JOGL and GlueGen Overview

Kenneth Russell

Sun Microsystems, Inc.

Sun microsystems

We make the net work.

Overview

- JOGL
  - Description
  - Comparison to other bindings
  - Features, Implementation
- GlueGen
  - Description
  - Features
  - Using GlueGen
  - Issues
- Build process, status, issues

JOGL

- Java™ programming language binding for OpenGL™ 3D API
- Exposes OpenGL 1.5 APIs and nearly all vendor extensions
- Designed with latest Java platform features in mind
- Integrates with AWT and Swing
- High performance
- Easy to use

JOGL

- Feature comparison:

	LWJGL	GL4Java	Magician	JOGL
Latest OpenGL support?	Yes	No	No	Yes
Supports New I/O?	Yes	Yes	No	Yes
Designed for New I/O?	Yes	No	No	Yes
Clean API?	Yes	No	Yes	Yes
Open source?	Yes	Yes	No	Yes
Swing/AWT support?	No	Yes	Yes	Yes
Available?	Yes	Yes	No	Yes
Actively developed?	Yes	No	No	Yes

JOGL Features

- Designed with New I/O at the core
  - Does not work on pre-1.4 JDKs
- Minimal API
  - Heavily inspired from Magician
    - GLEventListener, DebugGL, TraceGL
  - Some concepts/names borrowed from GL4Java
  - Hides unnecessary and problematic concepts
  - Does not expose OpenGL context directly
    - makeCurrent/free exchanged for GLEventListeners
  - Interface-based for flexibility

JOGL Features

- High performance
  - Game programming kept in mind
  - Optimized OpenGL context handling behind the scenes
- Latest OpenGL features
  - Pbuffers
  - Floating-point buffers
  - Vertex and fragment programs
  - ARB_shading_language_100
  - ARB_shader_objects

JOGL Features

- Implemented mostly in Java
  - ~50 lines of handwritten native code
  - Exceptions instead of (many) crashes
  - Improved diagnosability
  - Simpler code
  - Uses Java language features
    - Inheritance
    - Interfaces
    - Try/finally
    - Exceptions

JOGL Implementation

- Platform-specific Java code
- Uses JAWT to implement native drawing to AWT surfaces
- JAWT and OpenGL window system interface (wgl, glx) bound to Java programming language
- GlueGen tool written for this purpose

GlueGen

- Automatic JNI code generator
- Parses ANSI C header files
- Builds intermediate representation (IR) of functions and types
- Operates on IR to bind functions into Java classes as methods
- Emits Java classes, interfaces, and JNI code to call specified functions

GlueGen

- Many other similar tools available
  - GL4Java's C2J
  - SWIG
  - JNIWrapper
  - ...
- Why write a new one?

GlueGen Features

- Written in pure Java™
- Parser is built on ANSI C-compliant grammar for ANTLR (not handwritten)
  - Should handle most if not all C language constructs in function prototypes and type declarations
- Designed specifically for Java
  - Special handling of e.g. #defines
- Builds intermediate representation
  - Can transform code fairly drastically

GlueGen Features

- Extensible
  - Specialized subclasses for OpenGL-specific issues (i.e., calling through function pointers)
- Unique handling of C structs
  - Java data types wrapping New I/O buffers
- Powerful
  - Converting JOAL to use GlueGen: ~2 days
  - Upgrading JOGL from OpenGL 1.4 to 1.5: 1 day

Using GlueGen

1. Assemble header files to be parsed
   - GlueGen wraps all functions encountered unless specifically ignored
   - Usually necessary to stub out some headers
   - E.g.: place dummy version of windows.h, stdlib.h in include search path which contains only the typedefs necessary
   - When more than one header to be parsed, write a .c file which #includes both
   - See make/stub_includes/win32, x11, macosx

Using GlueGen

2. Write GlueGen configuration file
   - Controls code generation options (InterfaceOnly/AllStatic/ImplOnly), packages, classes to contain glue code, output dirs
   - Provides semantic information not present in C headers but necessary for Java
     - Length of returned arrays
     - Whether char* arguments are strings
     - Whether pointer arguments are held persistently and must be held in direct buffers, ...
   - Format not currently documented; infer from existing ones, look at code, or ask

Using GlueGen

3. Set up command line options
   - Include path, like C preprocessor (-I)
   - Emitter class (-E, defaults to JavaEmitter)
   - Configuration file (-C)
   - .c or .h files to process
4. Run GlueGen

GlueGen Issues

- Poor error reporting
  - Haven't figured out how to use #line directives to alter ANTLR's error reporting output
  - Work around by running PCPP (Pseudo C Pre-Processor) first, then running GlueGen on resulting file; line numbers are then correct
- Doesn't support all kinds of data types
  - Pointer-to-pointer is inherently problematic
  - Will throw exception if unsupported conversion is attempted
  - Fixing more of these as time goes by

Build process

1. ANTLR used to generate GlueGen's parser sources
2. GlueGen run in multiple stages
    1. GL interface class
    2. GL implementation class (platform-specific)
    3. Public WGL/GLX/CGL interfaces (for some window system-specific vendor extensions)
    4. Private WGL/GLX/CGL class (exposes window system APIs to Java)
    5. JAWT, GLU, optionally Cg

Build process

3. BuildStaticGLInfo run
   1. Maps OpenGL function names to the extensions containing them
   2. Helps with determining availability of extensions – but may need rethinking
4. BuildComposablePipeline run
   1. Generates DebugGL, TraceGL from GL class
5. Compile all Java code
6. Compile all native code

JOGL Status and Issues

- Mostly feature-complete
  - Recent additions: X11 visual selection bug fix, X11 multihead support, FSAA support
- Being actively used in many projects
- Some stability issues, in particular on ATI cards

JOGL Status and Issues

- Known issues with resource leaks; need to be fixed (in progress, by community)
- Some missing APIs (i.e., glMultiDrawElements) due to GlueGen limitations

JOGL Status and Issues

- AWT's inherently multithreaded nature causes problems
  - Hard to report errors as early as desired
  - Recent addition of full-scene antialiasing (FSAA) support to JOGL provided insights into how this may be improved
  - Stability issues in drivers
    - Hopefully will be resolved with better resource management (and synchronization?) inside JOGL

Q&A

Kenneth Russell

kenneth.russell@sun.com

Sun microsystems

We make the net work.