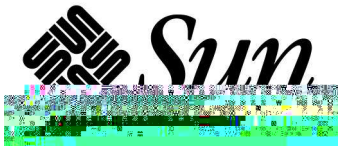
3D Application and Game Development With OpenGL®

Daniel Petersen
Kenneth Russell
Sun Microsystems, Inc.



Presentation Goal

Show how to build leading-edge 3D applications and games using the Java™ programming language and the OpenGL® 3D API

Agenda

Introduction
JSR Update
Using JOGL With AWT and Swing
Techniques for Application Development
Optimizing JOGL Applications
Demos
Conclusion

Agenda

Introduction

JSR Update

Using JOGL With AWT and Swing

Techniques for Application Development

Optimizing JOGL Applications

Demos

Conclusion

Introduction

- OpenGL
  - Powerful, cross-platform 3D API
  - Industry standard for writing 3D applications and games
    - On multiple platforms
    - From multiple programming languages
  - Supported by multiple vendors
  - Provides access to latest hardware features
    - Vertex and fragment programs
    - Floating-point framebuffers

Introduction

Java programming language OpenGL bindings

- A few bindings under active development
- LWJGL
  - Game-focused OpenGL binding
  - Single window
  - Usually full-screen
- JOGL
  - Integrates with AWT and Swing
- Relies on J2SE™ platform for, e.g., full-screen support
- Today's talk will use JOGL for examples

Introduction

Java programming language OpenGL bindings

- Standardization effort underway
- One Java programming language binding to OpenGL
  - Can be implemented by multiple vendors if desired
  - Being developed under the Java Community Process℠ service
  - JSRs 231 and 239

Agenda

Introduction

JSR Update

Using JOGL With AWT and Swing

Techniques for Application Development

Optimizing JOGL Applications

Demos

Conclusion

JSR Update

Overview

- What are JSR 231 and JSR 239
- Benefits of a JSR
- JSR 231 and JSR 239 Status

JSR 231 and 239

What they are

- JSR 231
  - Java bindings to OpenGL
  - Based on JOGL project on java.net
  - Most likely based on OpenGL 1.5
- JSR 239
  - Java bindings to OpenGL ES
  - Will use the GlueGen technology of JOGL to generate bindings
  - Most likely based on OpenGL ES 1.0
- Both being run under the Java Community Process version 2.6

Why Do a JSR?

Benefits of a JSR

- Align market around one specification
  - No need to download multiple APIs for the same binding
  - Specification and TCK ensure compliant bindings can be produced
  - Ensure all functionality of native library present
- Some industries (e.g., mobile devices) prefer JCP APIs

JSR 231 and 239

Status

- Both JSRs were filed and approved by their respective ECs
- Both Expert Groups have been formed and are meeting regularly
- EGs working together to make both APIs as similar as possible
- Both EGs hope to have EDRs late this year

JSR 231 and 239

Notes

- Both JSRs are fundamentally tracking a third-party API
- Want to track the OpenGL APIs as closely as possible
- Plan to use maintenance releases for updates
  - 30-day review period (shortest possible)

What You Can Do

More help is ALWAYS welcome

- Join an Expert Group
  - Specification work
  - SI work
  - TCK work
- Contribute to JOGL project on java.net
- Participate in EDRs

Agenda

Introduction

JSR Update

Using JOGL With AWT and Swing

Techniques for Application Development

Optimizing JOGL Applications

Demos

Conclusion

Using JOGL With AWT and Swing

DirectDraw incompatibilies on Windows

- `-Dsun.java2d.noddraw=true`
- Disables Java 2D™ API's internal use of DirectDraw APIs on Windows
  - Incompatible with OpenGL
  - Frequent driver bugs arise when mixing the two APIs, even when they are used in separate windows
- Should be specified for **all** JOGL applications!
  - No harm specifying on non-Windows platforms
  - Especially for Java Web Start applications
  - Add following to resources section of JNLP file:
  - `<property name="sun.java2d.noddraw" value="true"/>`

Using JOGL With AWT and Swing

GLCanvas and GLJPanel

- GLCanvas: heavyweight AWT widget for OpenGL rendering
  - Best performance (hardware accelerated)
  - Works in most GUI situations
  - See this article on mixing lightweight and heavyweight widgets successfully: http://java.sun.com/products/jfc/tsc/articles/mixing/
- `JPopupMenu.setLightweightPopupEnabled(false);`
  - To get Swing menus to overlap GLCanvas

Using JOGL With AWT and Swing

GLCanvas and GLJPanel

- GLJPanel: lightweight Swing widget for complete compatibility with Swing UIs
  - JInternalFrames
- Currently not hardware-accelerated
  - Poor performance
- Investigating using OpenGL pbuffers to implement GLJPanel
  - Faster, but still not fast enough
  - Still has texture readback
- Experimental work underway to integrate better with Java 2D API and “JFC/Swing”

Using JOGL With AWT and Swing

Rendering and animation options

- Automatic redraws initiated by the AWT
  - For static scenes
- Call `repaint()` in animation thread
- Use Animator class or start your own thread and call `GLDrawable.display()` directly
  - Most efficient for games
  - Allows optimized OpenGL context handling on some platforms
    - As efficient as single-threaded C code

Using JOGL With AWT and Swing

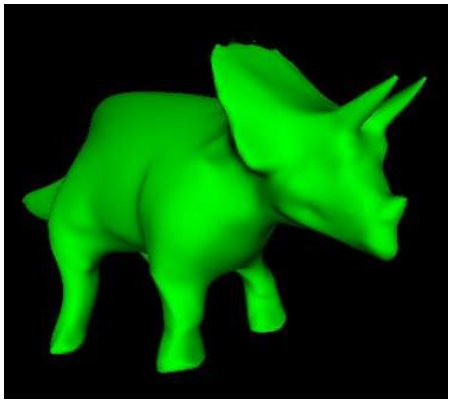
Multithreading

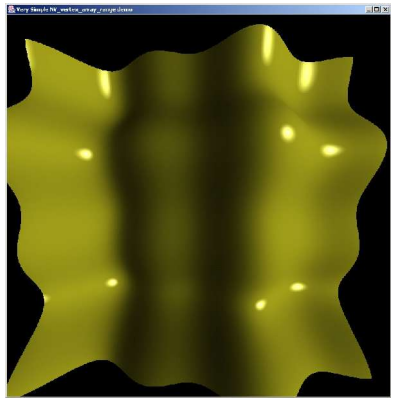
- AWT events like mouse and keyboard events are delivered on AWT event queue thread
- Not allowed / possible to make OpenGL calls directly inside these listeners
  - Though you can schedule or force a redraw
- Instead, pass information between these threads and any animation threads via member data
  - Use appropriate synchronization
  - Read data exactly once in your display() method
    - Avoids flickering and other artifacts during mouse interaction

Using JOGL With AWT and Swing

Examples

- See demos at http://jogl-demos.dev.java.net/ for examples of animation, interaction, and advanced features





Agenda

Introduction

JSR Update

Using JOGL With AWT and Swing

Techniques for Application Development

- Scene graphs and game engines
- Object picking
- Shadows

Optimizing JOGL Applications

Demos

Conclusion

Scene Graphs and Game Engines

Overview

- Higher-level, typically object-oriented layer for applications to build on top of
- Often have hierarchical structure
  - Good for representing character animation
- Use OpenGL or similar API at the bottom
  - Ones discussed here use JOGL
- Look for extensibility
  - Ability to call out to OpenGL from within scene graph to implement leading-edge effects

Scene Graphs and Game Engines

Examples

- Xith3D: http://www.xith.org/
  - General-purpose scene graph, but focused on gaming and high performance
  - Designed to be nearly identical to Java 3D™ APIs
  - Supports leading-edge functionality like shadow volumes and vertex and fragment programs
- Aviatrix3D: http://aviatrix3d.j3d.org/
  - Focused on visualization market
  - Minimal API design
  - Also supports vertex and fragment programs

Scene Graphs and Game Engines

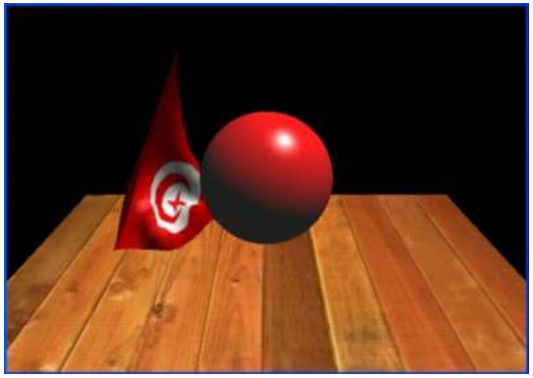
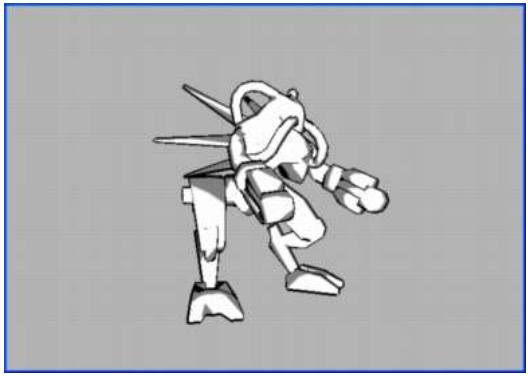
Examples

- OpenMind: http://www.mind2machine.com/
  - 3D game engine
  - Built-in support for 3D Studio Max ASE format
  - Supplies tool chain for developers

Scene Graphs and Game Engines

Demo

- Abdul Bezrati (a.k.a. “Java Cool Dude”)
  - Xith3D demos
  - http://xith.org/demo/JavaCoolDude.php


Object Picking

Using the selection buffer

- Interactive applications require the ability to pick objects in 3D
- OpenGL provides a built-in mechanism for object selection
  - Special rendering mode
- User supplies storage for results and sets up special “pick” matrix
  - View volume centered around cursor
- Any objects rendered into this view volume are reported to the user

Object Picking

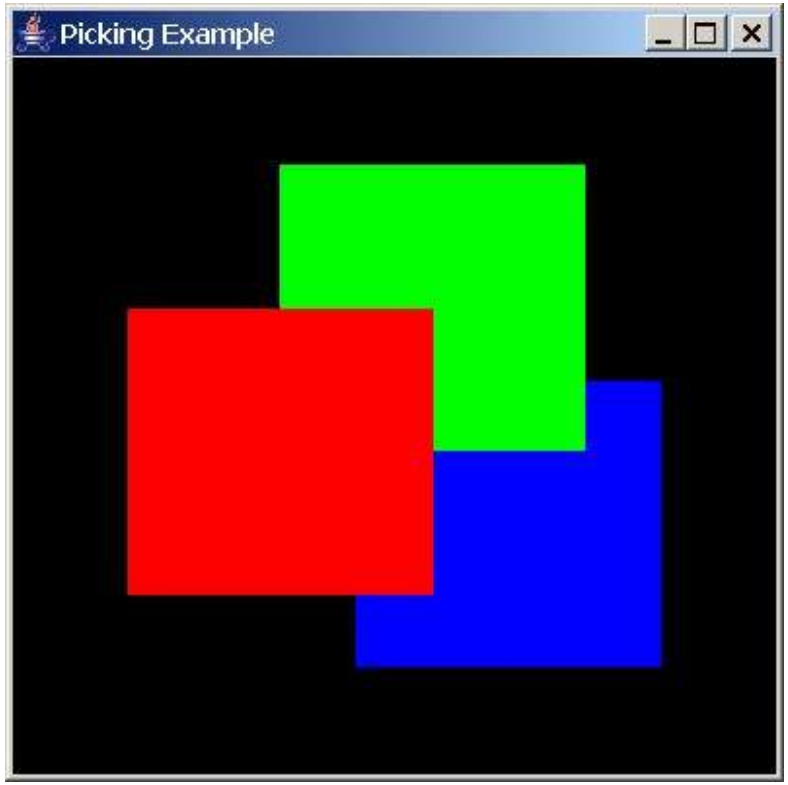
Using the selection buffer

- Set up selection buffer
  - ```
    IntBuffer buf =
      BufferUtils.newIntBuffer(1024);
    gl.glSelectBuffer(buf.capacity(), buf);
    ```
- Switch into selection mode
  - `gl.glRenderMode(GL.GL_SELECT);`
  - Color buffer is frozen at this point and not updated until selection mode is exited
- Initialize name stack
  - `gl.glInitNames();`
- Set up pick matrix
  - `glu.gluPickMatrix(...);`
    ```

Object Picking

Using the selection buffer

- Render objects, assigning names to them
  - ```
    int objectId = ...;
    gl.glPushName(objectId);
    renderObject(gl);
    gl.glPopName();
    ```
- Switch out of selection mode
  - ```
    int numHits =
      gl.glRenderMode(GL.GL_RENDER);
    ```
- Process hits
  - ```
    int idx = 0;
    while (idx < numHits) {
      int hit = buf.get(idx++);
      ...
    }
    ```
    ```

Object Picking

Demo

- Selection buffer demo (courtesy Thomas Bladh)
  - http://www.sm.luth.se/csee/courses/smd/159/TestPrograms/Picking.java



Object Picking

Using the selection buffer

- Advantages
  - Easy to start working with selection buffer
  - Can reuse normal rendering code; just add names
    - Names have no effect in GL_RENDER mode
- Disadvantages
  - Still have to disambiguate multiple hits based on depth values
  - No surface normal or other information at hit site
    - Not exactly casting a ray into the scene
  - To implement dragging behavior, still need some kind of policy for motion
  - Doesn't solve problem of moving camera in response to mouse motion

Object Picking

Using Manual Linear Algebra

- Picking can also be done at the application level
  - Perform ray-triangle intersection tests using a linear algebra library
- Depending on application's representation of geometry, may be able to accelerate drastically
  - Octrees or other spatial partitioning techniques
  - Degenerate cases like vertical rays
- Have full control over information returned and response to dragging
- May require some more code

Object Picking

Using Manual Linear Algebra

- Libraries exist for adding 3D interaction
  - gleem (OpenGL Extremely Easy-To-Use Manipulators)—in jogl-demos workspace on java.net
    - Will be shown shortly
  - Most scene graphs have picking mechanisms
    - Scene graphs discussed earlier support it
  - Depending on application and library, may be very easy to integrate the two

Shadows

Why do we need shadows?

- Humans use shadows to infer spatial relationships
  - Relative positions of objects
  - Locations of light sources
  - Shape of an object
- Scene looks more natural
- Scene is easier to understand
- Shadows look cool

Shadows

Two basic techniques

- Render-to-texture shadows
  - Image-space technique
- Volumetric
  - Geometric technique

Shadows

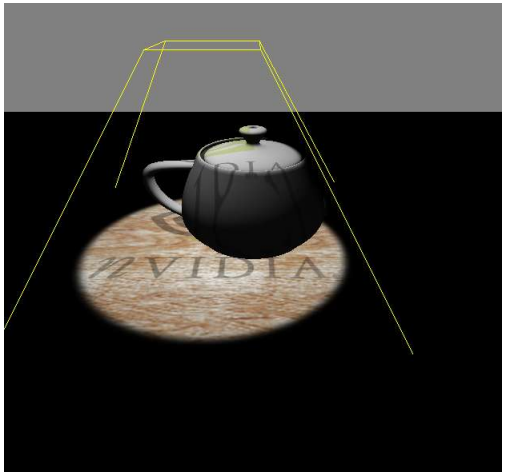
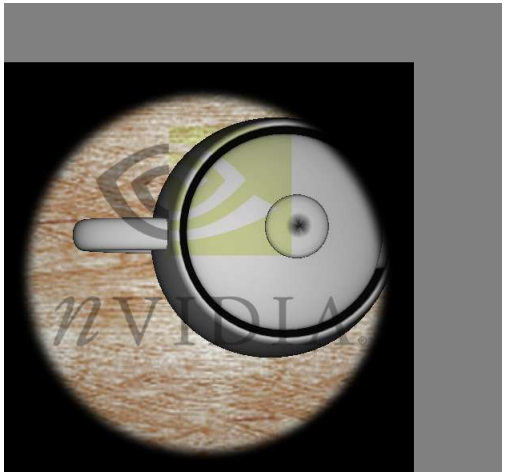
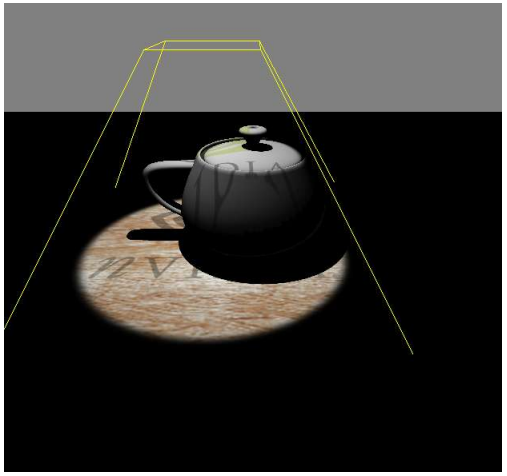
Render-to-texture shadows

- Render the scene from the light's perspective
- Store depth of rendered scene as texture
- Render scene from viewer's perspective
- Render the depth texture onto the scene
  - Careful setup of texture transform and texture-coord generation
    - Object's position maps to correct u-v texture coords in depth texture
    - Object's r texture coord maps to distance from the object to the light source
  - If r value is greater than texture value, pixel is in shadow

Shadows

Demo

- NVidia Hardware Shadow Mapping

Shadows

Render-to-texture shadows

- Advantages:
  - Performance independent of geometric complexity
  - No additional cost for animated geometry
  - Can take into account alpha-masked geometry (example: a chain-link fence)

Shadows

Render-to-texture shadows

- Disadvantages:
  - Dependent on texture resolution (aliasing)
    - Not good for long projections
  - Need special tricks to get self-shadowing to work well
  - Older hardware may not support render-to-texture in hardware
    - Fall back to slow framebuffer --> texture copy

Shadows

Volumetric shadows

- Basic idea: Use geometry to calculate volume of space that is in shadow
  - Calculate silhouette edge of object, from light's perspective
  - Extrude the silhouette away from the light
  - Objects inside this volume are in shadow from the light

Shadows

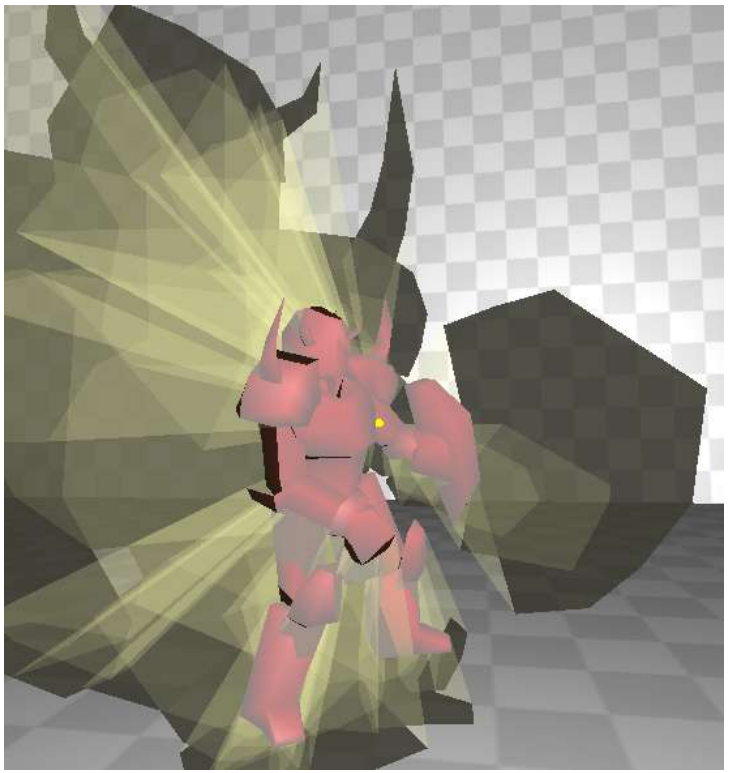
Volumetric shadows

- Uses stencil buffer for per-pixel in/out test
  - Render scene, ambient light only
    - Sets the depth buffer
  - Render shadow volumes with stencil enabled
    - Render front / back faces separately
    - If pixel passes depth test, adjust stencil value
    - Many adjustment heuristics (z-pass, z-fail)
  - If stencil value is 0 afterwards, pixel is not in shadow

Shadows

Demo

- NVidia Infinite Shadow Volumes



Shadows

Volumetric shadows

- Advantages:
  - Self-shadowing “just works”
  - No aliasing problems
  - Crisp shadows, even at infinite projection distances
  - Good for wide-open spaces

Shadows

Volumetric shadows

- Disadvantages:
  - Performance depends on scene
    - Expensive for complex objects, many lights, or many shadow receivers
      - N lights = N+1 render passes per shadowed object
    - Slow for non-static geometry / non-static lights
      - Silhouettes must be recalculated each frame
  - Incorrect shadows cast from alpha-masked geometry
    - Purely geometric technique
  - Many subtleties to make it work correctly for all intersections of light, viewer, and shadow volume

Agenda

Introduction

JSR Update

Using JOGL With AWT and Swing

Techniques for Application Development

Optimizing JOGL Applications

Demos

Conclusion

Optimizing JOGL Applications

Data organization

- Application writer needs to decide how to lay out data in memory
  - Multiple Java objects in heap?
  - Primitive types and/or primitive arrays?
  - New I/O? Use memory-mapped files instead of reading them in?
- Guide decisions by how various data structures will be used and how much data they store
- When compatibility with C data structures in memory-mapped files is required, GlueGen tool can help provide access to data
  - GlueGen is in JOGL workspace on java.net

Optimizing JOGL Applications

Data organization: Grand Canyon demo

- 300 MB of terrain data visualized in real time using Java technology and OpenGL
- Multiresolution algorithm
  - More detail for terrain closer to camera
- Two components of data: geometry and texture
  - NIO used to memory-map both
  - Highest-resolution geometry mapped all of the time
    - Processed by Java code to decimate to appropriate resolution
  - Appropriate resolution textures mapped in by background thread
    - Raw data handed off to OpenGL

Optimizing JOGL Applications

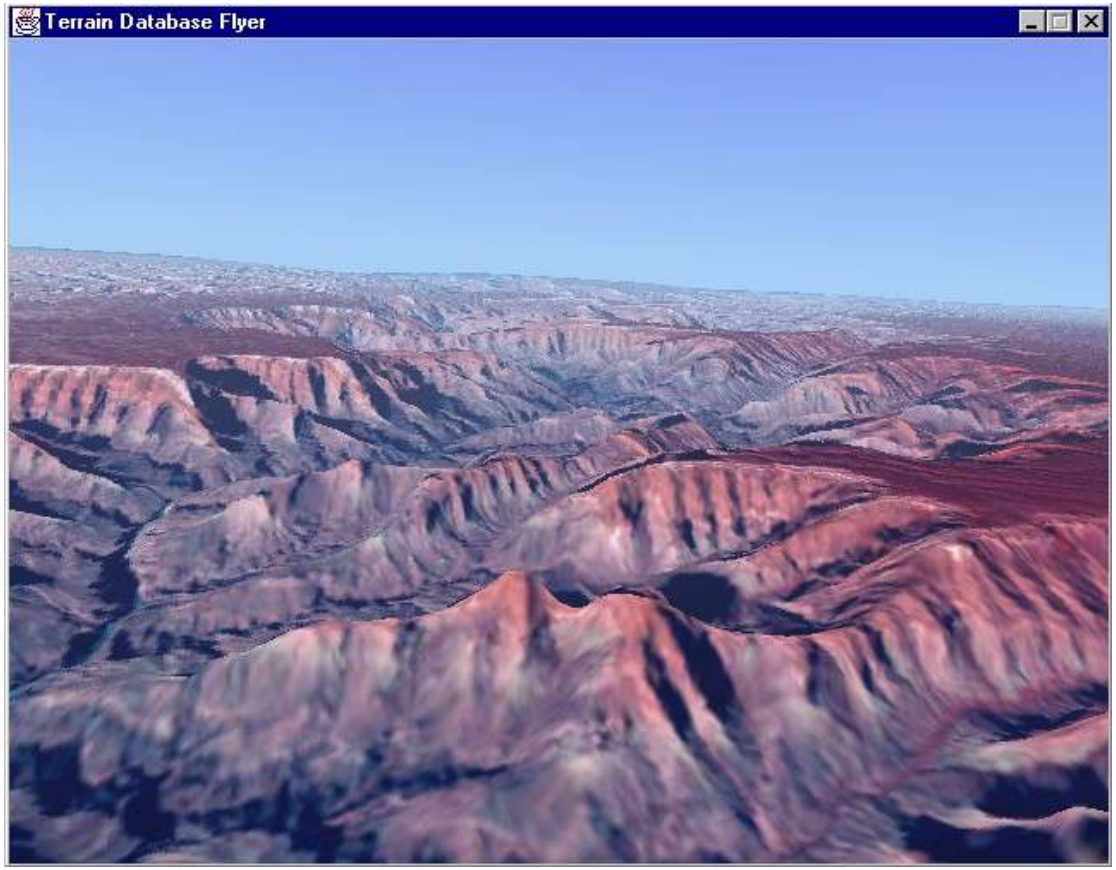
Data organization: Grand Canyon demo

- Very little data stored in Java objects heap
- Plenty of garbage generated, but all short-lived
  - No visible GC pauses
- Shows alternative to earlier programming models in Java language
  - E.g., all data read in to Java objects heap

Optimizing JOGL Applications

Demo

- Grand Canyon demo
  - http://java.sun.com/products/jfc/tsc/articles/jcanyon/



Optimizing JOGL Applications

Efficiency

- JNI has a non-zero cost
- All OpenGL routines are necessarily called from the Java programming language through JNI
- Minimize number of OpenGL function calls per frame
- Use vertex arrays and New I/O Float/Double/IntBuffers to store and send down geometric data to OpenGL
  - glVertexPointer, glNormalPointer, glColorPointer

Agenda

Introduction

JSR Update

Using JOGL With AWT and Swing

Techniques for Application Development

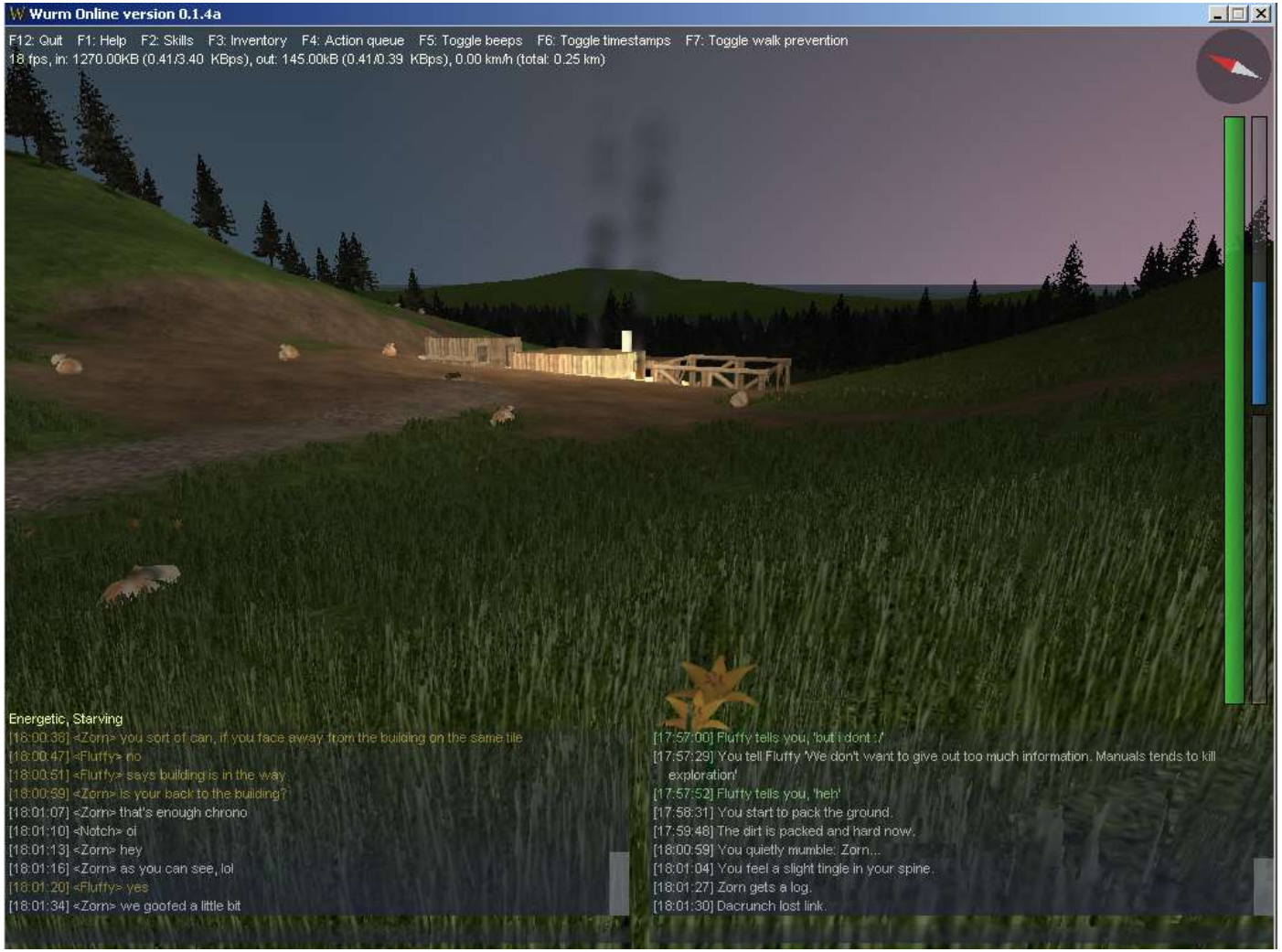
Optimizing JOGL Applications

Demos

Conclusion

Demos

Wurm Online



MOJANG specifications

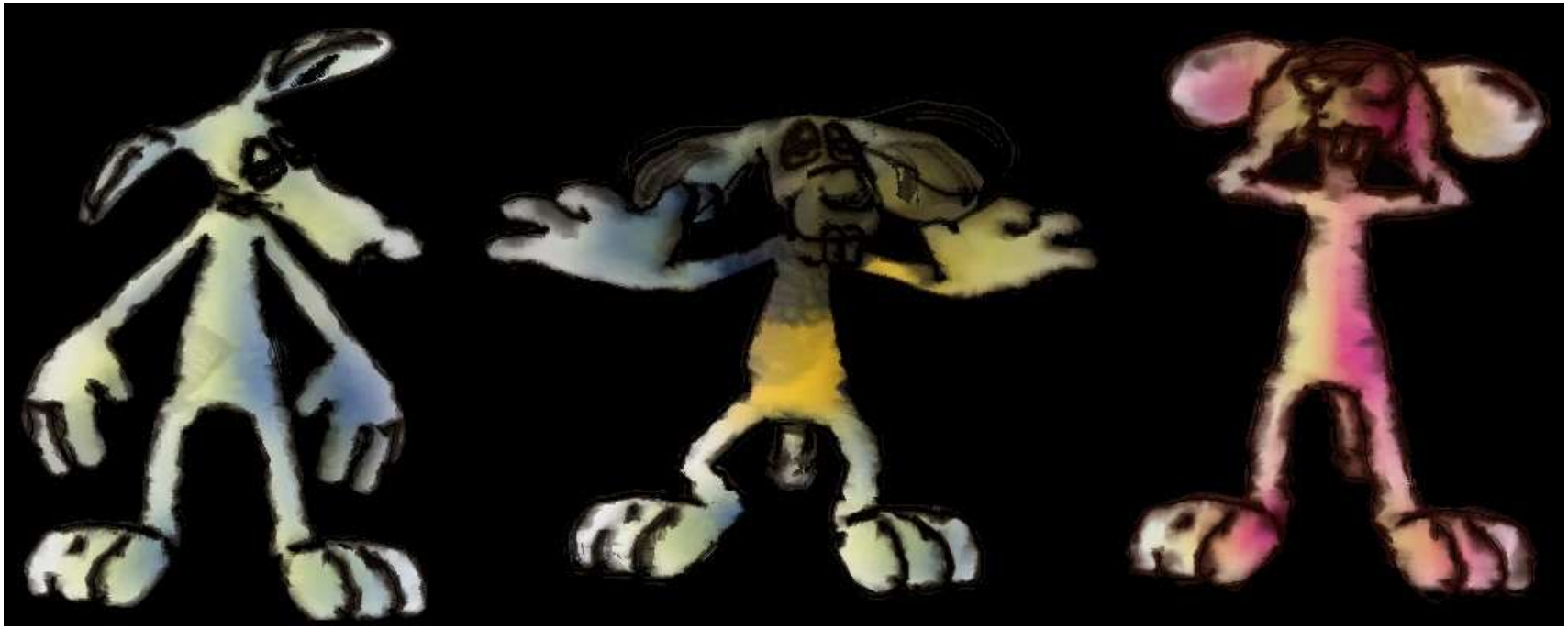
Demos

Wurm Online

- Being developed by Mojang Specifications
- Fantasy Massively Multiplayer Online Role Playing Game written in Java language using JOGL
- http://www.wurmonline.com/

Demos

Max



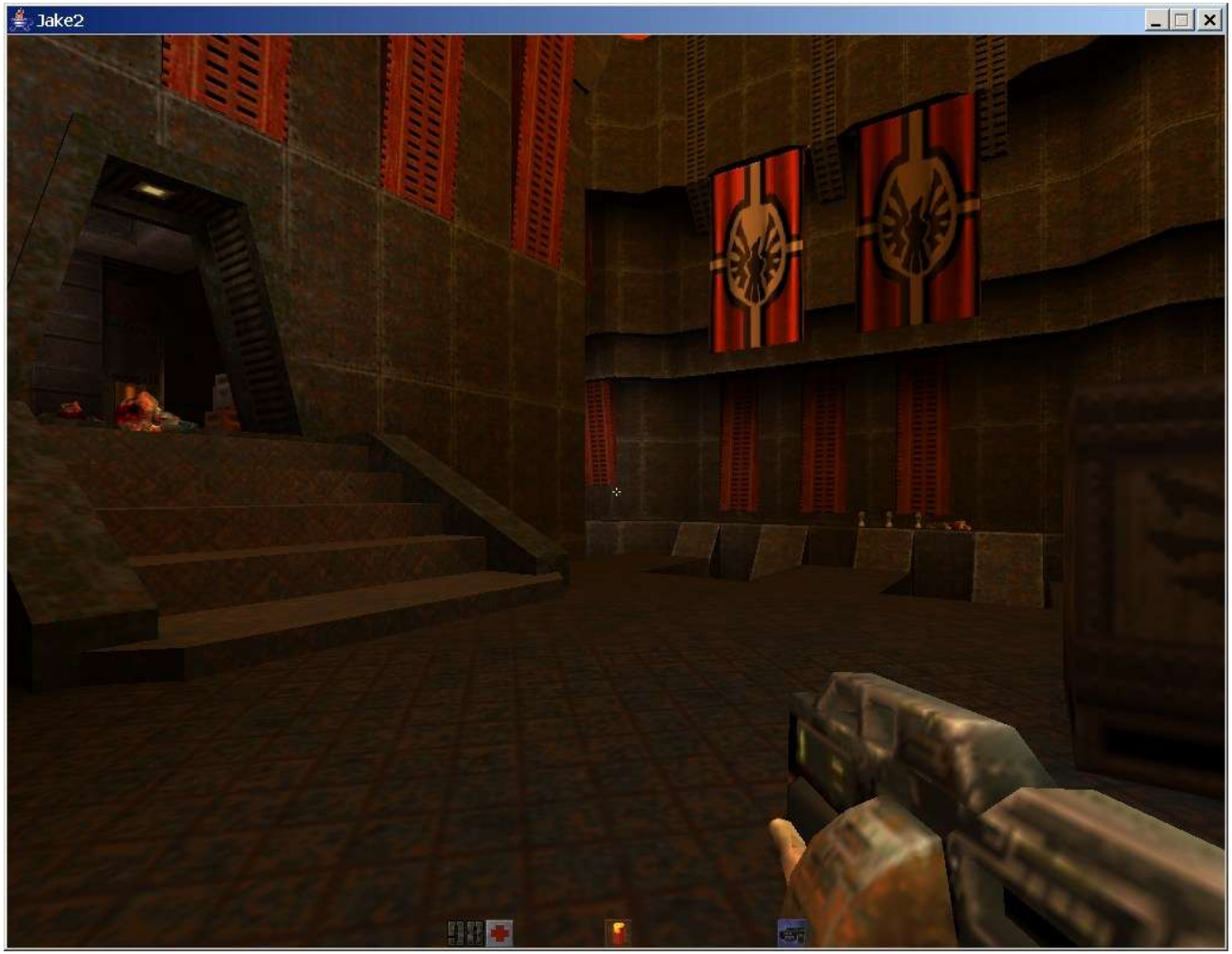
Demos

Max

- Developed by Synthetic Characters Group at The Media Lab, MIT
  http://characters.media.mit.edu/
- Explores behavior systems that:
  - Support teasing
  - Develop expectations about people they interact with
  - Develop over time, like animals do
- Characters code base written in Java programming language
  - Small pieces of native code to interface to custom input devices

Demos

Jake2



bytonic SOFTWARE

Demos

Jake2

- Port of Quake 2 engine to Java technology and JOGL
- Done by Bytonic Software http://www.bytonic.de/
- Illustrates that Java platform is capable of creating commercial-quality games
- Better than 85% of speed of original C
  - 210 fps compared to 245 fps

Agenda

Introduction

JSR Update

Using JOGL With AWT and Swing

Techniques for Application Development

Optimizing JOGL Applications

Demos

Conclusion

Conclusion

- Java platform and OpenGL 3D API provide the tools to develop leading-edge 3D applications and games
  - High performance, portability, and safety of the Java platform
- JOGL project and JSR 231 / 239 aiming for robust, full-featured, and easy-to-use interfaces to OpenGL
- Open source; join the development community
- Use the Java programming language for your next project

For More Information

- Technical Sessions
  - TS-1338 Desktop Game Development
- BOFs
  - BOF-1241 Meet the Java 2D API Team
  - BOF-1938 Meet the AWT Team
  - BOF-3215 Java 3D API
- URLs
  - http://jogl.dev.java.net/
  - http://community.java.net/games/

Q&A

3D Application and Game Development With OpenGL®

Daniel Petersen
Kenneth Russell
Sun Microsystems, Inc.