Java, Kinect and Gestural interfaces

Simon Ritter
Technology Evangelist, Oracle
Twitter: @speakjava
**The Man-Machine Interface**

How we interact with computers is changing (and will continue to do so)

Gone are the days of the keyboard and mouse

Behold! The rise of the gestural interface
Program Agenda

• The Man Machine Interface
• Building Interfaces: JavaFX and JMonkeyEngine
• Using the Wiimote with Java
• Data gloves and head tracking
• Using the Kinect with Java
• Putting it all together
• Conclusions and further information
THE MAN MACHINE INTERFACE
How It All Started
Progress was made...
Multi-touch has become popular
Gaming Has Driven Several Interfaces
Now It's About Gestures
BUILDING INTERFACES:
JAVAFX AND JMONKEYENGINE
Building An Interface: JavaFX 2.0

- Continuation of JavaFX product line
  - Now a Java API, no scripting language
  - Most APIs ported directly across
  - Things like binding and animation needed more thought
- Embrace more web technologies
  - Use of CSS for all JavaFX controls
  - Follow web spec for Drag-and-Drop
- Developers use scenegraph, not DOM
JavaFX and Gestural Interfaces

• Pro
  – Built in features like data binding and animations
  – Relatively simple Java API
  – Able to build rich, visually appealing interfaces

• Con
  – Currently limited to 2D environment
  – Engineering team working on 3D support
  – Some pseudo-3D support through perspective transform
Building An Interface: JMonkeyEngine

• A software system designed for the creation and development of video games
  – a game engine
• Built on top of OpenGL
  – Higher level constructs, rather than individual polygons
  – Camera, light sources, objects, quarterions, etc
JMonkeyEngine and Gestural Interfaces

• Pro
  – Full 3D support
  – Game engine facilities like collision detection, physics engine, etc

• Con
  – Hard to program (highly changeable APIs, poor backwards compatibility)
  – Games focused rather than generic interfaces
USING THE WIIMOTE WITH JAVA
Nintendo Wiimote

Java Interface

• Wiimote communicates using Bluetooth
  – Bluetooth stack needs to support L2CAP
  – JSR-82 Java Bluetooth API implementation
  – Wiimote specific Java APIs (IR sensors, accelerometer, etc)

• Mostly free and open source
  – Native Bluetooth stack (Windows, Linux, MacOS)
  – Avetana JSR-82 lib (free trial, €25 license)
  – Wiiremotej or motej Wii specific library
Wiimote Interface Code

Wiimote implements WiiRemoteListener

```java
public Wiimote() throws IOException {
    WiiRemote remote = WiiRemoteJ.connectToRemote("001CBE3C3E8B");
    remote.setIRSensorEnabled(true, WREvent.BASIC);
    remote.setLEDIlluminated(2, true);
    remote.addWiiRemoteListener(this);
}

public void buttonInputReceived(WRButtonEvent evt) { ... }
public void statusReported(WRStatusEvent evt) { ... }
public void accelerationInputReceived(WRAccelerationEvent evt) { ... }
public void IRInputReceived(WRIREvent wire) { ... }
// Other methods for extensions (balance board, nunchuk)
```
DATA GLOVES AND HEAD TRACKING
DIY Data Gloves

Hardware

- Sun SPOT controller
- Gyro sensor for more precise rotation data
- Three bend sensors for finger movement
Data Glove Architecture (Bend Sensor)
Data Glove Architecture (Gyro Sensor)
Hand Sensor Architecture

- **Glove SPOT**
- **IEEE 802.15.4**
- **Basestation**
- **USB Serial interface**
- **Hand Sensor Server**
Data Glove

Data Format

- Hand Identifier (left or right)
- Thumb bend value
- Index finger bend value
- Middle finger bend value
- Raw tilt (x and y)
  - From built in accelerometer
- Gyro compensated tilt (x and y)
  - Computed on SPOT
Hand Tracking Code (SPOT)

EDemoBoard demoBoard = EDemoBoard.getInstance();
thumb = demoBoard.getScalarInputs()[EDemoBoard.A0];
indexFinger = demoBoard.getScalarInputs()[EDemoBoard.A1];
middleFinger = demoBoard.getScalarInputs()[EDemoBoard.A2];
gyroX = demoBoard.getScalarInputs()[EDemoBoard.A3];
gyroY = demoBoard.getScalarInputs()[EDemoBoard.A4];
accelerometer = demoBoard.getAccelerometer();

currentPkt.writeInt(thumb.getValue());
currentPkt.writeInt(indexFinger.getValue());
currentPkt.writeInt(middleFinger.getValue());
currentPkt.writeInt((int)(accelerometer.getTiltX() * RADIANS_TO_DEGREES));
currentPkt.writeInt((int)(accelerometer.getTiltY() * RADIANS_TO_DEGREES));
currentPkt.writeInt(getTiltX());
currentPkt.writeInt(getTiltY());
Head Tracking Hardware

I2C to USB interface

Tilt-compensated compass

Also provides tilt and roll data
Head Tracking Hardware

I2C to USB Adapter

Compass Sensor

Mode unconnected to operate in I2C mode
Head Tracking

Data

• Use simple I2C protocol to read data
  – Start bit, module address, read/write bit, register number
• Compass value as a word
  – Direction returned as an integer angle in degrees
• Tilt of sensor (x axis)
• Roll of sensor (y axis)
• Serial interface is used to send bytes to I2C
Head Tracking Code

```java
portId = CommPortIdentifier.getPortIdentifier("/dev/ttyUSB0");
serialPort = (SerialPort) portId.open("compass", 500);
serialPort.setSerialPortParams(19200,
   SerialPort.DATABITS_8,
   SerialPort.STOPBITS_2,
   SerialPort.PARITY_NONE);
portInput = serialPort.getInputStream();
portOutput = serialPort.getOutputStream();

private void sendReadCommand(byte command, byte readSize {
   byte[] readCommand = new byte[4];
   readCommand[0] = I2C_AD1;          // I2C one byte access
   readCommand[1] = CMPS09_ADDR + 1;  // Sets write bit
   readCommand[2] = command;
   readCommand[3] = readSize;
   portOutput.write(readCommand);
}
```
Head Tracking Code

```
public int getBearing() {
    sendReadCommand(READ_BEARING_WORD, (byte) 2);
    int bytesRead = portInput.read(data);
    int hi = (int) data[0] & 0xFF;
    int lo = (int) data[1] & 0xFF;
    int b = (hi << 8) + lo;
    currentBearingInt = b / 10;
}

public int getTilt() {
    sendReadCommand(READ_TILT, (byte) 1);
    int bytesRead = portInput.read(data);
    currentTilt = data[0];
}
```
Demo
USING THE KINECT WITH JAVA
Kinect Sensor

- IR Light Source
- CMOS colour sensor (for RGB)
- CMOS IR sensor
- Motorised tilting base
- 3-axis accelerometer
- Provides 3D depth sensing
- Four downward facing microphones

Provides 3D depth sensing
Kinect Sensors

RGB Image
Depth Sensor Image
Body parts inferred by algorithm
3D body part data

Images from Microsoft Research: research.microsoft.com/en-us/projects/vrkinect/
OpenNI (Natural Interaction)

Kinect Java Interface

- Native C++ library in modular form (nodes)
  - Device
  - Depth generator
  - User generator
  - Image generator
  - IR generator
  - Scene analyser
- Java wrapper library using JNI
  - Required some bug fixing to make it work
  - These bugs have now been fixed
Kinect Java Code

```java
scriptNode = new OutArg<ScriptNode>();
context = Context.createFromXmlFile(config.getKinectXMLConfig(), scriptNode);
depthGenerator = DepthGenerator.create(context);
DepthMetaData depthMetaData = getDepthGenerator().getMetaData();
userGenerator = UserGenerator.create(context);
width = depthMetaData.getFullXRes();
height = depthMetaData.getFullYRes();
skeleton = new Skeleton(depthGenerator, userGenerator);
SkeletonCapability skeletonCapability = skeleton.getSkeletonCapability();

// Add listeners for events
context.startGeneratingAll();
```
Kinect Java Code

```java
circularSkeleton.get(user).put(joint,
    new SkeletonJointPosition(
        depthGenerator.convertRealWorldToProjective(pos.getPosition()),
        pos.getConfidence()));

...

public SkeletonJointPosition getJoint(int user, SkeletonJoint joint) {
    return getJointsForUser(user).get(joint);
}

...

SkeletonJointPosition jointPosition1 =
    skeleton.getJoint(user, SkeletonJoint.LEFT_SHOULDER);
Point3D pos1 = jointPosition1.getPosition();
```
PUTTING IT ALL TOGETHER
All Drivers Are Not All Created Equal
Especially Serial Ones

- I2C to USB interface requires serial driver
- Sun SPOT basestation requires serial driver
- Unfortunately they are different drivers
- Proved impossible to configure the classpath, java.library.path and LD_LIBRARY_PATH to make both work from the same JVM
Loopback To The Rescue

- Sun SPOT basestation
- Serial Driver
- TCP Loopback
- Server
- Main Controller
- Serial Driver
- TCP Loopback
- Kinect
- Main Controller
- Compass
- TCP Loopback

Insert Information Protection Policy Classification from Slide 8

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.
Demo Architecture

Camera

Simple 3D world (jmonkeyengine)

Event Listener

Head tracking events
- Pitch
- Roll
- Yaw

Hand Events
- Orientation
- Finger position
- Opn/close

Kinect events
- Joint positions
- User position
Demo
CONCLUSIONS AND FURTHER INFORMATION
Conclusions

- Java is still a really cool and powerful language
- Interfacing to exotic hardware is easy using free and open source libraries
- Modern hardware allows us to build some very interesting applications
- Be inspired, go out there and build more stuff!
Further Information

– wiiusej.googlecode.com
– javafx.com
– jmonkeyengine.com
– www.openni.org
– fivedots.coe.psu.ac.th/~ad/jg/ (Andrew Davidson at PSU)
– www.sunspots.com
The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.