1.1 - Release 0 )

o Added to the demo-package:
- Ron's demos
- RectRenderSpeed

14. November 1999 (Version 2.1.2 - Release 1 )

o ReFactoring of the C2J Parser,
  which generates all the Java/C native glue !

- Complete re-work to select all information
  into objects while parsing,
  instead of just printing them out !

- Printing the objects (functions-declarations)
  in different formats:
   - JNI-Java Code
- JNI-C Code
- MSJDirect Code

- Handels "void *" in the argument-list in the way,
  that each functions will be generate SEVEN times
  with the following type-substitutions:
   "byte[]", "short[]", "int[]",
"float[]", "double[]", "boolean[]", "long[]"
  Only 1 "void *" in the arg-lidt can be handled now.
  This is enough for OpenGL ;-)

  The function overloading is done
  with the Java function signature,
  so a ANSI-C compiler is enough !

- Handels double Pointers -> [][] - and more ;-) !

- Handels "const" arguments correct:
- no more SetArray calls

- Gives semantic error messages,
  if something can not be handled !
  E.g.: More than 1 "void *" argument,
  or a pointer-type as the function-type !

o Because of the essential changes of C2J
  The following benefits for GL4Java exist:

- Only the glGetString, gluGetString gluErrorString
  functions must be written manualy now !!!!

- Better compatibility with "GLvoid *" arguments.
  This means - not just "byte[]" can be used -
   "byte[]", "short[]", "int[]",
"float[]", "double[]", "boolean[]", "long[]"
  functions are generated !

- Better MS-JDirect (MS-JVM) integeration !

o Complete OpenGL 1.2 support (if native OpenGL supports it) !
  So missing functions like:
void glEnableClientState( GLenum cap );
void glDisableClientState( GLenum cap );
  do exist now.
  This is because of the new C2J !

o Nearly Complete GLU 1.1 support - Except :
- gluScaleImage (Only byte[] i/o is supported)

- The following are not supported,
  because they do use function-pointers:
- gluQuadricCallback
- gluNurbsCallback
- gluTessCallback
  This is because of the new C2J !

o Changed all "native pointer" presentation in Java
  from "int" to "long" !
  I guess this provides a better compatibility to
  64 bit machines with 64 bits per adress !

o Win32: Added ALL EXT functions which exists in MSVC 6.0 header files
  to the gl4java_bin_ext function in gl.win32.ext.c !

18. November 1999 (Version 2.1.2 - Release 2 )

o Some errors moved into the DEMOS section !
  - GL_TRUE and GL_FALSE are now set to the new boolean ...

        o Ron Cemer's Demos are adapted and modified
  to achieve more compatibility. See demos/RonsDemos/index.html !

o A new Class "gl4java/applet/SimpleGLAnimApplet1.java"
  is made and used by all animation classes of Ron Cemer !
  This package (gl4java/applet) is now part of the
  distribution !
  Please update at least the java archive/classes
  (especially the Windows-Users, where the native libs
   has no changes ! )

o ReWork of the X11-Window-System part,
  which achieve the XVisual.
  This is another try, because the Irix-Systems
  have some troubles .... :-(

07. Dezember 1999 (Version 2.1.3 - Release 0 )

o Added Ron Cemer's Installer to this repository.
          This is a changed version - version 2.00,
  which based upon Ron's orig. version 1.05 !
  Please read the Installer/CHANGES.txt !

o Added geometric GLUT support: gl4java.utils.glut !
  See some demos in demos/MiscDemos !
  This is a lightweight implementation,
  meaning that the SGI sources are used !

o Added more Demos, e.g. morph3d !

o Added PNG-TextureLoader support in gl4java.utils.textures !
  This packages uses the LGPL png classes:
  Copyright (c) 1998,1999  Six-Legged Software
          Please read the PNG-*.txt files !

  The 1st code, which usese the PNG class,
  and the original scale-code is written by:
  Max Rheiner  <mrn@paus.ch> !

  Check the demo: demos/MiscDemos/pngTextureTestApplet.java !

14. Dezember 1999 (Version 2.1.3 - Release 1)

For MesaNvidia 3.1 (Linux-x86) ONLY !

o Changed the version for Mesa-NVidia to MesaNvidia 3.1 (Linux-x86)
- This is done in the Installer,
- and the CNative/gl.nvidia.not.c file !

o Added installation instructions for the png.jar archive !

24. January 2000 (Version 2.1.3 - Release 2 )

o This version is based upon Mesa 3.1,
  the older versions are based upon Mesa 3.0
  (For GL and GLU function creation (C2J), etc.)
  So we do support:
   OpenGL 1.2
GLU    1.2
  But be sure, that your underlying OpenGL native library
  does support OpenGL 1.2 and GLU 1.2 also !!!
o Now supports - in respect to Mesa 3.1:
- GL_EXT_stencil_wrap
- GL_NV_texgen_reflection
- GL_PGI_misc_hints
- GL_EXT_compiled_vertex_array
- GL_EXT_clip_volume_hint

  - The following extensions are skipped from Mesa 3.1:
- GL_INGR_blend_func_separate
- GL_ARB_multitexture

o C2J has changed to respect the new Mesa Version 3.1
  All "void *" Arguments of one function are mapped to all java
  array-types.

6. March 2000 (Version 2.2.0 - Release 0 )

o The pointer conversion to the "long" type did not worked for
  Netscape under linux !
  A "bus error" occures, while using "long" in the native part !
  So all pointer presentations are kept "int" !!

o All GLU Callback Functions (Tessellators, ...)
  are now supported while using the JNI reflections
  and seeking the fitting method for method-type
  and gl-context !
  Please have a look in Tesselation.txt !

o Added GLUT Font support !
  Thanxs to Pontus Lidman (Mathcore) for supporting
  GL4Java with GLUT font support !
  The special package:
gl4java.utils.glut.fonts
  provides the new GLUT class:
        GLUTFuncLightImplWithFonts
  which is derived from the GLUT class:
gl4java.utils.glut.GLUTFuncLightImpl
  This new package provides font support
  and contains the font data and the font-class
  code generation code.

  This new package gl4java.utils.glut.fonts
  is distributet in an own jar/zip file:
  gl4java2.2.0.0-glutfonts-classes.zip
  gl4java2.2.0.0-glutfonts-jar.zip
  This is done, because of its size - about 200kBytes !
  But this optional package is selected
  within the Installer as the default ;-) !

  Look at demos/MiscDemos/glutFont*Test.html
  for the new font demos !

o Added texture grabber class:
```

```
gl4java.utils.textures.TGATextureGrabber implements
gl4java.utils.textures.TextureGrabber
  this default impl. uses the TGA file type,
  to save the opengl framebuffer !
          Thanxs to Erwin 'KLR8' Vervaet for the TGA writer code !
  The demos: tessdemo, tesswind and morph3d
  include a snapshot menue, if started as an application !

o In class gl4java.awt.GLCanvas,
  the following attributes are made public:
    public GLFunc gl = null;
    public GLUFunc glu = null;

  The following attribute stays protected and can be fetched
  by the method: "public final GLContext getGLContext()"
    protected GLContext glj = null;

o The stencilBits configuration for stencil-bit number setup
  for X11 and Win32 system is implemented !

  The OpenGL Stencil feature works now !

  The usage of the stencil buffer does work now !
   Look at demos/MiscDemos/stencil.html
  for the new stencil demo !

  Changes in:
   Native: X11, Win32, Win32JDirect
TODO: Macintosh

Java: gl4java.GLContext, gl4java.awt.GLCanvas

o The new Flag GLContext::createOwnWindow
  (mapped in GLCanvas, like doubleBuffer, stencilBits, ...)
  controlls the behavior of the native window code.
  If createOwnWindow equals true,
  an new overlayed window is created.
  This is sometimes needed for some machines,
  like SGI's Irix or maybe IBM's AIX !
  This flag can be forced to be always true,
  by setting the compile definition:
    #define GL4JAVA_FORCE_X11_OWN_WINDOW
    ( gcc -D GL4JAVA_FORCE_X11_OWN_WINDOW ... )
  This should be always set for IRIX !
  At this time some refresh problems occure,
  if the window comes on top again.

  Changes in:
   Native: X11

Java: gl4java.GLContext, gl4java.awt.GLCanvas

o C2J bugfix 1.2
  - GLUtesselator is now supported !
  - All GLU methods are now implemented !
  - Added support for NULL/null pointer arrays,
    so you can call e.g.
     "glu.gluTessBeginPolygon(tobj, (double[])null);"
    The C-JNI part does handle it now !
```

7.0.3 License

```
If you like to contact the author use:

Sven Goethel, Elpke 5, 33605 Bielefeld, Germany
email: sgoethel@jausoft.com
web  : http://www.jausoft.com
voice: +49-521-2399440
fax  : +49-521-2399442

Sven Goethel, 15 January 1999
```

7.1 Package-Documentation (javadoc)

To look at the javadoc generated source documentation, go to packages.html.

Only the paperware produced for the FH-Bielefeld contains a printout of this pages as an extra appendix.

7.2 Important Links

At our homepage we generated some important links at ../../LINKS.html.

Chapter 8

Resume

8.1 Contact us

Like we described in the Foreword, we would be very thankfull for any response.

- gl4java@jausoft.com 1st email adress
- a_kolac@bielefeld.netsurf.de 2nd email adress
- Homepage: http://www.jausoft.com

8.1.1 Thanxs

```
I want to THANK :

        o Leo Chan for porting the OpenGL library to java the first time !

        o Adam King for porting Leo Changs version to integrate
          the OpenGL-Rendering into the Java-Frame !

        o Tommy Reilly for participating and much improvements

        o Atilla Kolac for participating in GL4Java while his and mine
  diploma thesis work.

o Leung Yau Wai for debugging Win32 !

o Eloi Maduell for keeping me working on the Netscape support
  and his texture examples.

o Lee Elson & Odell Reynolds
  for makeing the source code more compatible
  and creating binaries for other unices.

o Gerard Ziemski for porting GL4Java to the MACintosh !

o Ron Cemer for porting GL4Java to the MS-JVM
  and writing the INSTALLER !

o Pontus Lidman (Mathcore) for adding
```

```
  GLUT Font support for GL4Java !

o And many others, who wrote software for GL4Java !
```

Appendix A

OpenGL Einf"uhrung‡

A.1 Was ist OpenGL?

OpenGL [2] [3] wurde von Silicon Graphics,Inc.(SGI) entwickelt und ist eine verbreitete Hardware- und Betriebsunabh"angige Bibliothek f"ur die Erstellung von dreidimensionalen Grafiken. GL steht hierbei f"ur Grafics Library.

Au"serdem stellt OpenGL eine unabh"angige Programmierschnittstelle dar. Wobei f"ur die Hardwareseite eine geeignete Implementation programmiert werden kann. Die Softwareschnittstelle besteht aus ca. 120 Kommandos, die zur Spezifikation von Objekten und Operationen ben"otigt werden. OpenGL ist client- und serverf"ahig. Das Protokoll welches OpenGL-Befehle "ubermittelt, ist identisch. Dadurch k"onnen OpenGL-Programme im Netzwerk ablaufen, in denen client und server Rechner verschiedenen Typs sind. Dies erreicht man indem die Kommandos zum Ablauf von Windows-Tasks oder zur Interaktion mit dem Benutzer nicht in dem OpenGL-Programm implementiert werden. Sondern diese Aufgaben werden direkt mit der Fensteroberfl"achenprogrammierung der entsprechenden Hardware mit"ubernommen.

Der Sinn der OpenGL-Bibliothek ist es, zwei- und dreidimensionale Objekte in einem Frame Buffer zu *rendern.* Dieser ist mit einem Pixelspeicher in der Grafik-Hardware eines PC vergleichbar. OpenGL ist streng prozedural. Das hei"st, da"s nicht das Aussehen eines Objektes beschrieben wird sondern wie das Objekt gezeichnet wird. Hierzu bedient sich OpenGL zwei - oder dreidimensionalen *Vertices.* Diese repr"asentieren einen Punkt, z.B. den Endpunkt einer Linie oder eines Polygons. Die n"achste Ebene sind *Primitive*, die aus einem oder mehreren *Vertices* bestehen.

Wie Vertices zu Primitiven zusammengesetzt werden und in ein Frame Buffer gezeichnet werden, wird durch eine Vielzahl von Einstellm"oglichkeiten kontrolliert.

Mit OpenGL kann man weiterhin Oberfl"achen zeichnen, Beleuchtungsspezifikationen anwenden und Texturen verwenden.

Die wichtigsten Schritte bis zum Rendern einer Szene

1. Aus den geometrischen Primitiven werden Formen konstruiert und damit mathematische Beschreibungen der Objekte erstellt. OpenGL sieht Punkte, Linien, Polygone, Bilder und Bitmaps als Primitive an.

2. Die Objekte werden im dreidimensionalen Raum arrangiert und der gew"unschte *Blickpunkt* "uber der Szene wird ausgesucht.

3. Die Farben aller Objekte werden berechnet. Zwar sind die Farben f"ur die Objekte im Programm definiert, m"ussen jedoch neu berechnet werden falls sich das Objekt bewegt oder die Schattierung sich "andert.

4. Die mathematische Beschreibung von Objekten und den ihnen beigef"ugten Farbinformationen werden in Bildschiermpixel konvertiert. Dieser Proze"s wird *Rasterung* genannt.

Alle Daten, ob sie nun Geometrien oder Pixel beschreiben, k"onnen in einer *Display List* gesichert werden. OpenGL unterscheidet zwischen zwei Berechnungsformen: Die Berechnung der Grafik wird sofort ausgef"uhrt (*immediate-mode*). Das bedeutet wenn das Kommando zum Zeichnen eines Objektes abgesetzt wird, wird das Objekt gezeichnet. Dieser Modus ist in OpenGL voreingestellt. Weiterhin lassen sich Kommandos in einer *Display List* sichern, um sie sp"ater auszuf"uhren. Die sofortige Ausf"uhrung der Grafik l"asst sich einfacher programmieren, das Sichern in einer Liste ist effizienter.

A.2 Grundlagen

A.2.1 Client- Serverprotokoll

A.2.2 Datentypen

A.2.3 Beleuchtung

OpenGL berechnet die Farben jedes Pixels, bevor die Szene dargestellt wird und speichert sie im Frame Buffer ab. Dabei wird das in der Szene benutzte Licht berechnet, und die Art, wie die Objekte in der Szene das Licht reflektieren oder absorbieren sollen. Ein unbeleuchtetes dreidimensionales Objekt kann nicht von einem zweidimensionalen Objekt unterschieden werden. Denn auf einer zweidimensionalen Fl"ache, sprich unser Bildschirm, wird die dreidimensionalit"at durch die Art der Beleuchtung und der Schattierung erreicht. Das zeigt, wie wesentlich die Beziehung zwischen Objekten und dem Licht in einer dreidimensionalen Szene ist. OpenGL stellt Licht und Schatten durch Zerlegung der Objektoberfl"ach in bestimmte rote, gr"une und blaue Komponenten dar. Der Ort der Lichtquelle, sein Einstrahlungswinkel und Art m"ussen programmiert werden.

Das Beleuchtungsmodell in OpenGL besteht aus vier Komponenten: *emitted, ambient, diffuse und specular.* Alle vier Komponenten bewirken eine unterschiedliche Beleuchtungsart. Sie werden getrennt angegeben und danach zusammenaddiert, so das die Szene in das gew"unschte Licht gesetzt wird.

A.3 Die Programmierung

A.3.1 Namenskonvention

A.3.2 Die Kommandosyntax

OpenGL-Kommandos beginnen in der Programmiersprache C mit dem Pr"afix *gl*, jedes Wort beginnt mit einem Gro"sbuchstaben (z.B. glClearColor()). Konstanten beginnen mit dem gleichen Pr"afix, allerdings werden sie in Gro"sbuchstaben geschrieben und durch Unterstriche verbunden (z.B. GL_CLEAR_BUFFER_BIT). in dem Kommando *glColor3f()* steht 3 f"ur die Anzahl der gegebenen Argumente, f steht f"ur die Gleitkommazahlen.

Die ANSI-C-Implementation von OpenGL erm"oglicht bis zu 8 verschiedene Datentypen. Die Datentypen umfassen 8, 16 und 32-Bit-Integer, im signed- und unsigned- Format und 32- oder 64-Bit-Gleitkommazahlen. Einige OpenGL-Kommandos sind mit dem Endbuchstaben v versehen, was darauf hindeutet, da"s das Kommando einen Zeiger auf einen Vektor oder Feld beinhaltet.

A.3.3 Die Bibliotheken

OpenGL bietet eine Anzahl leistungsf"ahiger und einfacher Kommandos zum Rendern. Alle komplexen Zeichnungen m"ussen mit diesen Kommandos ausgef"uhrt werden. Deshalb ist es m"oglich, eine eigene Bibliothek zu schreiben, die auf OpenGL aufsetzt. Damit l"a"st sich z.B. die Behandlung der Fenster vereinfachen.

Hier sind zwei sehr verbreite Bibliotheken aufgelistet:

- Die *OpenGL Utility Library (GLU)* enth"alt einige Routinen die einfachere OpenGL Kommandos benutzt. Es werden Funktionen zur Vereinbarung von Matrizen f"ur besondere Betrachtungsorientierungen und Projektionen und zum Rendern von Oberfl"achen definiert. GLU-Funktionen sind durch den Pr"afix *glu* zu erkennen.

- Die OpenGL-Erweiterung f"ur das X-Windows System (GLX) sieht die Erzeugung eines OpenGL-Kontextes vor und die Hinzuf"ugung eines Fensters, in dem gezeichnet werden kann (drawable window), falls auf einem Rechner mit X-Window System gearbeitet wird. GLX-Funktionen besitzen das Pr"afix *glx*.

Hinzukommen noch eine gro"se Anzahl weiterer Bibliotheken, die wir hier verst"andlicherweise nicht aufz"ahlen k"onnen.

A.3.4 Die OpenGL Windows-Programmierung

Bevor die OpenGL-Bibliothek unter MS-Windows verwendet werden kann, m"ussen eine Reihe von Initialisierungsschritte ausgef"uhrt werden. Jede Windows-OpenGL-Applikation mu"s seinen *Rendering Context* (OpenGL-Seite) mit seinem *Device Context* (Windows-Seite) verbinden. Hierzu mu"s zuerst die Win32-Funktion *SetPixelFormat* und danach die Funktion *wglCreateContext* mit dem Device Context Handle Parameter aufgerufen werden. Ist dies erfolgreich, gibt wglCreateContext einen Rendering Context Handle des Typs *HGLRC* zur"uck. OpenGL unter Windows kennt zwei Typen von Pixel-Modi: Einen Modus, der die direkte Angabe von Pixelfarben erlaubt. Und ein Modus, der die Auswahl der Farbe aus einer Palette unterst"utzt.

Es gibt spezielle Anforderungen durch OpenGL an ein Ausgabefenster. Es mu"s mit den Fensterstilen *WS_CLIPSIBLINGS* und *WS_CLIPCHILDREN* initialisiert sein um OpenGL Kompabilit"at zu gew"ahrleisten. Um die Perfomance der Applikation zu erh"ohen, sollte das Fenster einen NULL-Hintergrund haben, da dieser sowieso durch die OpenGL-Bibliothek gel"oscht wird. Bevor ein Rendering Context verwendet werden kann, mu"s er mit der wglMakeCurrent Funktion als der aktuelle Context bestimmt werden. Ist der Rendering Context bereit um Kommandos entgegenzunehmen k"onnen weitere Initialisierung-Routinen aufgerufen werden. Z.B. um den Frame Buffer zu l"oschen, Koordinatentransformationen aufzusetzen, Lichtquellen zu konfigurieren oder andere Optionen zu setzen.

Ein solcher Initialisierungsschritt, der nicht ausgelassen werden darf, ist der Aufruf der *glViewport*-Funktion. Sie initialisiert oder modifiziert die gr"o"se des *Rendering Viewports.* Typischerweise wird diese Funktion ganz zu Anfang der Applikation aufgerufen und dann jedesmal wenn sie eine *WM_SIZE*-Meldung erh"alt. Diese zeigt eine Gr"o"sen"anderung des Fensters an.

Appendix B

Java Einf"uhrung‡

Weitere englischsprachige Dokumentationen sind von Sun Microsystems erh"altlich[5].

B.1 Was ist Java?

Sun Microsystems stellte Mitte 1995 sein Java Language Enviroment (kurz Java) offiziell vor.

Java ist eine objektorientierte Programmiersprache und ist Plattformunabh"angig.

Java-Programme k"onnen entweder lokal auf einem Rechner ausgef"uhrt werden, oder aber auch "uber das Internet gestartet werden.

B.2 Grundlagen

B.2.1 Die Entwicklungsumgebung

Sun Microsystems stellt f"ur die Plattformen Solaris(Sparc), Solaris(Intel) und Win32(Intel)[a] die Java-Entwicklungsumgebung *Java Development Kit (JDK)* kostenlos zur Verf"ugung. IBM stellt ihrerseits f"ur Windows 3.1(Intel), AIX(RS6000) und OS/2(Intel) die JDK zur Verf"ugung.

Die JDK enth"alt den Java-Compiler, die JVM und die standard Bibliothek, sowie den Java-Debugger. Der Compiler als auch der Debugger sind Kommandozeilentools und enthalten keine GUI.

Mittlerweilen ist auch das *Java Runtime Environment (JRE)* ver"offentlicht, welches lediglich die JVM und die standard Bibliothek enth"alt. Eigene Java-Applikationen k"onnen mit dem JRE geb"undelt und herausgegeben werden.

Der Java-Compiler generiert aus dem Java-Sourcecode den Java-Bytecode, wobei dieser Java-Bytecode von der Java-Virtuellen-Maschiene (JVM) interpretiert wird.

Der Java-Sourcecode enth"alt die Klassendefinitionen. In der Regel gibt es je Klassendefinition eine Quelldatei. Der Dateiname der Quelldatei setzt sich zusammen aus dem Klassennamen und der Dateinamensendung *.java*. Z.B. ist die Klassendefinition fuer *Foo* in der Datei *Foo.java* enthalten. Gross- und Kleinschreibung ist hierbei zu beachten.

Der Java-Compiler "ubersetzt die Quelldatei mit der Namensendung *.java* in die Zieldatei f"ur den Java-Bytecode mit der Namensendung *.class*. Z.B. erzeugt der Aufruf von

```
javac Foo.java
```

[a]Windows 95 und Windows NT

die Datei *Foo.class*.

Das compilierte Java-Programm, der Java-Bytecode, kann z.B. mittels

```
java Foo
```

aufgerufen werden. Dieser Aufruf startet die JVM und beginnt das Programm *Foo* abzuarbeiten.

B.2.2 Java-Virtuelle-Maschine (JVM)

Die JVM simuliert eine komplette Betriebssystemumgebung und ist in der Lage den Java-Bytecode abzuarbeiten. Hiermit sind Java-Programme Plattform unabh"angig, da die JVM auf verschiedenen Plattformen ein Einheitliches Java-System zur Verf"ugung stellen.

Mittels der JVM und den standard Java-Bibliotheken kann ein Java-Programm

- auf das Netzwerk/Internet zugreifen
- eine grafische Oberfl"ache besitzen (AWT)
- auf native Funktionen zugreifen (JNI)
- auf Datenbanken zugreifen (jdbc)
- wiederverwendbare Komponente besitzen (Beans)
- gemeinsame standardisierte Objekte mit Java und nicht-Java Applikationen austauschen (Corba/RMI)
- etc.

B.2.3 Java-Interpreter und JIT-Compiler

Der Java-Interpreter ist bestandteil der JVM. Der Java-Bytecode wird von dem Java-Interpreter abgearbeitet. Der JIT-Compiler (Just-In-Time-Compiler) ist eine Erweiterung des Java-Interpreters und "ubersetzt den Java-Bytecode blockweise in plattformspezifischen Machienencode.

B.3 Java-Applets und Java-Applikationen

Java-Programme lassen sich grob in zwei Kategorien teilen: Java-Applikationen bezeichnen eigenst"andige Programme, welche innerhalb des ausf"uhrenden Rechners (Client) gestartet werden. Der Client fungiert hier gleichzeitig als Server. Java-Applikationen haben die M"oglichkeit auf das Dateisystem des Clients, bzw. Server zuzugreifen. Z.B. kann die Java-Applikation Foo mittels

```
java Foo
```

aufgerufen werden. Hierf"ur werden lediglich die JVM und die entsprechenden *.class*-Files ben"otigt.

Ein Java-Applet wird auf einem Server abgelegt. Das Applet wird auf dem Client "ubertragen und dort lokal ausgef"uhrt. Im Gegensatz zur Java-Applikation kann das Applet nicht auf Daten ausserhalb der JVM zugreifen. Somit kann ein Applet die Sicherheit des Systems nicht gef"ahrden. Java-Applets werden innerhalb einer HTML[b]-Seite aufgerufen.

[b]Hypertext Markup Language, Textformat f"ur World-Wide-Web(WWW) Seiten

Das Applet wird mittels des Internet, bzw. Intranet, auf den Client "ubetragen. Hierf"ur kann ein Java-F"ahiger-WWW-Browser oder der *appletviewer* des JDK benutzt werden. Z.B. kann das Java-Applet Foo eingebettet in der HTML-Seite *Foo.html* mittels

```
appletviewer http://www.server.de/Foo.html
```

aufgerufen werden. Mittlerweile haben die meisten Unternehmen Java lizensiert und in ihren Web-Browsern unterst"utzt. Dadurch lassen sich Web-Seiten interaktiver gestalten.

B.4 Objekt-Orientierte-Sprache

B.4.1 Was ist Objektorientierung?

Java ist eine reine Objekt Orientierte Programmiersprache. Im Gegensatz zu einer prozeduralen Programmiersprache, die sich durch einen linearen Programmflu"s und einer klaren Trennung von Daten und Funktionen auszeichnet verkn"upft das objektorientierte Programmieren Daten und Methoden[c] zu einem Objekt. Objekte eines Typs lassen sich zu einer Objektklasse zusammenfassen. Klassen sind Beschreibungen eines Datenmodells (Prototypen) von denen sich beliebig viele Objekte generieren lassen. In den Klassen werden sowohl die Daten als auch die damit verbundenen Operationen festgelegt.

B.4.2 Klassen und Methoden

In Java wird eine Klasse mittels des Schl"usselwortes *class* (Analog zu Klassen in C++) aufgerufen.

Der Zugriff auf Daten und Methoden k"onnen innerhalb einer Klasse f"ur andere Klassen mittels *public*-deklaration erlaubt, bzw. mit *private*- oder *protected*-deklaration verboten werden.

Z.B. definiert die Klasse Feuerzeug Daten und Methoden f"ur Objekte des selben Typs.

```
public class Feuerzeug
{
/* Gasinhalt in Liter */
private float gasInhalt;

        /* 1 milliliter pro Zuendung */
private float finalize gasProZuendung = 0.001

/* Oeffentliche Methode fuer das Zuenden des Feuerzeugs.
           Beachte: Das Zuenden kostet Gas !
*/
public boolean zuende()
{
if(gasInhalt<gasProZuendung)
return false;
gasInhalt-=gasProZuendung;
return true;
}
}
```

[c]Methoden ist der objektorientierte Name f"ur Funktionen

Eine Methode ist unmittelbar an ein Objekt bzw. die dazugeh"origen Daten gebunden. Methoden k"onnen in Java auch "uberladen werden. Z.B.

```
public class Feuerzeug
{
...

// Feuerzeug ganz aufladen
void aufLaden()
{
...
}

// Feuerzeug mit definierter Menge 'vol' aufladen
void aufLaden( float vol )
{
...
}

}
```

Durch Methoden werden bestimmte Funktionen eines Objektes beschrieben. Sie werden in Java mittels des . Operators aufgerufen.

Objekt.Methode (....)

Konstruktoren sind spezielle Methoden, die bei der Aktivierung von Objekten aufgerufen werden und deren Instanzierung und Initialisierung "ubernehmen. Konstruktoren benutzen den gleichen Namen wie die verwendete Klasse.

Klasse Objekt=new KLasse (...)

erzeugt aus der Klasse ein Objekt, daf"ur reserviert es Speicher f"ur dieses Objekt und initialisiert es mit den entsprechenden Werten (evt. als Argumente "ubergeben).

B.4.3 Vererbung

Vererbung bezeichnet den Mechanismus des Ableitens einer Klasse aus einer "übergeordneten Klasse (Einfachvererbung). Die Daten und Methoden der "ubergeordneten Klasse, Oberklasse oder Superklasse genannt, werden an die Subklasse weitergegeben (vererbt). Die vererbten Methoden k"onnen dort auch "uberschrieben, bzw. "uberladen werden.

Die Klassen der standard Bibliothek sind in der Regel alle von der superklasse *Object* abgeleitet. Die Klasse *Object* definiert einige elementare Methoden (equals, string, getclass), welche von allen Subklassen benutzt, bzw. neu definiert werden k"onnen.

Im Gegensatz zu C++ k"onnen in Java keine Operatoren "uberladen, bzw. neu definiert werden. Als L"osung werden in Java standard Methoden verwendet, wie z.B. die Methode *equals* der Superklasse *Object*.

B.4.4 Packages

Ein wesentlicher Vorteil der objektorientierten Programmierung ist die einfache Wiederverwendung bestehenden Programmcodes. Klassenbibliotheken bieten Programmierern die m"oglichkeit bestehende Klassen in Ihrer Applikation einzubinden. Es existieren eine Reihe von standard Klassenbibliotheken (Packages) f"ur unterschiedliche Zwecke wie z.B. Grafik, Netzwerkaktivit"aten und Ein-und Ausgabefunktionen. Packages werden mit dem *import*-Befehl in den Programmcode eingebunden. Die Zusammenfassung von mehreren Klassen in Packages erleichtert die Realisierung und Modularisierung gro"ser Systeme. Packages dienen ebenfalls der einfachen Wiederverwendung.

B.5 System-Managment

B.5.1 Garbage-Collector

Ein in Java integrierter Garbage-Collector sorgt f"ur die automatische Freigabe nicht mehr ben"otigter Speicherbereiche. Dadurch werden potentielle Fehler vermieden, wie sie z.B. bei doppelter Speicherfreigabe in ANSI-C vorkommen k"onnen. W"ahrend der Laufzeit eines Java-Programmes arbeitet im Hintergrund der Garbage Collector als Thread niedriger Priorit"at.

B.5.2 Exceptions

Mit den Exceptions besitzt Java einen Mechanismus zur strukturierten Behandlung von Fehlern, die w"ahrend der Programmausf"uhrung auftreten. Tritt ein Laufzeitfehler (z.B. ein Array-Zugriff au"serhalb der Array-Grenzen), so wird ein eine entsprechende Exception generiert (*throw*). Diese Exception kann, bzw. mu"s innerhalb eines *try-catch*-Blocks abgefangen werden (*catch*).

Das Grundprinzip des Exception-Mechanismus in Java kann wie folgt beschrieben werden: Wird eine Exception ausgel"ost, arbeitet die JVM den Stack soweit zur"uck, bis sie eine entsprechende *catch*-Anweisung findet. Wird keine geeignete gefunden, gibt die JVM eine Fehlermeldung aus.

Eine Exception kann an jeder Stelle der Aufrufkette (Stack) abgefangen werden. Methoden, welche in der Deklaration angeben eine Exception ausl"osen zu k"onnen, m"ussen innerhalb eines *try-catch* Blocks aufgerufen werden.

Das Behandeln von Exceptions mit der try-catch-Anweisung sieht etwa folgenderma"sen aus :

```
try {
    Anweisung;

    ...
    }
catch ( Exceptiontyp1 x ) {
    Anweisung;

   ...
   }
catch ( Exceptiontyp2 x ) {
    Anweisung;

   ...
   }
```

B.6 Grafisches User Interface (GUI)

B.6.1 Was ist eine GUI?

AWT Swing ..

B.6.2 Abstract Window Toolkit (AWT)

B.6.3 Events

Benutzer-Interaktion !

Bei der Programmierung unter einer grafischen Oberfl"ache erfolgt die Kommunikation zwischen Betriebssystem und Anwendungsprogramm durch den Austausch von Nachrichten. Die Anwendung wird dabei "uber alle Arten von Ereignissen und Zustands"anderungen vom Betriebssystem informiert. Dazu z"ahlen z.B. Mausklicks, Bewegung des Mauszeigers, oder Ver"anderungen der Fenstergr"o"se oder dessen Lage. Jedes dieser Ereignisse oder *Events* l"ost eine Nachricht aus, die an das aktive Fenster gesendet wird. Dadurch wird dort der Aufruf einer *Callback*-Methode ausgel"ost. Das Programm kann auf ein *Event* reagieren, indem es die zugeh"orige Methode "uberlagert und mit der gew"unschten Funktionalit"at ausstattet. Die unter Java m"oglichen Events lassen sich grob in folgende Klassen unterteilen :

- Maus-Events
- Tastatur-Events
- Fenster-Events
- Action-Events
- Komponenten-Events

Z.B. wird ein Mausklick mit den Methoden *mouseDown* und *mouseUp* der Klasse *Component* behendelt. Wenn die Maustaste gedr"uckt wird, ruft das AWT die Methode *mouseDown* auf, l"asst man sie los wird die Methode *mouseUp* aufgerufen.

public boolean mouseDown(Event e, int x, int y); public boolean mouseUp(Event e. int x, int y);

Der erste Parameter beider Methoden ist eine Instanz Klasse *Event.* Die beiden anderen Parameter geben die Position in der Client-Area an, an der die Maustaste gedr"uckt bzw. losgelassen wurde. Ein Objekt der Klasse *Event* repr"asentiert das Ereignis, durch das die Nachricht ausgel"ost wurde. Ein *Event*-Objekt besitzt eine Reihe "offentlicher Instanzmerkmale wie z.B.:

Element	Bedeutung
public int x;	x-Koordinate des Mauszeigers bei Tastatur- und Mausereignissen
public int y;	y-Koordinate des Mauszeigers bei Tastatur- und Mausereignissen
public long when;	Zeitpunkt des Ereignisses
public int key;	ASCII-Wert der gedrckten Taste bei Tastaturereignissen bzw. Wert der Funktionstaste
public int modifiers;	Eine Kombination der Konstanten *Event.ALT_MASK, Event.CTRL_MASK, Event.META_MASK, Event.SHIFT_MASK*

B.7 Multithreading

B.7.1 Was ist Multithreading?

Durch die Weiterentwicklungen im Bereich der Betriebssystemtechnologie wurde das Konzept der *Threads* in den letzten Jahren immer popul"arer. Und durch bereitstellung der Basis von Library-Routinen auch konventionellen Programmiersprachen zur Verf"ugung gestellt. Java hat *Threads* direkt in die Sprache integriert und mit den erforderlichen Hilfsmitteln als Konstrukt zur Nebenl"aufigkeit implementiert. Ein *Thread* ist ein eigenst"andiges Programmierfragment, das parallel zu anderen *Threads* laufen kann. Der Laufzeit-Overhead zur Erzeugung und Verwaltung ist deutlich geringer als bei gew"ohnlichen Prozessen und kann in den meisten Programmen vernachl"assigt werden. *Threads* sollen unter anderem die Implementierung grafischer Anwendungen erleichtern, die durch Simulation komplexer Abl"aufe oft nebenl"aufig sind. *Threads* k"onnen auch dazu verwendet werden, die Bedienbarkeit von Dialoganwendungen zu verbessern, indem rechenintensive Anwendungen im Hintergrund ablaufen.

B.7.2 Programmierung von Threads

Threads werden in Java durch die Klasse *Thread* und das Interface *Runnable* implementiert. in beiden F"allen wird der *Thread-Body*, also der parallel auszuf"uhrende Code, in Form der "uberlagerten Methode *run* zur Verf"ugung gestellt. Die Kommunikation kann dann durch Zugriff auf die Instanz- oder Klassenvariablen oder durch Aufruf beliebiger Methoden, die innerhalb von *run* sichtbar sind, erfolgen. Zur Synchronisation stellt Java das aus der Betriebssystemtheorie bekannte Konzept des *Monitors* zur Verf"ugung. Dises erlaubtes, kritische Abschnitte innerhalb korrekt geklammerter Programmfragmente und Methoden zu kapseln. Somit wird der Zugriff auf gemeinsam benutzte Datenstrukturen koordiniert. Dar"uber hinaus stellt Java Funktionen zur Verwaltung von *Threads.* Diese erlauben es, *Threads* in Gruppen zusammenzufassen, zu priorisieren und Informationen "uber Eigenschaften von *Threads* zu gewinnen. Das *Schedulling* kann dabei wahlweise unterbrechend oder nichtunterbrechend implementiert sein. Die Sprachspezifikation legt dies nicht fest, aber in den meisten Java-Implementierungen wird dies von den M"oglichkeiten des darunter liegenden Betriebssystems abh"angen.

Die Klasse *Thread* ist Bestandteil des Packages *java.lang* und steht damit allen Anwendungen standartm"a"sig zur Verf"ugung. *Thread* stellt die Basismethoden zur Erzeugung, Kontrolle und zum Beenden von *Threads* zur Verf"ugung. Um ein konkreten *Thread* zu erzeugen, mu"s eine eigene Klasse aus *Thread* abgeleitet und die Methode *run* "uberlagert werden. Mit Hilfe des Aufrufs der Methode *start* wird der *Thread* gestartet und die weitere Ausf"uhrung an die Methode *run* "ubertragen. *start*wird nach dem starten des *Threads* beendet, und der Aufrufer kann paralell zum neu erzeugten *Thread* fortfahren. Die "ubliche Vorgehensweise, einen *Thread* zu beenden, besteht darin, die Methode *stop* der Klasse *Thread* aufzurufen. In diesem Fall wird der *Thread* angehalten und aus der Liste der aktiven *Threads*entfernt. Mit Hilfe der Methoden *suspend* und *resume* ist es m"oglich, einen *Thread* vor"ubergehend zu unterbrechen. Durch *suspend* wird die Ausf"uhrug unterbrochen, und durch *resume* wird der *Thread* (genauer gesagt: die Methode *run*)an der Stelle fortgesetzt, an der die Unterbrechung erfolgte.

B.8 Java Native Interface (JNI)

B.8.1 Was ist JNI?

Java stellt mit den sogenannten *nativen* Methoden die M"oglichkeit, nicht in Java geschriebene Programmes oder Abl"aufe in die Laufzeitbibliothek mit einzubinden. Gr"unde fr die Implementierung von *nativen* Methoden sind folgende:

- das man spezielle F"ahigkeiten der einzelnen Rechner oder der Betriebssysteme nutzen kann; die die Java-Bibliothek nicht bereitstellt. Dazu geh"ort z.B. der Anschlu"s an neue Peripherieger"ate oder Steckkarten, der Zugriff auf verschiedene Netztypen oder die Verwendung eines eindeutigen Merkmals des vorhandenen Betriebssystems. Solche F"ahigkeiten werden derzeit von der Java-Umgebung nicht bereitgestellt und m"ussen daher *au"serhalb* von Java in einer anderen Sprache (meist C oder eine mit C vertr"aglichen Sprache) implementiert werden.

- Ein weiterer Grund ist die Performance des auszuf"uhrenden Programms. Falls man den von Java zur Verf"ugung gestellten JIT-Compiler f"ur die Erh"ohung der Geschwindigkeit nicht einsetzen will, kann man f"ur den Geschwindigkeitsorientierten Teil (z.B. kritische innere Schleifen), in C geschriebene *native* Methoden einsetzen. Die Java-Klassenbibliothek nutzt diese M"oglichkeit bei bestimmten kritischen Systemklassen selbst, um die Effiziens des Systems zu steigern.
- Jedoch ist wohl der Hauptgrund f"ur den Einsatz von nativen Methoden in Java-Klassen der; da"s man auf diese Art und Weise schon existierende Programme ohne sie erst umzuschreiben, relativ einfach in Java einbinden kann. Diese M"oglichkeit haben wir in der Diplomarbeit auch genutzt. Die 3D-OpenGL-Animationen sind in C geschrieben und wurden sozusagen in Java eingebettet, doch dazu sp"ater mehr.

B.8.2 Grundlagen der JNI

Dynamische Bibliothek Unix: lib¡Name¿.so, LD_LIBRARY_PATH WIN: ¡Name¿.dll, SYSTEMVERZEICHNIS

B.8.3 Vorgehensweise

Um ein Interface zwischen Java und dem C-Code zu erstellen, sind im wesenlichen f"unf Schritte erforderlich :

1. Erstellen der Java-JNI Funktionsdeklaration
2. Generierung der C-Header Datei
3. Erstellen der C-JNI Functionen
4. Laden der JNI-Bibliothek in Java
5. Linken der Dynamischen Bibliothek

B.8.4 Erstellen der Java-JNI Funktionsdeklaration

Den Java-Code mit der deklarierten *nativen* Methode zu schreiben und zu "Ubersetzen.

Um *native* Methoden in Java-Klassen zu benutzen, mu"s man in der Deklaration der Methode lediglich das schl"usselwort *native* einf"ugen.

wie z.B.

```
public nativ void Methodenname();
```

Und die Methode System.loadLibrary() in die Klasse mit einbinden. Das Schl"usselwort *native* zeigt dem javac-Compiler und dem java-Interpreter das sie nach einem Methodenrumpf in einer dynamisch-ladbaren Bibliothek Ausschau halten m"ussen. Um die gew"unschte Bibliothek letzendlich wirklich aufzurufen, wird sie mit der Methode *System.loadLibrary()* aus dem System-Package von Java eingelesen. Java durchsucht alle Pfade die in dem Betriebssystem-Umgebungsvariable angegeben wurden.

Alternativ steht die Methode *Runtime.getRuntime().loadLibrary()* und die Methode *System.load()* zur Verf"ugung. Letztrige liest eine Klassen-Bibliothek anhand ihres vollst"andigen Pfadnamens ein. So k"onnen auch Bibliotheken au"serhalb des Suchpfades benutzt werden. Die "Ubersetzung des Java-Codes kann wie "ublich ausgef"uhrt werden.

B.8.5 Generierung der C-Header Datei

Der n"achste Schritt ist es, die "ubersetzte Datei *Datei.class* dazu zu benutzen, um die entsprechende Header-Datei *Datei.h* zu erzeugen. Dazu wird das *javah*-Tool aus der JDK-Distribution eingesetzt. Per Default wird die neue h-Datei im gleichen Verzeichnis angelegt. Mit der Option -d kann ein anderes Verzeichnis angegeben werden. Dies ist zu raten da noch diverse andere Dateien erzeugt werden m"ussen und schnell die "Ubersicht verloren gehen kann. Der Aufruf

```
javah -jni Datei
```

erzeugt das C-Headerfile *Datei.h.*

In der Header-Datei wird durch die typedef-struct-Anweisung der *nativen* C-Routine mitgeteilt, wie die Daten in der Java-Klasse angeordnet sind. Die einzelnen Variablen in der Struktur k"onnen benutzt werden, um die Klassen und Instanz-Variablen von Java zu benutzen. Weiterhin wird in dem h.File ein Prototype angegeben, um die Methode aus dem objektorientierten Namensraum der Java Klasse in den C-Namensraum zu "uberf"uhren. Der Name einer Funktion, der eine *native* Methode implementiert, ergibt sich dabei immer aus dem Packet-Namen, dem Klassen-Namen und dem Namen der nativen Methode von Java, getrennt durch einen Unterstrich.

B.8.6 Erstellen der C-JNI Functionen

Nun m"ussen die nativen Methoden implementiert werden. "Ublicherweise werden die Implementationsdateien mit dem Klassennamen und einer eindeutigen Endung z.B. *Datei.c* versehen. Die Implementationsdatei mu"s die Datei *jni.h* mittels #include-Anweisung einbinden. *jni.h* ist im *include* Verzeichniss des JDK enthalten. Ebenfalls muss die mittels *javah -jni* erzeugte Headerdatei eingebunden werden. Der Funktionskopf in der Implementationsdatei muss den generierten Prototypen aus der Headerdatei entsprechen.

Diese Fehlerart macht sich noch nicht beim Linken sondern erst zur Laufzeit des Programms bemerkbar und ist daher m"uhsam zu beheben.

B.8.7 Erstellung der Dynamischen Bibliothek

Nachdem nun alle ben"otigten Dateien vorhanden sind, mu"s die Implemtierungsdatei nur noch "ubersetzt und mit der Java-Bibliothek zu einer dynamischen Bibliothek zusammen gelinkt werden.

B.8.8 Datenaustausch zwischen Java und C

Jedem elementaren Typ in Java wird ein Typ in der Prototype-Funktion und damit auch in C zugeordnet Ein char in Java ist nach dem UNICODE kodiert, im Gegensatz zu ASCII in C. Dem Java-Typ String kann nur eine Struktur zugeordnet werden. Funktionen zur Umwandlung von UNICODE und ASCII sowie die Definition String-Strukturen finden sich in der Header-Datei wieder javaString.h .

Variablen nicht als Parameter einer Methode sondern als Klassenvariable anzulegen ist objektorientiert und erm"oglicht einen einfachen Zugriff von C aus. Dieser

Zugriff kann als 'Call by Reference' bezeichnet werden. "Ubergibt man die Variablen als Parameter ist der Zugriff ein Call by Value. Alle Klassenvariablen werden im .h-File zu einem *struct* zusammengefa"st. Auf diesen erh"alt die aufgerufene C-Funktion als Parameter ein Handle-Pointer. Das in *StubPreamble.h* definierte, *unhand()*-Makro erm"oglicht den Zugriff auf die einzelnen Klassenvariablen. Das *unhand()*-Makro "ubernimmt einen Pointer auf das Handle einer Klasse und gibt einen Pointer auf die im .h-File erzeugte Klassenstruktur zur"uck. "Uber den R"uckgabewert des Makros lassen sich die Instance-Variablen der Java-Klasse direkt auswerten und ver"andern.

Strings

Wie schon erw"ahnt bilden *Strings* in Java eine eigene Klasse. M"ochte man einen *String* von Java nach C oder umgekehrt "ubergeben, mu"s man *javaString.h* im C-Implementationsfile mit einbinden. In *javaString.h* finden sich die Typdefintionen und Funktionen um *Strings* von Java nach C und umgekehrt zu transformieren. So gibt es noch weitere Methoden um *Strings* zu bearbeiten, wie z.B. *MoveString_GetString(), CString()* oder *makeJavaString()*. Java-*Strings* bilden eine eigene Klasse, somit wird bei der Konvertierung nicht nur ein elementarer Typ sondern eine ganze Klasse an die *native* Methode durchgereicht. Dieses Konzept l"a"st sich auch auf andere Klassen erweitern.

Felder

Felder werden von Java nach C "ubergeben indem vom *javah*-Tool ein Funktionsparameter von Typ *struct HArrayOf¡Objekt¿ ** erzeugt wird. Dieser Handle enth"alt ein Element *body* mit dem auf die Feldelemente zugegriffen werden kann. Der Zugriff auf das Feld erfolgt mit dem *unhand()*-Makro und dem *body*-Element.

Speichermanagment

in Java wird die Speicherverwaltung automatisch vom *Garbage-Collector* erledigt indem er die Methode *dispose()* aus der Java-Klasse ausf"uhrt. Wird in einer C-Funktion einer *nativen* Methode Speicher angelegt, so h"angt die Vorgehensweise des Programmierers beim Freigeben, von der Art des Speichers ab. Bei normalen Speicherbereichen f"ur die interne C-Anwendung mu"s der Programmierer sich explizit um die Freigabe k"ummern. Da der Java-Interpreter von diesem Speicher nichts wei"s, m"ussen eigene *native* Methoden implementiert werden um ihn freizugeben. In C erzeugte Speicherbereiche von Java-Objekten werden nicht unbedingt vom *Garbage-Collector* gefunden. Diese Java-Objekte, also Instanzen einer bestimmten Java-Klasse, k"onnen auch aus Java heraus freigegeben werden. Eine einfache Variante der Freigabe ist der explizite Aufruf der Methode *dispose()* wenn die Instanz nicht mehr ben"otigt wird.

Bibliography

[1] Silicon Graphics: OpenGL; http://www.opengl.org

[2] Woo, Mason: OpenGL Programming Guide 2nd Edition; Addison Wesley Developer Press April 1997; ISBN 0-0201-46138-2; http://www.aw.com/devpress

[3] Ron Fosner: OpenGL Programming for Windows 95 and Windows NT; Addison Wesley Second Printing April 1997; http://www.aw.com/devpress

[4] Nabajyoti Barkakati: X Window System Programming; Sams Publishing Secound Edition

[5] Mary Campione and Kathy Walrath: The Java Tutorial; Sun Microsystems; http://www.javasoft.com/docs/books/tutorial/index.html

[6] Leo Chan (lchan@cgl.uwaterloo.ca): OpenGL4Java; http://ftp.cgl.uwaterloo.ca/pub/software/meta/OpenGL4java.html

[7] Adam King (adam@opcom.ca): OpenGL4Java; http://www.magma.ca/%7Eaking/java

[8] Tommy Reilly (tom@pajato.com): Jogl; http://www.pajato.com/jogl

[9] Sun Microsystems: JavaCC Version 1.1; http://www.metamata.com/JavaCC/

[10] Helmut Kopka: LaTeX : Eine Einf"uhrung; Addison-Wesley 1991; ISBN 3-89319-338-3

[11] Cygnus Solutions, GNU-Win32 Project Version b18; http://www.cygnus.com/misc/gnu-win32

[12] Prof. Dr. Wolfgang Bunse (bunse@fhzinfo.fh-bielefeld.de); Fachhochschule Bielefeld; http://www.fh-bielefeld.de

[13] Goethel: GL4Java Homepage; http://www.jausoft.com

[14] Nikos Drakos (nikos@cbl.leeds.ac.uk): Latex2Html; http://cbl.leeds.ac.uk/nikos/tex2html/doc/latex2html/latex2html.html