Sidebar – Java3D for Web-Based Robot Control

Igor Belousov

Hewlett-Packard 52/1, Kosmodamianskaya Nab., 115054 Moscow, Russia
igor.belousov@hp.com

1 Introduction

Robot control via Internet is a perspective direction of scientific research having important practical value. Recent period could be considered as a breakthrough moment to extend traditional use of Internet media from information exchanges to control of physical devices, including robots.

Internet limitations on communication bandwidth and data exchange rate make robot control based on TV images inefficient because of the slow responses made by the system to the operator's actions. One way of providing suitable control conditions for the operator is by using an on-line Java3D model of the robot and environment [BCC01] - a dynamic, Java3D-based virtual environment in which graphical models of the robot and objects are used to provide real-time visualisation of the current state of the robot site. Data transmission is restricted to small parcels defining the current coordinates of the robot and the objects; this has proved effective in practice, in contrast to TV images, which would have caused substantial communication delays. Under this regime, the robot can be successfully controlled, even when communication rates are extremely low (0.1-0.5 KB/sec). Moreover, the use of a 3D virtual environment significantly simplifies the remote performance of operations as it allows changes of viewpoint, zooming, the use of semitransparent images, etc. Web control interface is presented on figure 1

The use of open technology, Java3D, allows the virtual control environment to run on any type of computer platform. Moreover, any user of the Internet can use this environment for controlling the robots from within a Web page using a standard Web browser. Since Java3D is based on the OpenGL library, it runs rapidly on computers with OpenGL acceleration boards. For the scene in figure 1, we achieved a refresh rate of 25 frames/sec on a medium PC.

Java3D was developed by Sun Microsystems (java.sun.com), current version is Java3D 1.3.2 for Java2 (JDK 5.0). The following Java3D features makes it ideal instrument for development robotics software:

D. Brugali (Ed.): Software Eng. for Experimental Robotics. STAR 30, pp. 485–490, 2007.

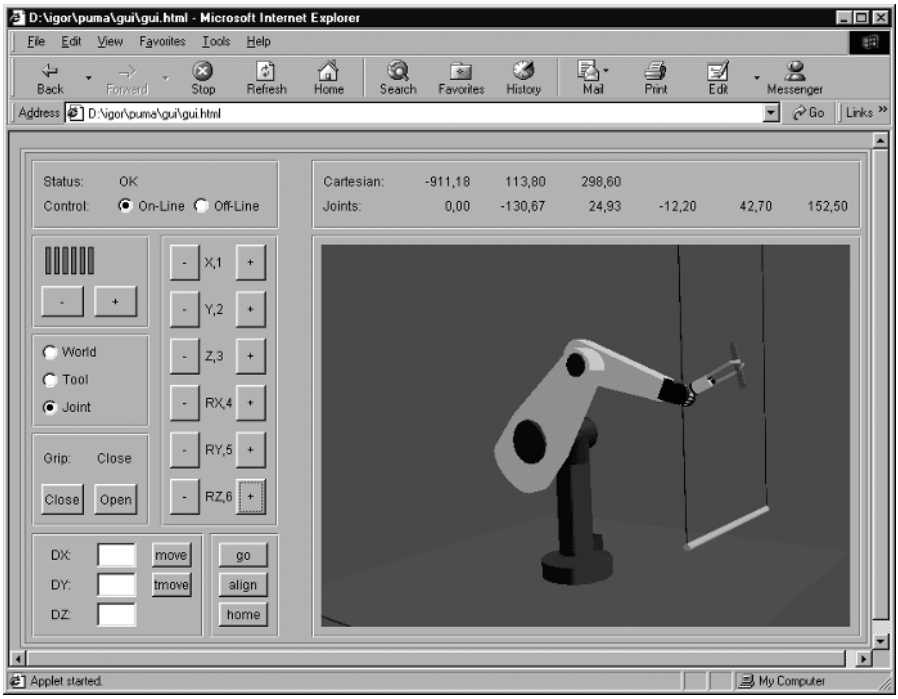

Fig. 1. Web interface for control of PUMA robot

- interactive 3D-visualisation in the network (applets)
- high performance (OpenGL at low level) ” 3D sound support
- LOD
- collision detection
- support of external 3D formats - 3D Studio (3DS), AutoCAD (DXF), VRML 97 (WRL), Wavefront (OBJ), etc.
- VR ready (head-mounted stereo displays, data-gloves and position trackers)
- supported platforms: Windows, Sun/Solaris, IBM/AIX, SGI IRIX, HP/UX, Linux
- free distribution and outstanding user support
- support of mobile devices (phones etc.)

Such Java3D feature as built-in collision detection algorithm has been used to provide operational security of the system. Before sending the control command to the robot site this command was simulated at the client site at the Java3D environment. If collision is detected, the corresponding command is cancelled. During the realisation of this procedure a bug in Java3D collision detection algorithm has been found (wrong collision was detected when the

objects are close to each other). It was reported to and fixed by the Sun's Java3D team.

Based on Java3D, the systems for control via the Internet of the PUMA 560 and CRS A465 manipulators and mobile robot Nomadic XR4000 [ABea00] have been developed (see figure 2). Systems were successfully tested under real Internet conditions from different locations in Russia, U.K., France and Korea [BC02]. The use of on-line Java3D-based control environment allowed to solve the most complicated task of robot control via Internet - it's interaction with fast moving objects [BSC05].

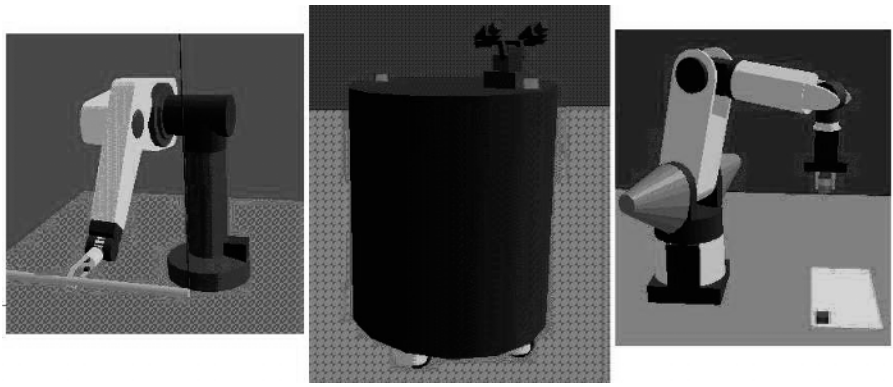

Fig. 2. Examples of the developed Java3D models of the different robots

Let's consider an example of Java3D program for generation of the model of the robot with several revolute joints. Program contains several classes. Base steps needed to create any 3D scene using Java3D are performed at the class Robot. Besides that constructor of the class Envir is invoked there. In the class Envir some objects of the robot working environment are defined, and constructor of the class Base is invoked. Class Base defines the fixed robot base and also is used to join to this base the first link of the manipulator, class Link1. Angle Link1.angle defines position of the first link relatively robot base. The particularity of the class Link1 - procedure draw(), which defines transformation of rotation corresponding to the change of the angle Link1.angle. Joining the second link Link2 is performed in the class Link1. This process continues for the rest joints and links of manipulator.

We can notice that it is needed to describe the 3D scene as a graph. Java3D graph should have a tree structure. The nodes of the graph are some classes of Java3D, for example geometric primitives or transformations. Links between the nodes are defined by the "parent-child" relations. In the above example the root of the tree is the node objRoot. Its "child" is the mouse transformation node. Then we have class Envir, etc. Hierarchy is presented in figure 3

```
public class Robot extends Applet {
 public BranchGroup createSceneGraph() {
  BranchGroup objRoot = new BranchGroup();
  TransformGroup tgMouse = HF.setMouseBehav(objRoot);
  Envir envir = new Envir();
  tgMouse.addChild(envir.getBG());
  return objRoot;
  }
 public Robot() {
  Canvas3D c = new Canvas3D(null);
  add("Center", c);
  BranchGroup scene = createSceneGraph();
  SimpleUniverse u = new SimpleUniverse(c);
  u.getViewingPlatform().setNominalViewingTransform();
  u.addBranchGraph(scene);
  }
}
```

Code structure of class *Robot*

```
public class Envir {
 private BranchGroup envirBG;
 public Envir() {
  envirBG = new BranchGroup();
  Box floor = new Box(2.0, 0.0, .0,
  Box.GENERATE_NORMALS,
    HF.createColorMaterialAppearance(Colors.green));
  envirBG.addChild(floor);
  Cylinder cyl = new Cylinder(0.05,
    0.2,Cylinder.GENERATE_NORMALS,
    HF.createColorMaterialAppearance(Colors.red));
  TransformGroup tgTranslCyl = HF.getTranslTG(0.8, 0.15, -0.1);
  envirBG.addChild(tgTranslCyl);
  tgTranslCyl.addChild(cyl);
  Base base = new Base();
  TransformGroup tgTranslBase = HF.getTranslTG(0.0, 0.2, 0.0);
  envirBG.addChild(tgTranslBase);
  tgTranslBase.addChild(base.getBG());
  }
 public BranchGroup getBG() {
  return envirBG;
  }
}
```

Code structure of class *Envir*

```
public class Base {
 private BranchGroup baseBG;
 public Base() {
  baseBG = new BranchGroup();
  Cylinder baseCyl=new Cylinder(R, H, Cylinder.GENERATE_NORMALS,
   HF.createColorMaterialAppearance(Colors.black));
  baseBG.addChild(baseCyl);
  Link1 link1 = new Link1();
  TransformGroup tgTranslLink1=HF.getTranslTG(0.0, 0.25, 0.0);
  Link1.tgRot = HF.getRotTG(2, Link1.angle);
Link1.tgRot.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE)
  baseBG.addChild(tgTranslLink1);
  tgTranslLink1.addChild(Link1.tgRot);
  Link1.tgRot.addChild(link1.getBG());
  }
 public BranchGroup getBG() {
  return baseBG;
 }
}
```

Code structure of class *Base*

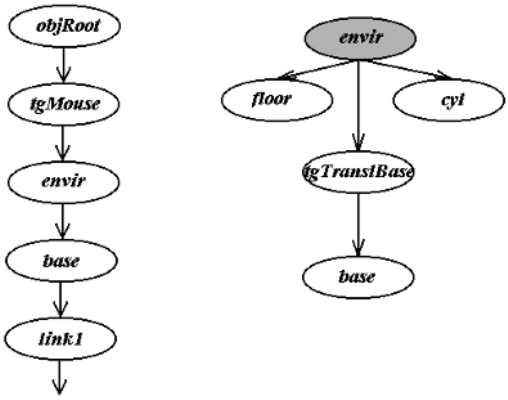

Fig. 3. Java3D scene graph.

Lets note that each node could be also presented as the tree, combined from the simple nodes. Thus, the developing of the Java3D program is generation of corresponding scene graph and its description using related functions of the Java3D language. This scene graph defines the program specifications.

```
public class Link1 {
 public static double angle = 0.0;
 private BranchGroup link1BG;
 public static Transform3D rot = new Transform3D();
 public static TransformGroup tgRot;
 public Link1() {
  link1BG = new BranchGroup();
  Cylinder baseCyl=new Cylinder(0.1, 0.2,
Cylinder.GENERATE_NORMALS,
    HF.createColorMaterialAppearance(Colors.wheat));
  link1BG.addChild(baseCyl);
  Link2 link2 = new Link2();
  TransformGroup tgTranslLink2 = HF.getTranslTG(0.0, 0.1, 0.0);
  Link2.tgRot = HF.getRotTG(3, Link2.angle);
Link2.tgRot.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE)
    link1BG.addChild(tgTranslLink2);
  tgTranslLink2.addChild(Link2.tgRot);
  Link2.tgRot.addChild(link2.getBG());
 }
 public BranchGroup getBG() {
  return link1BG;
 }
 public static void draw() {
  rot.rotY(angle);
  tgRot.setTransform(rot);
 }
}
```

Code structure of class *Link*1

References

[ABea00] R. Alami, I.R. Belousov, and et al., *Diligent: towards a human-friendly navigation system*, Proc. IEEE/RSJ Intern. Conf. on Intelligent Robots and Systems IROS'2000 (Takamatsu, Japan, October 30 - November 5), 2000.

[BC02] I. Belousov and G. Clapworthy, *Remote programming and java3d visualisation for internet robotics*, SPIE's International Technical Group Newsletter (Vol. 11, No. 1, February 2002, p. 8), 2002.

[BCC01] I. Belousov, G. Clapworthy, and R. Chellali, *Virtual reality tools for internet robotics*, Proc. IEEE Intern. Conf. on Robotics and Automation ICRA'2001 (Seoul, Korea, May, 2001, pp. 1878-1883), 2001.

[BSC05] I. Belousov, V. Sazonov, and S Chebukov, *Web-based teleoperation of the robot interacting with fast moving objects*, Proc. of the IEEE Int. Conf. on Robotics and Automation ICRA'2005 (Barcelona, Spain), 2005.