JAviator3D*

Silviu Craciunas

Department of Computer Sciences
University of Salzburg, Austria
scraciunas@cs.uni-salzburg.at

April 18, 2007

1 Intro

Using Java 3D API here it is:

- full JAviator model with rotating propellers
- custom build models can also be imported (so we can even fly a beer with this thing)
- full 3D visualization (rotation, zoom, translation) of the scene using 2 types of behavior : orbital and centered
- customizable and extendable configuration for the scene(XML format)
- scene includes : axis, floor(textured and checkboard), fog, background(sky)
- integration in ControlApp
- ambient and directional lighting;
- shape colouring and lighting;
- shape positioning;

Java3D renders scenes on a special component called javax.media.j3d.Canvas3D. This component is an extension of the Abstract Window Toolkit (AWT) java.awt.Canvas class. While Canvas3D is also extensible, it is sufficient for most 3D applications.

In general, Java3D uses a scene graph for rendering purposes (as you may find out in your perusal of the Java3D documentation, this is not always true, e.g., immediate

*This readme file is for internal use only, do not distribute

mode does not require the use of a scene graph). A scene graph is a tree structure that contains Java3D nodes. Each node connection represents a parent-child relationship. A scene graph is constructed in such a way that state information cannot be shared among subgraphs. This enables Java3D to render scenes concurrently.

In this code, the creation of the scene graph requires the following substeps:

1. Create a branch group object.
2. Create the background and add it to the branch group.
3. Create the transform group and transformation and add the transform group to the branch group.
4. Compile the branch group.

The first object created is the javax.media.j3d.BoundingBox. The BoundingBox object specifies the cubical region in which the background will be active. In Scene.java, the BoundingBox is located at x, y, z coordinates 0.0, 0.0, 0.0, respectively, and has a size of 200.0 meters. By default, the background of our 3D scene is null and void (black). Just to make the scene a little more intersting, we will add an image to the background. The createBackground() method in scene.Scene contains the code required to add a background image to the scene.

A 3D transformation is the movement of a 3D point from one location to another. Rotations, scales, translations, and reflections are all examples of transformations. In Java3D, a 3D transformation is represented by the javax.media.j3d.Transform3D class. The scene graph group node that contains a Transform3D object is an instance of javax.media.j3d.TransformGroup. All objects in the scene are moved, rotated, scaled using a transformGorup.

You can interact with the scene by using the mouse :

- rotate view : hold left mouse button and move around
- translate view : hold right mouse button and move around
- zoom : hold both mouse buttons and move around

Important:

- in the 3DScene the axis are not like in the normal aerospace way .. Java3D uses the right hand coordinate system, meaning that the x-axis points toward the right, the y axis points up, and the z-axis points out of the screen.
- In Java3D, the meter is the default unit of measurement.
- for loading 3D models in 3DS format into Java 3D I used Inspector3DS/Loader3DS (http://www.starfireresearch.com/services/java3d/inspector3ds.html)
- if you use the default model the movement of the propellers is accurate but because of the 20 msec refresh rate it might seem a little unnatural

Listing 1: JAviator3DControl Interface

```
/**
 * interface function to create the scene and the model,
 * the implementation should remain the same
 */
public void createModel();

/**
 * reset the model,
 * should remain the same
 */
public void resetModel();

/**
 * send the sensor data
 * implementation cast to whatever you need
 * eg. in ControlApp the data is received as SensorData
 * @param data
 */
public void sendSensorData(Object data);

/**
 * send the motor values
 * implementation cast to whatever you need
 * eg. in ControlApp the data is received as ActuatorData
 * @param data
 */
public void setRotorSpeed(Object data);
```

2 Implamentation issues :

javiator.ui.JAviator3DControl - is an interface that specified the necessary methods to fly this thing

javiator.ui.JAviator3DControlPanel - is the implementation thereof for the controlPanel, you can change it as you wish for the tuningfork. Basically the only thing that matters is to create the scene using createScene(), to send the sensor data(here you can add your own packet type as long as it contains floats of roll, pitch, yaw, and z), send the motorSignals if you want to rotate the propellers accordingly and resetModel does what it name says it does

model

model.HelicopterModel - the implementation that uses a standard javiator CAD model without propeller rotation

model.JAviatorModel - the actual JAviator model implementation full with rotating propellers

model.Model - interface for the CAD model , here you can basically implement your own model and fly a cheeseburger around

scene

scene.Axis - some cool XYZ axis

scene.CheckBoardFloor - standard checkboard floor

scene.Colors - some definitions for colors

scene.Scene - the main scene

scene.SceneFog - scene fog (looks good with a fog density of 0.01 to keep a gradient feel but also let the textures be visible with light and reflection)

scene.TestScene - a test scene (just adds 3 objects in the center)

scene.TextureFloor - a floor with texture

simulation

simulation.Constants - constants, important is the refresh period, it is currently 20 msec synchronized with the period of the Javiator, you might want to change this for tuning fork

simulation.JAviator3D - main class

util

util.Configuration -read and save configuration in XML, change things like fog density, background image

util.KeyNavigatorTest - a simple test for openGL

3 How to run

Use the normal sequence : MockJAviator , Main and then run ControlApp with an additional argument : -3d

When the ControlPanel appears along with the 3D window try connecting the javiator and running the test sequence, each propeller should spin independently.

This is 2 weeks work so there might be small bugs and stupid things so I would appreciate your feedback.

For more information on java3D just google .. that is how I did it

That’s it!
enjoy :)