

OSG Virtual Planets 2.2.X - User Guide

Info: <http://www.osor.eu/projects/osgvp>
Author: Rafael Gaitán - rgaitan@ai2.upv.es
Author: Jordi Torres - jtorres@ai2.upv.es
Author: María Ten - mten@ai2.upv.es
Author: Jesús Zarzoso - jzarzoso@ai2.upv.es
Date: 2010-05-27, 08:40
Revision: 2
Description: This document is for application developers who want to include the osgVP functionality in their GIS software and as well too for those who want to contribute to the extension of the functionality of this toolkit.

Table of Contents

Introduction	7
System Requirements	9
Building libraries	11
Prerequisites	11
Environment variables	13
Compile sources	14
Import projects with Eclipse	16
Run the examples	18
Troubleshooting and FAQ	20
Download SDK	21
Installation	25
Getting Started	29
Creating a example step by step on Eclipse	29
OSGVP Viewer	35
Overview	35

Creating a Viewer	35
Camera manipulators	36
Display settings	37
MultiSampling	37
Stereo Settings	38
Intersections	38
Printing utilities	39
OSGVP Core	41
Osg features	41
Managing the scene-graph	41
Mathematic Tools	42
Positioning a Node	43
Geodes, Drawables & Primitives	43
Geometry creation	44
Using LODs	45
StateSets	46
Textures and Materials	47
Loading images	48
Updating a Node	50
GLSL Programming	50
Loading and saving scenes	52
Lighting	53
Adding Text	53
Text3D	53
Text	54
FadeText	54

Utilities	55
Tessellator	55
Optimizer	56
Camera HUD	56
Normals	56
OSGExtruder	57
OSGVP Terrain	59
The Terrain View	59
Create a terrain viewer	59
Set the scene data in a terrain viewer	61
Using camera manipulators	62
Define a terrain	63
Layer management	66
Adding a layer manager	67
Adding layers	67
Creating data drivers	68
Removing layers	70
Reorder layers	70
Get a layer	70
Forcing to retrieve data	71
Changing layer properties	71
Terrain utilities	72
OSGVP Features	75
Overview	75
Points	76
Polylines	78

Polygons	82
Text	83
Extruded Geometries	84
Particles	88
OSGVP Manipulator	89
The Manipulator node	89
Types of dragger	89
Adding a Node	90
Other available methods	90
Setting the Manipulator Handler	91
Manipulate an object	92
Managing the Scene with EditionManager	93
Methods implemented by Edition Manager	94
Implementing the picking functionality	95
The GeometryManipulator node	97
OSGVP Symbology	99
The symbol visitor	99
Basic symbols	99
Composite symbols	100
Extruded symbols	102
Node symbols	103

Introduction

Welcome to the OSG Virtual Planets (osgVP) 2.2.0 Users Guide. This manual serves as a reference to the osgVP library, and collecting documentation and examples provided with the code.

As you probably know, any software application requires some effort to learn. We have done our best to minimize the learning curve while making the process as enjoyable as possible. This document is a short programming guide that covers the basic and essential elements of the osgVP API.

In this section, you will be able to see the main structure of osgVP library and a brief introduction of supported features and architectural goals.

The osgVP is a set of libraries built specially for GIS development. As you can see in the image shown in Figure 1, it runs over OpenSceneGraph, an OpenSource cross-platform graphics toolkit for the development of high-performance graphics applications. Communication between layers is done through JNI. The classes belongs to this library are implemented by native calls (from Java to C++).

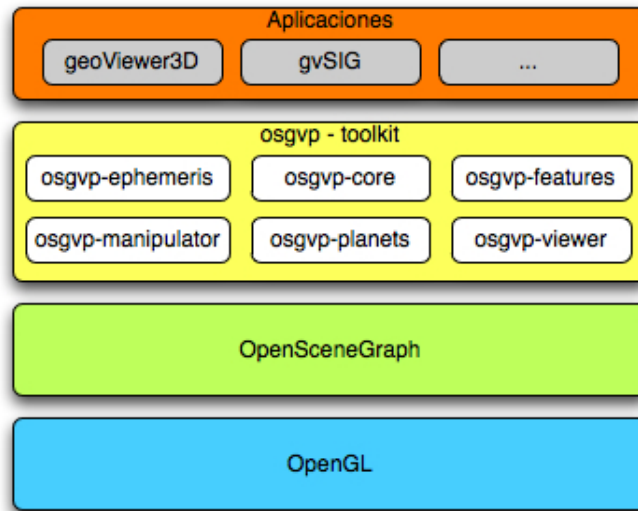


Figure 1: Architecture of the osgVP library.

The osgVP could be interpreted as an abstraction layer between JAVA-GIS applications and the render system. So, osgVP API is offered in JAVA. The following libraries were implemented.

- **osgVP-core:** involves necessary elements for building and optimize the scene graph. Also includes Mathematic tools to handle vectorial data.
- **osgVP-features:** serves to draw simple geometric figures such as text, points, lines, polygons, simple and extruded figures.
- **osgVP-geometries:** provides the necessary classes to describe the geometry of 2D and 3D features in a SIG.
- **osgVP-manipulator:** manages the edition of transformations associated to 3D objects. The library also manages modifications over Geometries.
- **osgVP-planets:** this library is deprecated, from now on osgVP-terrain will be used to create planets or terrain surfaces.
- **osgVP-symbolology:** provides a mechanism to represent GIS features like points, lines and polygons with 3D symbols.

- **osgVP-terrain:** allows developers to create Terrain specific scene graphs and manages geometry generation and memory paging. Also controls texture and elevation layer.
- **osgVP-viewer:** with this library users are able to create a scenegraph OSG viewer inside Java application, using JPanel or a integrated Canvas. JOGL is used in this library just to start a render context. There are several classes to take control over the Camera or Intersections.

System Requirements

- Minimum system requirements:** Pentium IV / 512 MB RAM / Graphics card OpenGL 1.5 compatible.
- Recommended:** Pentium IV / 1 GB RAM / Graphics card OpenGL 2.0 compatible.
- Operative Systems:** Windows (7, Vista, XP), Linux and Mac OS X.

Notes:

1. In the case Linux OS is used, the libc library installed should be version 2.4 or higher. Lower versions (for instance, in Ubuntu Dapper) will cause problems.
2. Tested in Windows (7, Vista and XP), Ubuntu Linux (9.10 and 8.10 release) and Mac OS X (on Mac Intel with Leopard and Snow Leopard).

Building libraries

Throughout this section, we show you how to download and build osgVP libraries step-by-step from sources. First, we establish the requisites for begin to build the libraries and you learn how to import Eclipse projects of our libraries. Finally we run some examples.

AS well as source code, we provide to developers a pre-built libraries of osgVP for most common operating systems. Therefore, you can skip this section and go to Download SDK if only you need to include the pre-built libraries in your own projects.

Prerequisites

Before compile osgVP libraries, you need to install some development tools.

- **SVN Client:** we recommend you to use the Tortoise SVN Client for Windows. It is available at: <http://tortoisesvn.tigris.org/>. On Linux, the system provides a SVN client, but if there is anyone you can download it using apt-get. On MacOSX, there is a SVN client available at terminal.
- Download the latest version of **Java SDK** available at: <http://java.sun.com>. We recommend installing the latest version, but osgVP can run with JDK 5 or higher. On MacOSX, Java JDK is integrated on Xcode SDK instead of be provided by SUN. Therefore, you need to install the developers tools available at Apple Developers Connection site (<http://developer.apple.com/mac/>) or in MacOSX DVD installer. Later, install the updates from *Software Updates* menu.
- **Apache Ant:** Usually MacOSX and some Linux systems provides Ant but you

can download the latest version available at: <http://ant.apache.org/> or using the apt-get command on Linux systems. After download the binaries, unzip the package and add the *bin* directory to your *PATH* environment variable.

- **Apache Maven:** Usually MacOSX and some Linux systems also provides Maven but you can download the latest version available at: <http://maven.apache.org/>. Then, unzip the package and add the *bin* directory to your *PATH* environment variable. You should also make sure you are using at least version 2.0.10.
- **CMake:** is a utility to cross platform compilation. Under Windows and MacOSX systems you can download the installer available at <http://www.cmake.org/>. Don't forget to check *Add CMake to the system PATH* during installation. On Linux systems, download the latest version available at CMake web or using the apt-get command. After download the binaries, unzip the package and add the *bin* directory to your *PATH* environment variable.
- **Source editor or compiler for C++:** On Windows systems, we are developing and testing with Microsoft Visual Studio 2008, however if you don't have a license for this software, you can download the *express* version for free at <http://www.microsoft.com/express/Downloads/#2008-Visual-CPP>. We strongly recommend you use this version of Visual Studio to compile because we provided all the necessary dependencies. On the other hand, it is possible to use newer versions of Visual Studio whenever do not forget to replace Visual Studio libraries with new ones in *.depman* directory. This folder is generated on your user home when Maven download or build osgVP native libraries. After *mvn clean* command, this directory is removed and you need to replace Visual Studio libraries again. Other compilers available for CMake can be used to compile osgVP however cannot guarantee error free compilation. On Linux and MacOSX systems, CMake generates the necessary makefiles to compile the C++ libraries of our project, by the way you can use any editor of source code. Xcode can be used for MacOSX to compile and debug our libraries but you need to modify the CMake properties to build the Xcode projects instead of the makefiles.
- **Eclipse:** is an excellent editor for Java development. You can use other software but we provide pre-built Eclipse projects for our libraries. Download the latest version of Eclipse at <http://www.eclipse.org/>. You also can use Eclipse to edit and compile C++ libraries.

Environment variables

On Windows systems you can modify your *PATH* going to *System Properties* on your *Control Panel* and clicking on *Environment Variables* as shown in the Figure 1. On Linux and MacOSX, edit the *.bash_rc* or *.bash_profile* located at your home directory to modify your *PATH* with any text editor. First, check if *JAVA_HOME* variable is set to your JDK directory instead of a JRE directory. If the variable doesn't exist add to your environment variables.

After, check if binaries directories for CMake, Ant and Maven are included on your *PATH* environment variable.

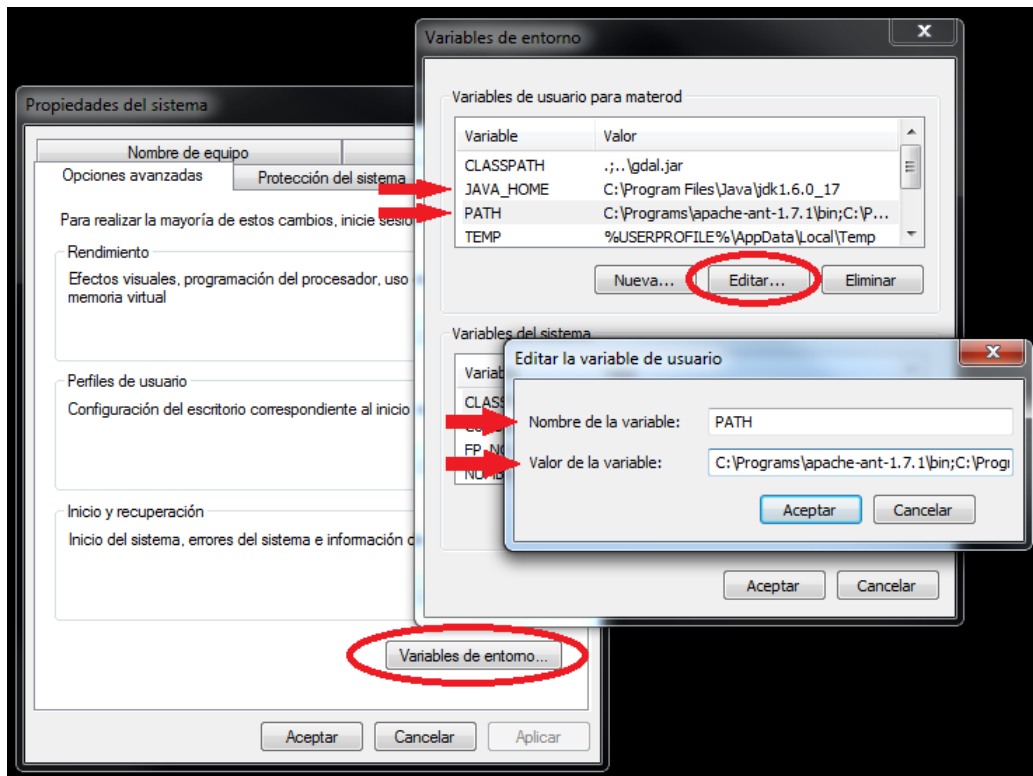
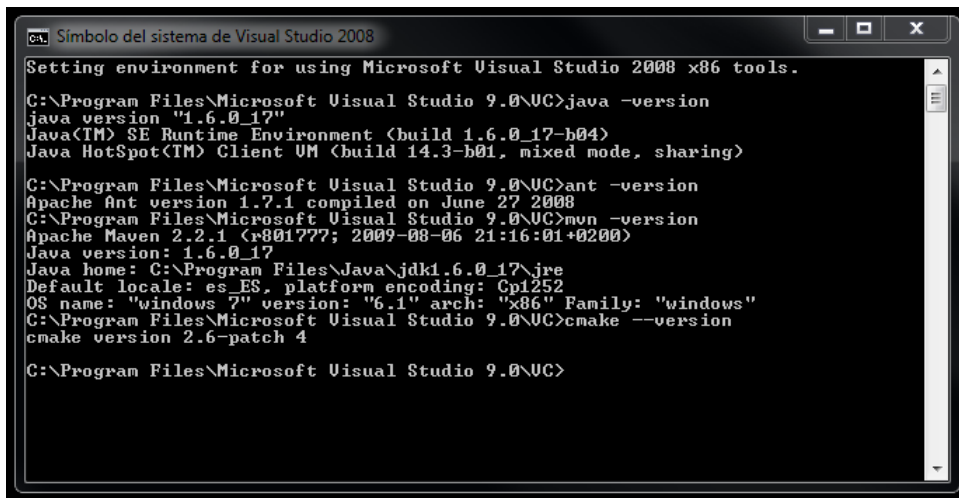


Figure 2: Example of setting environment variables on Windows systems.

Then, try execute the following commands in your command prompt to verify that the environment variables are set correctly:

```
java -version
ant -version
mvn -version
cmake --version
```



```
Símbolo del sistema de Visual Studio 2008
Setting environment for using Microsoft Visual Studio 2008 x86 tools.
C:\Program Files\Microsoft Visual Studio 9.0\VC>java -version
java version "1.6.0_17"
Java(TM) SE Runtime Environment (build 1.6.0_17-b04)
Java HotSpot(TM) Client VM (build 14.3-b01, mixed mode, sharing)
C:\Program Files\Microsoft Visual Studio 9.0\VC>ant -version
Apache Ant version 1.7.1 compiled on June 27 2008
C:\Program Files\Microsoft Visual Studio 9.0\VC>mvn -version
Apache Maven 2.2.1 (r801777; 2009-08-06 21:16:01+0200)
Java version: 1.6.0_17
Java home: C:\Program Files\Java\jdk1.6.0_17\jre
Default locale: es_ES, platform encoding: Cp1252
OS name: "windows 7" version: "6.1" arch: "x86" Family: "windows"
C:\Program Files\Microsoft Visual Studio 9.0\VC>cmake --version
cmake version 2.6-patch 4
C:\Program Files\Microsoft Visual Studio 9.0\VC>
```

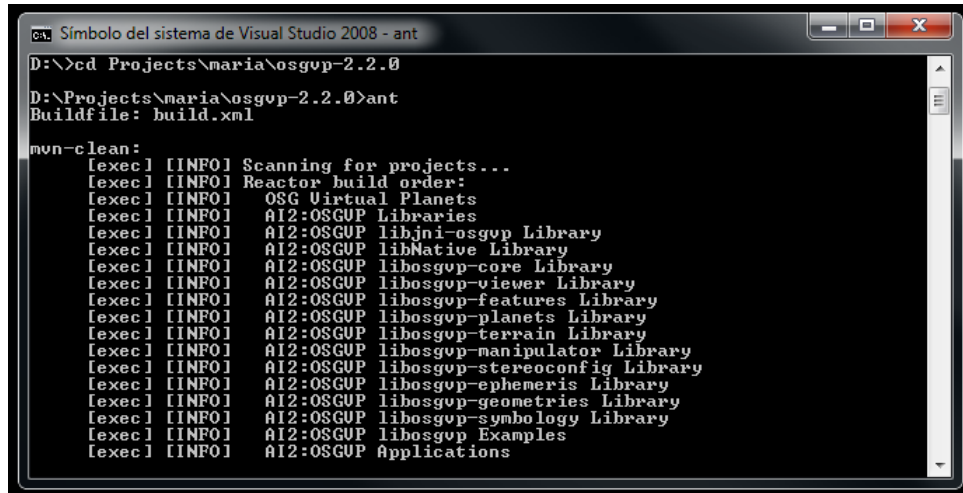
Figure 3: Testing environment variables are set correctly.

If you get some errors with these commands, one or more of the environment variables aren't correctly added. Please check again.

Compile sources

The source code of `osgVP` compiles using Ant and Maven from command prompt. However, on Windows systems you must use the **Visual Studio 2008 Command Prompt** available at *Start Menu » Microsoft Visual Studio 2008 » Visual Studio Tools*. This command prompt set all necessary variables to compile with Visual Studio. On Linux or MacOSX you can use any terminal.

On terminal, change directory to `osgVP-2.2.0` and execute `ant` command as shown in the Figure 3. This command runs the actions of `build.xml`; cleaning the workspace, downloading dependencies, compiling sources and building Eclipse projects.



```

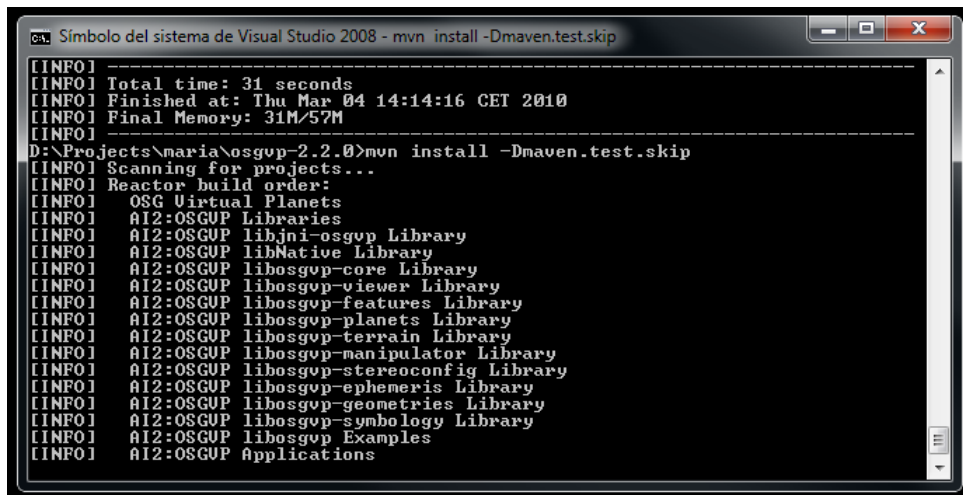
ca. Simbolo del sistema de Visual Studio 2008 - ant
D:\>cd Projects\maria\osgvp-2.2.0
D:\Projects\maria\osgvp-2.2.0>ant
Buildfile: build.xml

mvn-clean:
[exec] [INFO] Scanning for projects...
[exec] [INFO] Reactor build order:
[exec] [INFO]   OSG Virtual Planets
[exec] [INFO]   A12:OSGUP Libraries
[exec] [INFO]   A12:OSGUP libjni-osgvp Library
[exec] [INFO]   A12:OSGUP libNative Library
[exec] [INFO]   A12:OSGUP libosgvp-core Library
[exec] [INFO]   A12:OSGUP libosgvp-viewer Library
[exec] [INFO]   A12:OSGUP libosgvp-features Library
[exec] [INFO]   A12:OSGUP libosgvp-planets Library
[exec] [INFO]   A12:OSGUP libosgvp-terrain Library
[exec] [INFO]   A12:OSGUP libosgvp-manipulator Library
[exec] [INFO]   A12:OSGUP libosgvp-stereoconfig Library
[exec] [INFO]   A12:OSGUP libosgvp-ephemeris Library
[exec] [INFO]   A12:OSGUP libosgvp-geometries Library
[exec] [INFO]   A12:OSGUP libosgvp-symbolology Library
[exec] [INFO]   A12:OSGUP libosgvp Examples
[exec] [INFO]   A12:OSGUP Applications

```

Figure 4: Executing the command ant on Visual Studio 2008 Command Prompt.

First time you compile the osgVP libraries is recommended to use *ant* command because this command make all necessary compilation steps. Next time, you can use *mvn install* instead of *ant* to compile source. Some test could be out of date producing errors in compilation. To skip test use the command *mvn install -Dmaven.test.skip*.



```

ca. Simbolo del sistema de Visual Studio 2008 - mvn install -Dmaven.test.skip
[INFO] Total time: 31 seconds
[INFO] Finished at: Thu Mar 04 14:14:16 CET 2010
[INFO] Final Memory: 31M/57M
[INFO]
D:\Projects\maria\osgvp-2.2.0>mvn install -Dmaven.test.skip
[INFO] Scanning for projects...
[INFO] Reactor build order:
[INFO]   OSG Virtual Planets
[INFO]   A12:OSGUP Libraries
[INFO]   A12:OSGUP libjni-osgvp Library
[INFO]   A12:OSGUP libNative Library
[INFO]   A12:OSGUP libosgvp-core Library
[INFO]   A12:OSGUP libosgvp-viewer Library
[INFO]   A12:OSGUP libosgvp-features Library
[INFO]   A12:OSGUP libosgvp-planets Library
[INFO]   A12:OSGUP libosgvp-terrain Library
[INFO]   A12:OSGUP libosgvp-manipulator Library
[INFO]   A12:OSGUP libosgvp-stereoconfig Library
[INFO]   A12:OSGUP libosgvp-ephemeris Library
[INFO]   A12:OSGUP libosgvp-geometries Library
[INFO]   A12:OSGUP libosgvp-symbolology Library
[INFO]   A12:OSGUP libosgvp Examples
[INFO]   A12:OSGUP Applications

```

Figure 5: Executing the command mvn on Visual Studio 2008 Command Prompt.

There are useful commands that you can use in osgVP source directory. Notice that *ant* command executes the first three commands automatically.

- **mvn clean:** clean all previous build files. After use this command you need to compile all again.
- **mvn install -Dmaven.test.skip:** compile all the source code (including native libraries) and copy them into your local maven repository, skipping test.
- **mvn eclipse:eclipse:** build Eclipse projects. After run this command you only need to import the projects with Eclipse.
- **mvn eclipse:clean:** clean all previous Eclipse projects.
- **mvn test:** run unit test.
- **mvn install assembly:assembly -Dmaven.test.skip:** create a binary distribution of osgVP libraries. The distribution file is available at “osgVP-2.2.0 » build » product” directory.

Import projects with Eclipse

Open Eclipse and select **osgVP-2.2.0** folder like your workspace.

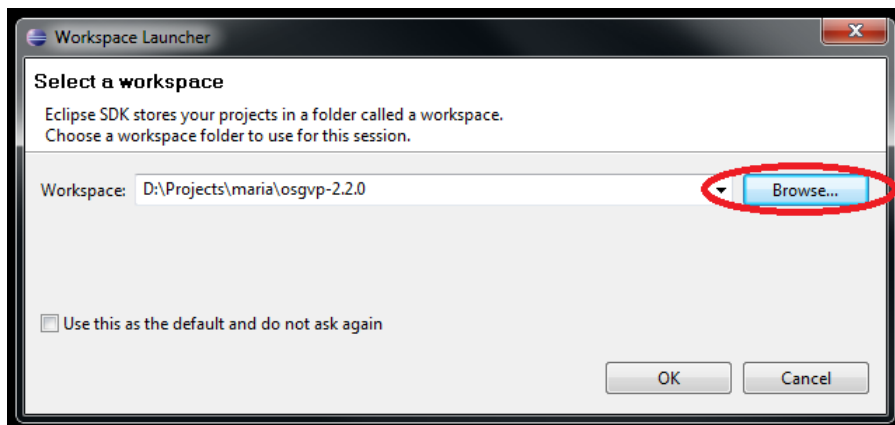


Figure 6: Select workspace directory on Eclipse.

Close *Welcome* tab and select *File* » *Import*. Then, select *General* » *Existing Projects into Workspace* at *Import window* and click *Next*.

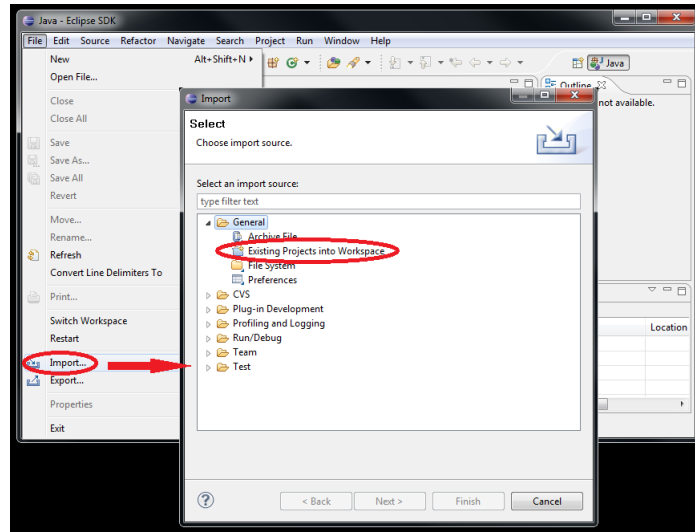


Figure 7: Select *Existing Projects into Workspace* at *Import* window.

Click on *Browse* button on *Select root directory* or write your path to **osgVP-2.2.0** folder. Some projects must appear on *Projects* list. Finally, press *Finish*.

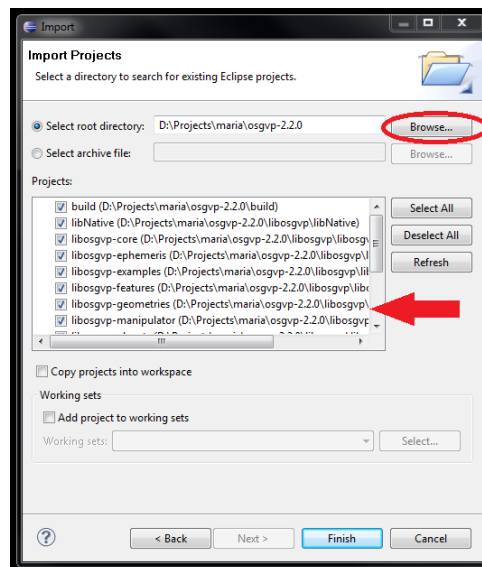


Figure 8: Set the path of your **osgVP-2.2.0** directory.

Eclipse compile all projects automatically after import step.

Run the examples

We provide a run configuration for osgVP examples. Press pull-down *Run* menu and select *Run Configurations*. Then pull-down *Java Application* and select *Examples Launcher*.

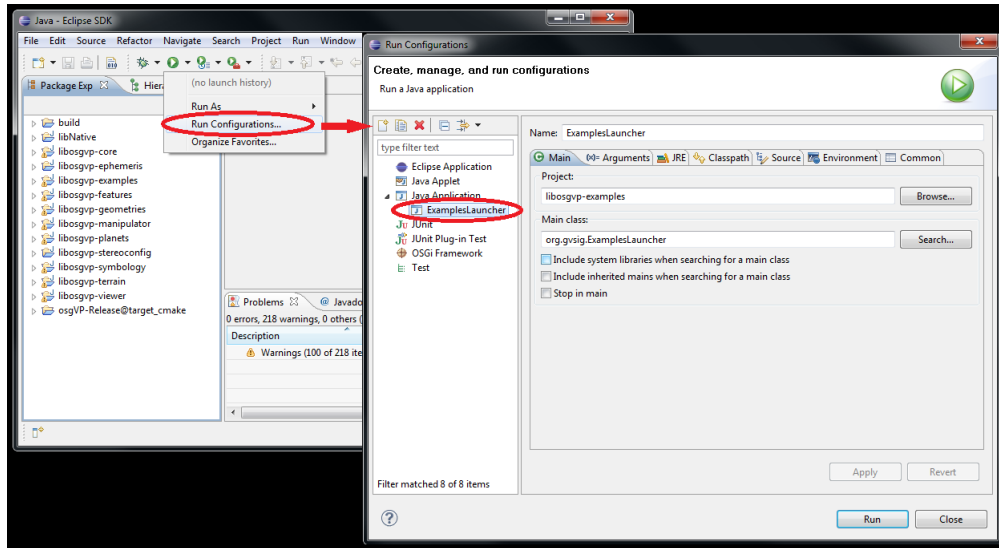


Figure 9: Select *Run Configurations* at *Run* menu.

The *ExamplesLauncher* is ready for use but you need to set the path of the osgVP native libraries before. Select *Environment* tab and edit the *PATH* variable on Windows, *LD_LIBRARY_PATH* on Linux or *DYLD_LIBRARY_PATH* on MacOSX. You must set the value of these variables to your *.depman* directory:

System	Variable	Value
Windows XP	PATH	c:\Document and Settings\ {username}\.depman\bin ¹
Windows Vista or 7	PATH	c:\Users\{username}\.depman\bin ¹
Linux	LD_LIBRARY_PATH	/home/{username}/.depman/lib
MacOSX	DYLD_LIBRARY_PATH	/Users/{username}/.depman/lib

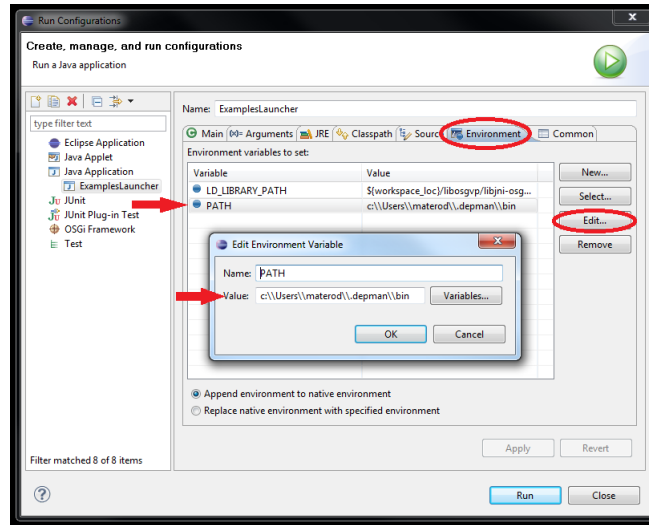


Figure 10: Editing *PATH* variable on Windows 7.

Finally press *Run* and the examples window is showed. Now, you can test osgVP examples.

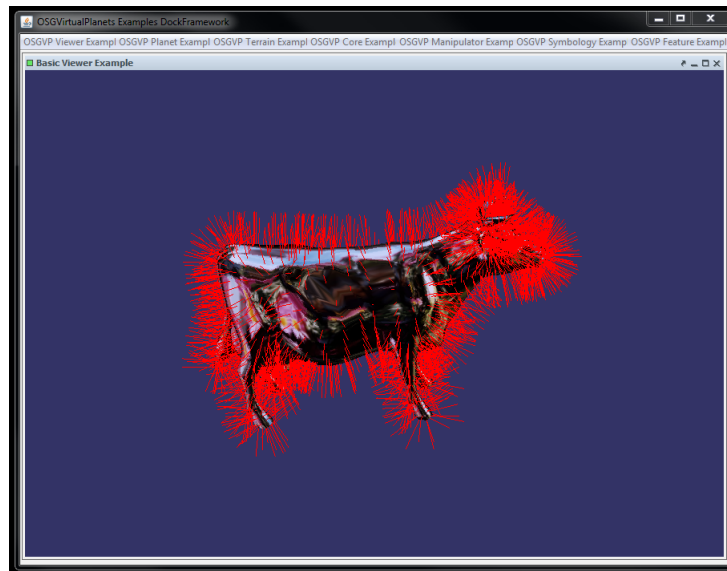


Figure 11: Running Basic Viewer Example.

¹Notice that “\” character is a special character on Windows and you must set “\\” to escape it.

Troubleshooting and FAQ

- **I have other JDK installed in the system like Eclipse JDK. Do I need to install the SUN JDK?** Yes, you have to install SUN Java JDK in your system and set the `JAVA_HOME` environment variable to this JDK.
- **I get some errors during compilation process when I run `mvn install -Dmaven.test.skip` or `ant` command.** Try to clean your workspace with `mvn clean` command. Check if you set all the environment variables and compile again. If you have other version of OpenSceneGraph libraries installed in your system, remove them from any environment variable before compile osgVP.
- **I try to run `mvn test` command and I get errors in JUnit test.** Don't worry, some test are out of date or need a special environment to be executed(don't work from command prompt).
- **When I import osgVP projects into Eclipse, there are only build and osgVP-Release@target-cmake in the projects list.** You have to compile and build osgVP projects before import them with Eclipse. Open the Visual Studio 2008 Command Prompt and run `mvn eclipse:eclipse`.
- **The osgVP Eclipse projects have some errors.** If you have the *Build Automatically* feature disabled on Eclipse, you need to build all projects with *Projects > Build All*.
- **I launch the examples and get this error: Unable to locate win32/bin/jniosgvpviewer.dll in class path or similar.** Your `PATH` environment variable is not set correctly. Follow the instructions of 6. *Run examples* and check the `PATH`.
- **I do some modifications in the osgVP source code, how can i build it again?** Use `mvn install -Dmaven.test.skip` command.

Download SDK

We provide to developers a pre-built libraries of osgVP for most common operating systems that you can use instead of compiling your own libraries. You can download this SDK from *Files* section at <http://forge.osor.eu/projects/osgvp/>. Each package contains native libraries whose format depends on your Operative System (.so for Linux, .dll for Windows and .dylib for MacOSX) and necessary jar files to compile and run your osgVP projects and compiled examples framework.

The SDK package contains:

- binaries directory: Native examples, binaries and pre-compiled dependencies per platform.
- lib directory: Jar files.
- examples script: Script to run the examples.

Resources required for carrying out osgvp projects:

- Jar files:
 - gluegen-rt-1.1.0.jar
 - idw-1.5.0.jar
 - jogl-1.1.0.jar
 - libNative-2.2.x.jar
 - libosgvp-core-2.2.x.jar
 - libosgvp-ephemeris-2.2.x.jar
 - libosgvp-examples-2.2.x.jar

- libosgvp-features-2.2.x.jar
- libosgvp-geometries-2.2.x.jar
- libosgvp-manipulator-2.2.x.jar
- libosgvp-planets-2.2.x.jar
- libosgvp-stereoconfig-2.2.x.jar
- libosgvp-symbolology-2.2.x.jar
- libosgvp-terrain-2.2.x.jar
- libosgvp-viewer-2.x.x.jar
- Native libraries:
 - jniosgvpcore.*
 - jniosgvpfeatures.*
 - jniosgvpmanipulator.*
 - jniosgvpplanets.*
 - jniosgvpstereoconfig.*
 - jniosgvpterrain.*
 - jniosgvpviewer.*
 - osgvpcore.*
 - osgvpfeatures.*
 - osgvpmanipulator.*
 - osgvpplanets.*
 - osgvpstereoconfig.*
 - osgvpterrain.*
 - osgvpviewer.*
- Native dependencies:
 - osg 2.8.2 libraries and plugins
 - gluegen-rt.*
 - jogl*.*
- Win 32 additional native dependencies:
 - Microsoft.VC90.CRT.manifest
 - gdal16.dll
 - glut32.dll
 - liblua.dll

- libpng13.dll
 - msvc?90.dll
 - zlib1.dll
- Mac OS additional native dependencies:
 - libfreetype.6.dylib

Once the osgVP SDK is downloaded the next step is to run the examples script located in root directory of the SDK package. If your OS is Windows you shall execute `run-Examples.bat`, in case you are running on Linux or on MacOSX you must execute `run-Examples.sh`.

All binaries and precompiled dependencies are placed inside binaries directory and the script exports the `PATH` to this directory, so you should view the examples framework and you will be able to execute any example in the framework.

If all is ok and previous process is correct, the image shown in the next Figure should be similar to your examples execution.

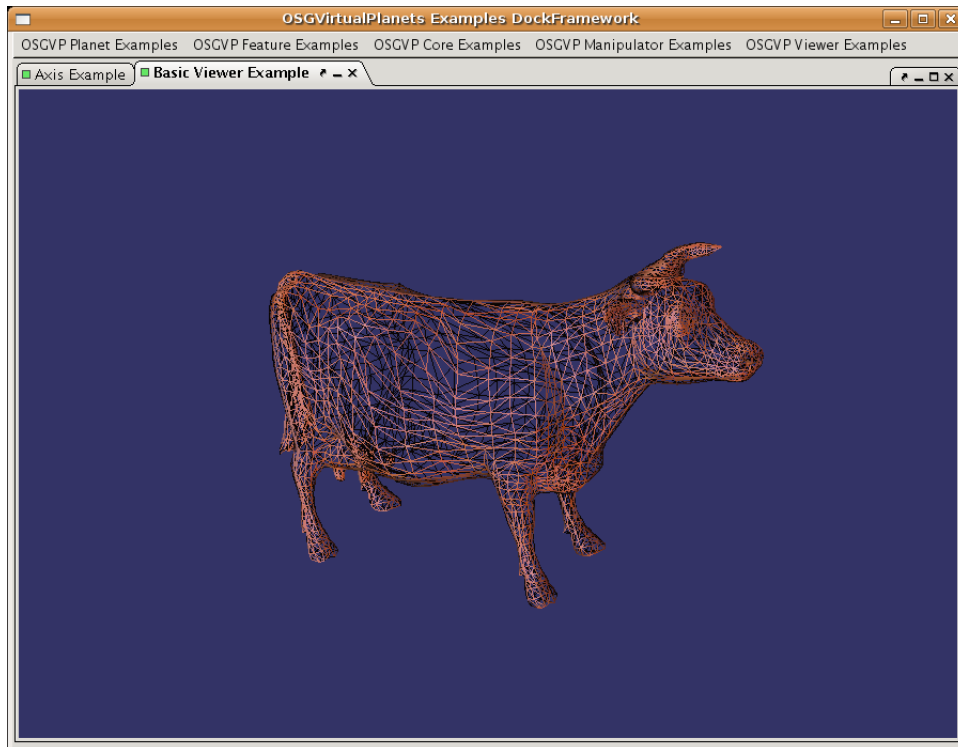


Figure 12: Examples framework. Running basicViewer Example in wired mode.

Installation

If you compile the osgVP SDK using ant or mvn install, the build files are located in .depman directory at your home. The dependencies are included in this directory too. By the way, these files are located at target directory in each project folder too.

Once you have downloaded the libraries or compile them from source code, there are several forms to include them into your project, depending how you are developing.

If you are developing with Eclipse, maybe the easy-way is creating a *lib* directory inside your project and include all the jar files into this directory, also create a *binaries* directory and add native libraries into this directory. Then add *lib* directory to the *java.build.path*, and to execute is necessary to add the correct environment variables (see Getting Started section).

If you want to use our library in your project you can get java binaries using Maven, it's necessary to add this dependencies section in your pom.xml.

```
<dependencies>
  <dependency>
    <groupId>org.gvsig.osgvp.libosgvp</groupId>
    <artifactId>libosgvp-core</artifactId>
    <version>2.2.0</version>
  </dependency>
  <dependency>
    <groupId>org.gvsig.osgvp.libosgvp</groupId>
    <artifactId>libosgvp-features</artifactId>
    <version>2.2.0</version>
  </dependency>
</dependencies>
```

```

</dependency>
  <dependency>
    <groupId>org.gvsig.osgvp.libosgvp</groupId>
    <artifactId>libosgvp-geometries</artifactId>
    <version>2.2.0</version>
  </dependency>
<dependency>
  <groupId>org.gvsig.osgvp.libosgvp</groupId>
  <artifactId>libosgvp-manipulator</artifactId>
  <version>2.2.0</version>
</dependency>
  <dependency>
    <groupId>org.gvsig.osgvp.libosgvp</groupId>
    <artifactId>libosgvp-stereoconfig</artifactId>
    <version>2.2.0</version>
  </dependency>
  <dependency>
    <groupId>org.gvsig.osgvp.libosgvp</groupId>
    <artifactId>libosgvp-symbology</artifactId>
    <version>2.2.0</version>
  </dependency>
<dependency>
  <groupId>org.gvsig.osgvp.libosgvp</groupId>
  <artifactId>libosgvp-terrain</artifactId>
  <version>2.2.0</version>
</dependency>
<dependency>
  <groupId>org.gvsig.osgvp.libosgvp</groupId>
  <artifactId>libosgvp-viewer</artifactId>
  <version>2.2.</version>
</dependency>
</dependencies>

```

Also need to add a new remote repository in your pom.xml. Notice the snapshotRepository tag offers a snapshot version compiled from osgVP developers with latest changes and modifications, but this version is not final or official release.

```
<repositories>
```

```
...
<repository>
  <id>osgvp-repository</id>
  <name>osgVP maven repository</name>
  <url>scp://shell.forge.osor.eu/home/groups/
    osgvp/www/maven-repository</url>
</repository>
<snapshotRepository>
  <id>osgvp-repository</id>
  <name>osgVP maven repository</name>
  <url>scp://shell.forge.osor.eu/home/groups/
    osgvp/www/maven-repository</url>
</snapshotRepository>
</repositories>
```


Getting Started

The osgVP has an ever growing number of examples available for developers to learn from. Following is a guide to getting these examples running. Don not forget to visit the section Installation before trying to run some examples.

Creating a example step by step on Eclipse

1. Create a new Eclipse project containing a Java file with this simple example code:

```
package org.examples.Main;

import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JFrame;
import javax.swing.JPanel;

import org.gvsig.osgvp.core.osg.PositionAttitudeTransform;
import org.gvsig.osgvp.core.osg.Vec3;
import org.gvsig.osgvp.core.osgdb.osgDB;
import org.gvsig.osgvp.exceptions.node.NodeException;
import org.gvsig.osgvp.features.Text;
import org.gvsig.osgvp.terrain.Terrain;
```

```
import org.gvsig.osgvp.terrain.TerrainCameraManipulator;
import org.gvsig.osgvp.terrain.TerrainViewer;
import org.gvsig.osgvp.terrain.CustomCameraManipulator.
    MouseButtonMaskType;
import org.gvsig.osgvp.viewer.Camera;
import org.gvsig.osgvp.viewer.IViewerContainer;
import org.gvsig.osgvp.viewer.ViewerFactory;

public class Main {

    private static IViewerContainer _canvas3d;

    public static void main(String[] args) {

        JPanel jContentPane = new JPanel();
        jContentPane.setLayout(new BorderLayout());

        JFrame jFrame = new JFrame();

        jFrame.setContentPane(jContentPane);
        jFrame.setTitle("Create Terrain View Example");
        jFrame.setSize(600, 400);
        jFrame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE);

        /**
         * Create a terrain viewer
         */
        TerrainViewer terrainViewer = null;

        try {
            terrainViewer = new TerrainViewer();
        } catch (NodeException e) {
            e.printStackTrace();
        }
    }
}
```

```
/**
 * Define the viewer type and add to the canvas
 */
_canvas3d = ViewerFactory.getInstance().
    createViewer(ViewerFactory.
        VIEWER_TYPE.CANVAS_VIEWER, terrainViewer);

jContentPane.add((Component) _canvas3d,
    BorderLayout.CENTER);
ViewerFactory.getInstance().startAnimator();

/**
 * Add a planet to the scene data
 */
Terrain earth = null;
try {
    earth = new Terrain();
    terrainViewer.addTerrain(earth);
} catch (NodeException e2) {
    e2.printStackTrace();
}

/**
 * Put the camera in the scene
 */
Camera cam = new Camera();
cam.setViewByLookAt(earth.getRadiusEquatorial()
    * 5.0, 0, 0, 0, 0, 0, 0, 0, 1);
terrainViewer.setCamera(cam);

/**
 * Add a cow in the north pole
 */
double factor = earth.getRadiusEquatorial()
    / 20.0;

try {
    PositionAttitudeTransform transform =
```

```
        new PositionAttitudeTransform();
transform.addChild(osgDB.
    readNodeFromFileFromResources(
        "/cow.ive"));
transform.setScale(new Vec3(factor,
    factor, factor));
transform.setPosition(new Vec3(0, 0,
    earth.getRadiusPolar() * 1.1));
terrainViewer.addFeature(transform);
} catch (NodeException e1) {
    e1.printStackTrace();
}

/**
 * Adding some information in the HUD
 */
try {
    Text info = new Text();
    info.setText("Terrain View Example");
    terrainViewer.addNodeToCameraHUD(info);
} catch (NodeException e1) {
    e1.printStackTrace();
}

/**
 * Customizing the manipulator
 */
TerrainCameraManipulator manip =
    (TerrainCameraManipulator) terrainViewer
        .getCameraManipulator();

manip.registryActionMask("AZIMUT",
    MouseButtonMaskType.LEFT_MOUSE_BUTTON,
    'a');

manip.setMinimumDistance(
    (float)earth.getRadiusEquatorial());
manip.setMaximumDistance(
```


3. Configure the Launcher. Open *Run Configurations* dialog on *Run* menu and create a new Java Application with this configuration:

- Name: SimpleExample
- Main Tab:
 - Project: simple-example
 - Main Class: org.example.Main
- Environment Tab:
 - **Linux:**

```
LD_LIBRARY_PATH=${workspace_loc}/binaries/linux32
```
 - **Windows:**

```
PATH=${workspace_loc}\\binaries\\win32
```
 - **MacOSX:**

```
DYLD_LIBRARY_PATH=${workspace_loc}/binaries/mac
```

4. Run the new launcher, if all is ok, then you should see the image shown in the next Figure.

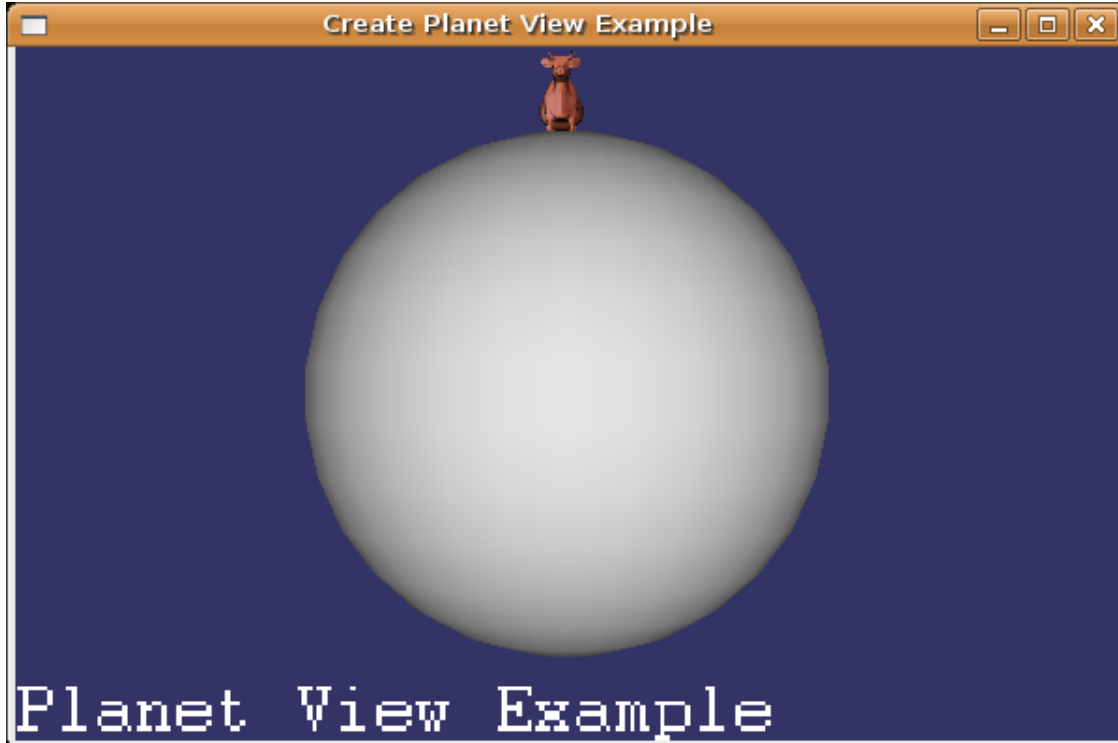


Figure 14: Simple example

OSGVP Viewer

In the following lines you might find the way to create a viewer and to be able to render your scenegraph from different views. Doing high resolution printing, making your viewer stereo or getting your scene multisampled are some of the features explained in this section.

Overview

A difficulty users have with OpenSceneGraph is complexity of building, with the number of external dependencies being a barrier to entry. Also when learning the API having multiple API's to learn adds to steepness of the learning curve - if we can provide a native viewer library all using the same matrix, memory management, and coding style and design then hopefully it'll become easier to learn.

With this library you are able to create different types of Viewer, including CompositeViewers and Offscreen Viewers. Offscreen Viewers are useful in the print process. They are implemented with pbuffers. Composite Viewers are util when we need different views of the same scene.

Creating a Viewer

First of all, you must create a new *IViewerContainer* variable to access the canvas and viewer properties.

```
private static IViewerContainer _canvas3d;
```

Later the *OSGViewer* instance can be created and assigned to the canvas. The first parameter of the *createView* method specifies the viewer type. A viewer can be a *CANVAS_VIEWER* or a *JPANEL_VIEWER* type. In the second parameter of the method you have to assign the recently created planet viewer. A planet viewer can be added into a *JPanel* and integrated in your application.

```
JPanel jPanel = new JPanel();
jPanel.setLayout(new BorderLayout());

OSGViewer viewer = new OSGViewer();

_canvas3d = ViewerFactory.getInstance().createView(
    ViewerFactory.VIEWER_TYPE.CANVAS_VIEWER,
    viewer);

jPanel.add((Component)_canvas3d, BorderLayout.CENTER);
ViewerFactory.getInstance().startAnimator();
```

Then we must attach a scene graph to it, and the viewer allows it to render. The way to do that is through a method called *setSceneData* in *OSGViewer* to add nodes into the scene graph.

Before your application finalizes you must call the *dispose* method of the viewer and stop the animator.

```
jFrame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        ViewerFactory.getInstance().stopAnimator();
        _canvas3d.dispose();
    }
});
```

Camera manipulators

A camera manipulator defines how to move the camera in the scene. You can attach a camera manipulator to the *OSGViewer*. There are different types of *CameraManipulators* and it should be easy to implement your own *CameraManipulator*. Each manip-

ulator modify the way that the mouse controls the camera position. The manipulators we implemented are :

- **DriveManipulator:** is a camera manipulator which provides drive-like functionality. By default, the left mouse button accelerates, the right mouse button decelerates, and the middle mouse button (or left and right simultaneously) stops dead.
- **FlightManipulator:** is a MatrixManipulator which provides flight simulator-like updating of the camera position & orientation. By default, the left mouse button accelerates, the right mouse button decelerates, and the middle mouse button (or left and right simultaneously) stops dead.
- **TerrainManipulator:** is a camera manipulator done to move the perspective in a Terrain scenerio.
- **TrackballManipulator:** is the manipulator created and attached to viewer by default. The TrackballManipulator class receives updates of mouse events in the form of GUIEventAdapter instances.

Display settings

This class serves, among other capabilities, to change the features of the viewer depending on the display type you want to use. The different kinds of displays that you can choose are: monitor, powerwall, reality center and head mounted display. Other thing you can do with this entity is apply multisampling to reduce aliasing.

MultiSampling

According to the OpenGL `GL_ARB_multisample` specification, "multisampling" refers to a specific optimization of supersampling. The specification dictates that the renderer evaluate one color, stencil, etc. value per pixel, and only "truly" supersample the depth value. (This is not the same as supersampling, but by the OpenGL 1.5 specification[2], the definition had been updated to include fully supersampling implementations as well.) In graphics literature in general, "multisampling" refers to any special case of supersampling where some components of the final image are not fully supersampled. Most modern GPUs are capable of this form of antialiasing, but it greatly taxes resources such as texture bandwidth and fillrate. Let's see some example:

```
//create the viewer
OSGViewer viewer= new OSGViewer();
DisplaySettings ds= new DisplaySettings();
//set multisamples
ds.setNumMultiSamples(4);
//set display settings to the viewer
viewer.setDisplaySettings(ds);
```

Stereo Settings

The osgVP has support for anaglyphic stereo (i.e. red/green or red/cyan glasses), quad buffered stereo (i.e. active stereo using shutter glasses, or passive stereo using polarized projectors & glasses) and horizontal and vertical split window stereo implementations. Almost all OSG applications have the potential for stereo support simply by setting the relevant environmental variables, or using DisplaySettings class. Little or no code changes will be required, the support is handled transparently inside the sceneview handling of rendering. To accomplish stereo settings from code you can follow next lines.

```
//create the viewer
OSGViewer viewer= new OSGViewer();
DisplaySettings ds= new DisplaySettings();
//enabling stereo
ds.setStereo(true);
//set the stereo preferred mode
ds.setStereoMode(DisplaySettings.StereoMode.ANAGLYPHIC);
//set display settings to the viewer
viewer.setDisplaySettings(ds);
```

For more information about making stereo viewing you can look at www.openscenegraph.org.

Intersections

Typically, 3D applications need to support user interaction or selection, such as picking. The osgVP-viewer library efficiently supports picking with some classes that test the scene graph for intersection. Next piece of code demonstrates how to obtain the intersections from a ray traced from View point.

```
Intersections hits = getCanvas3D().getOSGViewer().rayPick(
    _manager, arg0.getX(), arg0.getY(),
    Manipulator.NEG_MANIPULATOR_NODEMASK);
if (hits.containsIntersections()) {
    Intersection hit = polytopeHits.getFirstIntersection();
    ...
}
```

In the same way we are able to calculate intersections inside a polytope traced from the view point.

```
Intersections polytopeHits = getCanvas3D().getOSGViewer().rayPick(
    _manager, arg0.getX(), arg0.getY(),
    Manipulator.NEG_MANIPULATOR_NODEMASK);
if (polytopeHits.containsIntersections()) {
    Intersection hit = polytopeHits.getFirstIntersection();
    ...
}
```

Printing utilities

For High Resolution Printing we decided to take several tile images in memory and then build a collage beginning from this tile images. The way to do that is using an `OffscreenViewer`.

```
printViewer = new OSGViewer();
printViewer.setUpViewerInBackground(0, 0, getCanvas3D().getWidth(),
    getCanvas3D().getHeight());
printViewer.setSceneData(cow.osg);
```

The first step was create a tiled view of the scene. Then we have to take a image of this tiles in memory.

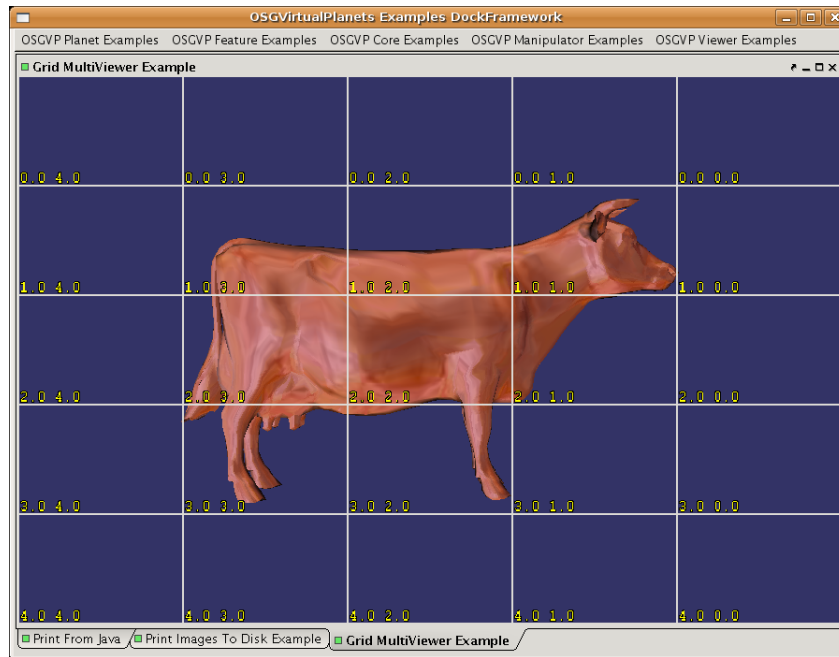


Figure 15: Tiled scene.

The class `PrintUtilities` makes for us this work, we can decide which is the size in pixels of final image. This image is converted to a `BufferedImage` in Java.

```
PrintUtilities util = new PrintUtilities();
util.setViewer(printViewer);
BufferedImage s = util.getHighResolutionImage(g, viewerCam,
        5600, 2700);
```

As you can see, you can create big images in memory and then convert it to Java images, following a tiling process all done in background.

OSGVP Core

The following section is dedicated to explain how the osgVP-core library works. In particular, adding or removing nodes to the graph, positioning or updating a node, loading and applying textures or materials, etc. The osgVP-core library also contains mathematical tools necessary to manage the scene-graph.

Osg features

The org.gvsig.osgvp.core.osg includes the necessary classes to create and manages OSG scene-graph in Java applications. Most of the basic OSG features are included in the library.

Managing the scene-graph

Adding and/or removing nodes to the scene should be easy if you know the basics of scene-graphs. Usually the root of the scene is a Group node. This object is capable to add or remove nodes to itself by a simple call. Following lines of code tries to illustrate the process.

```
Group root = new Group(); // creates the root of the scene

try {
    Node dummy = new Node(); // creates a dummy node
    root.addChild(dummy); //Adds node to root group
}
```

```
} catch (NodeException e) {
    Util.logger.severe(e.toString());
    return null;
}
```

In the same way you can remove nodes from groups.

```
try {
    root.removeChild(dummy);
} catch (NodeException e) {
    Util.logger.severe(e.toString());
    return null;
}
```

The rest of the public interface of Nodes and Groups are easy to manage. The name of the methods are quite illustrative for people who have the necessary know-how of scene-graphs. For more information you can look up in the Java-docs.

Mathematic Tools

Mathematic tools are included in this library. They are necessary to manage some behaviours of nodes in the scene, like positioning, rotating, scaling, etc. You can do this through Vector, Matrix or Quaternion operations. Using this entities depends on your maths skills. The public Api of these classes is extensive and lets the user to do almost any operation.

The defined mathematics classes are:

- Vec2, Vec3 and Vec4: Usually Vec3 are used to define vertex positions and Vec4 to define colors. Since Java 1.5 you can use `Vector[]`, this is the way to create arrays of Vecs.
- Matrix: Everybody who works in CG knows the importance of matrices in the view process. Matrix operations could serve for changing the perspective or the projection of the view, etc. For more information you can consult any manual of OpenGL.
- Quat: This entity is used to accumulate rotations. It could be used in spherical interpolations, etc. The use of this class requires some advanced maths skills.

- Ellipsoid Model: Serves to do changes between coordinate systems using an ellipsoid model to do the operations.

Positioning a Node

Once we know how to use Maths tools it's quite easy to implement affine transformations. It can be done using the structures PositionAttitudeTransform. Indeed this transformations, the class Autotransform could be used to implement Billboarding or auto-scaling to screen.

Let's see some code:

```
Group root= new Group();

//create node
AutoTransform at = new AutoTransform();
at.addChild(osgDB.readNodeFromFileFromResources("/cow.ive"));

//affine transformations
at.setPosition(new Vec3(0, 0, -2));
at.setScale(new Vec3(0.8, 1, 0.8));
at.setRotation(90, new Vec3(0, 0, 1));

//add transform to the scene
root.addChild(at);
```

If we want to do billboarding we must write this line:

```
at.setAutoRotateMode(AutoTransform.AutoRotateMode.ROTATE_TO_CAMERA);
```

A full implementation of this example is available on "Camera Matrices Example" of the examples framework. See the CameraMatricesExample class in examples package of source code.

Geodes, Drawables & Primitives

A brief explanation of the following classes is helpful:

-Geodes: The geode class is derived from the 'node' class. Nodes (and thus geodes) can be added to a scene graph as leaf nodes. Geode instances can have any number of 'drawables' associated with them.

-Drawables: The base class 'drawable' is a pure virtual class with six concrete derived classes. (reference) The 'geometry' class can have vertex (and vertex data) associated with it directly, or can have any number of 'primitiveSet' instances associated with it. Vertex and vertex attribute data (color, normals, texture coordinates) is stored in arrays. Since more than one vertex may share the same color, normal or texture coordinate, and array of indexes can be used to map vertex arrays to color, normal or texture coordinate arrays.

-PrimitiveSet: This class loosely wraps the OpenGL drawing primitives - POINTS, LINES, LINE_STRIP, LINE_LOOP,... QUADS,... POLYGON.

Geometry creation

The following section of code sets up a viewer to see the scene we create, a 'group' instance to serve as the root of the scene graph, a geometry node (geode) to collect drawables, and a geometry instance to associate vertexes and vertex data. The following code is from "Geometry Example" of the examples framework.

```
//Creation of Instances
Geometry geometry = new Geometry();
Geode g = new Geode();

// Arrays of normals vertexes and colors
Vector<Vec3> vertices = new Vector<Vec3>();
Vector<Vec4> color = new Vector<Vec4>();
Vector<Vec3> normal = new Vector<Vec3>();

//Fill in the vertex array
vertices.add(new Vec3(-1.02168, -2.15188e-09, 0.885735));
vertices.add(new Vec3(-0.976368, -2.15188e-09, 0.832179));
vertices.add(new Vec3(-0.873376, 9.18133e-09, 0.832179));
vertices.add(new Vec3(-0.836299, -2.15188e-09, 0.885735));
vertices.add(new Vec3(-0.790982, 9.18133e-09, 0.959889));
//Normal array
normal.add(new Vec3(0.0, -1.0, 0.0));
```

```
//Color array
color.add(new Vec4(1.0f, 1.0f, 0.0f, 1.0f));

//Setting the arrays into geometry
geometry.setVertexArray(vertices);
geometry.setColorArray(color);
//The value in the first element of the color array
//will be used in all the vertices
geometry.setColorBinding(Geometry.AttributeBinding.BIND_OVERALL);
geometry.setNormalArray(normal);
//The value in the first element of the normal array
//will be used in all the vertices
geometry.setNormalBinding(Geometry.AttributeBinding.BIND_OVERALL);

try{
//We can decide the primitive to draw the geometry
geometry.addPrimitiveSet(new DrawArrays(DrawArrays.Mode.POINTS, 0,
                                         geometry.getVertexArray().size()));

...
}
```

If you want to use textures you must define the texture coordinates array. For dig in PrimitiveSets and Geometries you can take a look to the OpenSceneGraph website.

Using LODs

In the following code, we show how to use a LOD to change the level of detail of the geometry of the scene-graph. In this example we use three different geometries in different distance ranges to represent a figure. The full example is available on “LOD Example” of the examples framework. See the LODExample class in examples package of source code.

```
Group g = new Group();

try {

    LOD lod = new LOD();
    g.addChild(lod);
}
```

```

lod.addChild(osgDB.readNodeFromFileFromResources
            ("/cow.ive"), 0, 1000);
lod.addChild(osgDB.readNodeFromFileFromResources
            ("/cessna.osg"), 1000, 2000);
lod.addChild(osgDB.readNodeFromFileFromResources
            ("/MachineGun.ive"), 2000, 50000);

} catch (NodeException e) {
}

```

StateSets

A scene graph manager traverses a scene graph to determine what geometry needs to be sent to the graphics pipeline for rendering. During this traversal, the scene graph manager can also collect information on how that geometry should be rendered. This information is stored in `osg::StateSet` instances. StateSets contain lists of OpenGL attribute/value pairs. These StateSets can be associated with nodes of the scene-graph. During pre-render traversal, StateSets are accumulated from the root to leaf nodes. State attributes that are not changed at a node are simply inherited from above.

A few additional features allow more control and flexibility. A state's attribute value can be set to `OVERRIDE`. This means that all the children of that node - regardless of what the children's attribute value is - will inherit the parent node's attribute value. This `OVERRIDE` can be, well, over-ridden. If one of the child nodes set that attributes value to `PROTECTED`, they can set this attribute value regardless of the parent's setting.

With StateSets you can switch the lighting mode, or activate blending, fog, texture and material modes, etc. The normal way to do that is creating or getting the StateSet of a node and then activate the mode you want. In the example of code below Material mode is activated.

```

g = new Group();
//create the stateset
StateSet st= g.getOrCreateStateSet();
//activating Material mode
Material m = new Material();
st.setMaterial(m, Node.Mode.ON);

```

For more information about StateSet modes you can look up in the OpenScene-Graph website or in the Java-doc of osgVP.

Textures and Materials

As in OpenGL, you can apply textures or materials to an object in the scene. The way to do that is activating the desired stateset mode associated to the object as explained before. In case of materials you can define it as in OpenGL, let's see an example:

```
Sphere sphere = new Sphere();
//create material
Material m = new Material();
    try {
        //settings
        m.setDiffuse(Material.Face.FRONT, new Vec4(1.0,
            (float) i / 10.0f, 0.6f, 1.0f));
        m.setAmbient(Material.Face.FRONT, new Vec4(0.9f,
            0.8f, 0.6f, 1.0f));
        m.setSpecular(Material.Face.FRONT, new Vec4(0.9f,
            0.8f, 0.6f, 1.0f));
        m.setEmission(Material.Face.FRONT_AND_BACK,
            new Vec4(1, 0, 0, 1));
        m.setShininess(Material.Face.FRONT, 85);
        m.setTransparency(Material.Face.FRONT, (float) i
            / 15.0f);
        //apply material
        sphere.getOrCreateStateSet().setMaterial(m,
            Node.Mode.ON);
        ...
    }
```

If you want to use textures, the object receiver of these textures must have the TexCoord vector defined. If TexCoord vector is not defined, the texture will not be applied correctly. To load Textures the library osgVP-core offers the classes Texture2D and Image. The procedure to activate the corresponding stateset is very similar to the material case.

```
try{
```

```
Sphere sphere = new Sphere();
Texture2D tex = new Texture2D();
//activates the texture mode in stage 0
sphera.getOrCreateStateSet().setTexture2D(tex, 0,
    Node.Mode.ON);
.....
```

Once you have activated the corresponding stateset mode, now you have to load one or several images in the texture instance.

A full implementation of this example is available on “Material Example” of the examples framework. See the `MaterialExample` class in `examples` package of source code.

Loading images

There are two basic ways to load images through the public interface of the class `Image`. First way is loading images of a `.jpg` `.bmp` or whatever kind of image file supported by `OpenSceneGraph`. The other way is using `BufferedImage` of Java. The class `image` is prepared to convert images from `BufferedImage`s to `osgVP` images. Coding first way should look like next lines.

```
Texture2D tex = new Texture2D();

try {
    //We have image file in a resources folder
    File texture = Util.extractFromURL(this.getClass()
        .getResource("/test.jpg"));

    //load the image
    Image im = new Image(texture.getPath());

    //setting the image in the texture instance
    tex.setImage(im);

} catch (Exception e) {
}
```


If you want to load textures using `BufferedImages` you can implement a similar code like the written below.

```
Texture2D tex = new Texture2D();

try{

    Image im = new Image();

    //charging the imagefile in a BufferedImage
    BufferedImage bufferim = ImageIO.read(new File(
        Util.extractFromURL(this.getClass().
            getResource("/planet.png")).getAbsolutePath()));

    im.setBufferedImage(bufferim);
    tex.setImage(im);

}
```

In a similar way you can convert images from `osgVP` to `BufferedImages` of Java.

```
try {

    File texture = Util.extractFromURL(this.getClass().
        getResource("/earth.gif"));

    Image im = new Image(texture.getAbsolutePath());

    BufferedImage bufferim = im.getBufferedImage();
} catch (Exception e) {
}
```

A full implementation of this example is available on “`Buffered Image Example`” and “`Image Example`” of the examples framework. See the `BufferedImageExample` and `ImageExample` class in examples package of source code.

Updating a Node

Updating a Node is a very usual process in CG applications. The scene-graph gives us the way to do it. Your main class must implement the UpdateNodeListener Interface. Implementing this interface forces you to write the update method. Is in this method where you have to do whatever you want with your node. You can take a look to the example AxisExample of examples framework. Let's see how it works.

```
public class AxisExample extends AbstractCoreExample
    implements UpdateNodeListener {
    ...

    //declare the axis
    Axis ejes = new Axis();
    //Attention!! You must set the Listener
    ejes.setUpdateListener(this);

    ...

    //Do updating stuff
    public void update(Node node) {
        ejes.update(getCanvas3D().getOSGViewer()
            .getCamera());
    }
}
```

A full implementation of this example is available on "Axis Example" of the examples framework. See the AxisExmample class in examples package of source code.

GLSL Programming

OpenGL Shading Language allows programmers to write custom pixel and vertex shaders. For more information on shading languages - including minimum hardware and software requirements - see www.opengl.org. The classes Program and Shader allow users to apply these shaders as part of a StateSet to selected subtrees within a scene graph. In this manual we explain nothing about writing shaders, but we explain how to apply these shaders to a Node of our scene-graph. Using a custom vertex or fragment shader in osgVP involves the following basic classes:

- Program - application level abstraction of the OpenGL Shading Language `glProgramObject`. Instance of the Program class can be associated with StateSets and enabled using the StateSet class. Enabling a program object for a stateset results in drawables within that stateset being rendered using the Program's shaders.
- Shader - application level abstraction of the OpenGL Shading Language `glShaderObject`. This class manages loading and compiling shader source code. Instances of the Shader class can be assigned to one or more Program instances. There are two types of shader objects: `Shader.Type.FRAGMENT` and `Shader.Type.VERTEX`.

The steps to create an application that uses an OpenGL pixel and fragment shader are as follows:

- Create a Program instance
- Create VERTEX and FRAGMENT instances of the Shader class
- Load and compile the shader source
- Add the shaders to the Program instance
- Associate and enable the Program instance as part of a StateSet.

The code implemented in the GLSL Program Example of examples framework, available at `ProgramExample` class, looks like this:

```
//create the instances
Program prog= new Program();
Shader shad = new Shader();
Shader shadfrag = new Shader();
//set type of shader
shad.setType(Shader.Type.VERTEX);
shadfrag.setType(Shader.Type.FRAGMENT);
//Load and compile the shader source
File source = Util.extractFromURL(this.getClass().getResource(
    "/marble.vert"));
File sourcefrag = Util.extractFromURL(this.getClass().getResource(
    "/marble.frag"));
shad.loadShaderSourceFromFile(source.getPath());
shadfrag.loadShaderSourceFromFile(sourcefrag.getPath());
//adding shaders to program instance
prog.addShader(shad);
```

```
prog.addShader(shadfrag);  
//activate the stateset  
mynode.getOrCreateStateSet().setProgram(prog, Node.Mode.ON);
```

Loading and saving scenes

The osgVP-core library is capable to load and save scenes or nodes in the same format as OpenSceneGraph does it using the package org.gvsig.osgvp.core.osgdb. It includes .osg .ive .3ds or/and any format supported in the plugins of OpenSceneGraph. The way to load/save nodes or scenes is using osgDB class.

In the next piece of code a Node is loaded from disk, and added to the root of scene.

```
public Node createScene() {  
    root = new Group();  
    try {  
        g.addChild(osgDB.readNodeFileFromResources("/cow.ive"));  
    } catch (NodeException e1) {  
        e1.printStackTrace();  
    }  
    return root;  
}
```

A full implementation of this example is available on “Update Node Example” of the examples framework. See the UpdateNodeExample class in examples package of source code.

In the same way you should be able to persist nodes using de call

```
osgDB.WriteNodeFile("/cow.ive");
```

A full implementation of this example is available on “Save Test Example” of the examples framework. See the SaveTestExample class in examples package of source code.

Lighting

The `org.gvsig.osgvp.core.osgsim` package contains the `LightPoint` and `LightPointNode` classes to define light points into the scene-graph as shown in the next example.

```
LightPoint lp = new LightPoint();
LightPointNode lpn= new LightPointNode();

lp.setColor(new Vec4(0f,1f,1f,0.5f));
lp.setPosition(new Vec3(0f,0f,0f));
lp.setRadius(200);

lpn.addLightPoint(lp);
```

A full implementation of this example is available on “LightPoint Example” of the examples framework. See the `LightPointExample` class in `exampl` package of source code.

Adding Text

This `org.gvsig.osgvp.core.osgtext` package has the necessary classes to rendering text. The text base class has the necessary parameters to define a text into a scene-graph. A text can be rendered like a 2D text, using `Text` class, or like a 3D text, using `Text3D`. Both classes inherit their methods from `TextBase` class. The `FadeText` class is an extension of `Text` class which includes fade in and fade out functionality.

Text3D

To add a 3D Text into the scene-graph you can follow the next example:

```
Geode geode = new Geode();
Text3D t3d = new Text3D();

t3d.setFont("fonts/dirtydoz.ttf");
t3d.setText("3D Text");
```

```
t3d.setCharacterSize(0.5f);
t3d.setCharacterDepth(1.5f);
t3d.setPosition(new Vec3(50 + (i * 5), j * 5, k * 5));
t3d.setDrawMode(TextBase.DrawModeMask.TEXT);
t3d.setAlignment(TextBase.AxisAlignment.SCREEN);
t3d.setRenderMode(Text3D.RenderMode.PER_GLYPH);

geode.addDrawable(t3d);
```

Text

The example shows how to draw a 2D text:

```
Geode geode = new Geode();
Text t = new Text();

t.setFont("fonts/arial.ttf");
t.setText("2D Text");
t.setAutoRotateToScreen(true);
t.setCharacterSize(1);
t.setPosition(new Vec3(25 + (i * 5), j * 5, k * 5));
t.setColor(new Vec4(1f, 0f, 0f, 1f));

geode.addDrawable(t);
```

FadeText

You can use FadeText instead of normal Text in your scene-graph like next example:

```
Geode geode = new Geode();
FadeText ft = new FadeText();

ft.setFont("fonts/arial.ttf");
ft.setText("Fade Text");
ft.setAutoRotateToScreen(true);
ft.setCharacterSize(1);
ft.setPosition(new Vec3(25 + (i * 5), j * 5, k * 5));
```

```
ft.setColor(new Vec4(1f, 0f, 0f, 1f));

geode.addDrawable(ft);
```

A full implementation of this example is available on “Text Example” of the examples framework. See the `TextExample` class in `examples` package of source code.

Utilities

This `org.gvsig.osgvp.core.osgutil` package contains some helpful classes imported from OSG to improve the rendering efficiency like tessellator and optimizer. Other utilities are available at `org.gvsig.osgvp.core.util` package. This package includes some helpful classes like `ActionCommand`, `EventHandler` or `UpdateNodeListener` used along the `osgVP` libraries to implement some functionality. The `NativeDeps` class which load the native libraries of `osgVP` its also included in this package. Finally we include a camera HUD, a geometry extruder and show normals utilities too.

Tessellator

The tessellator technique included in `osgVP` can divide a polygon in triangles for rendering without holes, including concave polygons. This technique can change the winding rule of tessellation (see the *Red Book of OpenGL*, ch. 11) and determine if the triangles resulting are stored in a unique geometry object or in different geometries.

Next example, shows how to tessellate a polygon:

```
Tessellator t = new Tessellator();

t.setTessellationType(Tessellator.TessellationType
    .TESS_TYPE_GEOMETRY);
t.setWindingType(Tessellator.WindingType
    .TESS_WINDING_ODD);
t.retessellatePolygons(geometry);
```

Optimizer

This class traverse the scene-graph to improve rendering efficiency. There is several options imported from OSG optimizer class. See the OSG documentation before to change this options. The next example optimizes a geometry converting it into a triangle strip.

```
Optimizer opt = new Optimizer();

try {
    opt.optimize(geometry,
        Optimizer.OptimizationOptions.TRISTRIP_GEOMETRY);
} catch (NodeException e1) {
    e1.printStackTrace();
}
```

Camera HUD

Exports the OSG camera HUD functionality to be used in Java side. The following code shows how to create a camera HUD and add a axis node into it. A full implementation of this example is available on “Axis Example” of the examples framework. See the AxisExample class in examples package of source code.

```
CameraHUD hud = new CameraHUD();
Group root = new Group();
Axis ejes = new Axis();

root.addChild(hud);
hud.addChild(ejes);
```

Normals

This class draw the normals of a given geometry or node of the scene-graph. To use this class, follow the next code:

```
Group g = new Group();
try {
    g.addChild(new Normals(osgDB.readNodeFileFromResources
```



```
        ("/cow.ive"), 1, Normals.Mode.VERTEX));  
        g.addChild(osgDB.readNodeFromFileFromResources("/cow.ive"));  
    } catch (NodeException e1) {  
    }  
}
```

A full implementation of this example is available on “Basic Viewer Example” of the examples framework. See the `BasicViewerExample` class in `examples` package of source code.

OSGExtruder

Our `osgVP` libraries include a geometry extruder utility that can be used as shown in the following code:

```
Group g = new Group();  
Geode ge = new Geode();  
Vector<Vec3> shape = new Vector<Vec3>();  
  
shape.add(new Vec3(0, 0, 0));  
shape.add(new Vec3(0, 0, 5));  
shape.add(new Vec3(0, 5, 5));  
shape.add(new Vec3(0, 5, 0));  
  
OSGExtruder extruder = new OSGExtruder();  
extruder.start();  
for (int i = shape.size() - 1; i >= 0; i--)  
    extruder.addPoint3D(shape.get(i));  
  
Vec3 result = shape.get(1).crossProduct(shape.get(3));  
extruder.extrude(result);  
  
extruder.finish();  
  
ge.addDrawable(extruder.getGeometry());  
g.addChild(ge);
```

A full implementation of this example is available on “Extrusion Example” of the examples framework. See the `ExtrusionExample` class in `example’s` package of source

code.

A specific implementations of the `OSGExtruder` for points, polylines and polygons are available in the package too.

OSGVP Terrain

In this section, you will learn how to build planets and terrains using `osgVP-terrain` library and how to visualize and manipulate them in your own application. Next, we present the layer management for textures and elevations as well as the editable layer properties.

The Terrain View

Since a terrain is a node into the scene graph, you can use a viewer of `osgVP-viewer` library or a default viewer of `OpenSceneGraph` to visualize it. But the terrain special characteristics like the ellipsoidal geometry, the coordinate system or the huge size of the terrain, forces to use a special viewer adapted to the terrain needs.

There is a specific *TerrainViewer* class inside the `osgVP-terrain` library. This viewer inherit his methods of the *OSGViewer* class defined in the `osgVP-viewer` library and re-implements the scene graph cull visitor to solve some issues with the Z-buffer. To use this terrain viewer for visualization is highly recommended instead of the other `OpenSceneGraph` based viewers.

Create a terrain viewer

The procedure to create a terrain viewer is similar to the procedure to create an *OS-Viewer*. First of all, you must create a new *IViewerContainer* variable to access the canvas and viewer properties.

```
private static IViewerContainer _canvas3d;
```

Later the *TerrainViewer* instance can be created and assigned to the canvas. The first parameter of the *createView* method specifies the viewer type. A viewer can be a *CANVAS_VIEWER* or a *JPANEL_VIEWER* type. In the second parameter of the method you have to assign the recently created terrain viewer. A terrain viewer can be added into a *JPanel* and integrated in your application.

```
JPanel jContentPane = new JPanel();
jContentPane.setLayout(new BorderLayout());

TerrainViewer planetViewer = new TerrainViewer();

_canvas3d = ViewerFactory.getInstance().createView(
    ViewerFactory.VIEWER_TYPE.CANVAS_VIEWER,
    terrainViewer);

jContentPane.add((Component)_canvas3d, BorderLayout.CENTER);
ViewerFactory.getInstance().startAnimator();
```

When a terrain viewer is created, three new nodes are created and added to the scene graph. A *osgVP-viewer* viewer only has a method called *setSceneData* to add nodes into the scene graph, but in the terrain viewer the scene data is composed by three void nodes: **terrain node**, **features node** and **camera hud node** with their set and get methods. The methods *setSceneData* and *getSceneData* are still available in the terrain viewer but is recommended don't use them because you have to keep the structure of the scene graph. A default special manipulator for terrain navigation called *TerrainCameraManipulator* is added to the viewer. We will analyze the methods for add nodes to scene graph and how to use a manipulator in depth in the next sections.

We would like to remind you that a terrain viewer inherits his methods from the *OSGViewer* class defined in the library *osgVP-viewer*. You can use this methods to modify the viewer properties. Before your application finalize you must call the *dispose* method of the viewer and stop the animator.

```
jFrame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        ViewerFactory.getInstance().stopAnimator();
        _canvas3d.dispose();
    }
});
```

```
});
```

Set the scene data in a terrain viewer

The scene data in a terrain viewer must be distributed in three different nodes according to the type of visualization of the node. All terrain nodes must be included in the terrain node of the scene data because the terrain viewer needs this nodes to compute some intersections with them. This intersections are used in the terrain manipulator of the viewer and to compute the near and the far planes of the view.

The nodes added in the features group aren't used in the compute of the intersections for manipulate the view. You can use this node to add some 3D geometry in the surface of your terrain.

Finally the camera hud node let you to add some nodes in the display that are always visible like text or images with information useful for the users.

You can add some nodes into the scene graph to test your new viewer. For example, you can create a new terrain with the default constructor (it builds the Earth) and add to the scene graph.

```
Terrain earth = new Terrain();
terrainViewer.addTerrain(earth);
```

By default, the position of the camera when you build a viewer is inside the planet. You must move the camera to a new position for view the whole planet.

```
Camera cam = new Camera();
cam.setViewByLookAt(earth.getRadiusEquatorial() * 5.0, 0,
    0, 0, 0, 0, 0, 0, 1);
terrainViewer.setCamera(cam);
```

Now, we can put some 3D models in the terrain surface. For example, you can put a OpenSceneGraph model called *cow.ive* in the North Pole. Adding a matrix transform to the model let you set the scale and the position in the scene graph.

```
double factor = earth.getRadiusEquatorial() / 20.0;
```

```
PositionAttitudeTransform transform =
    new PositionAttitudeTransform();
```

```
transform.addChild(osgDB.readNodeFromFileFromResources("/cow.ive"));
transform.setScale(new Vec3(factor, factor, factor));
transform.setPosition(new Vec3(0, 0, earth.getRadiusPolar()*1.1));

terrainViewer.addFeature(transform);
```

Finally you can add some information always visible in the screen. For example, you can add some text in the HUD node.

```
Text info = new Text();
info.setText("Terrain View Example");
terrainViewer.addNodeToCameraHUD(info);
```

Using camera manipulators

A camera manipulator define how to move the camera in the scene. By default, a `TerrainViewer` set a *CustomTerrainManipulator*. This camera manipulator is specific for terrain navigation. It defines three basic movements: **Zoom**, **Azimuth** and **Roll**. The first one, let you to move closer or away from the terrain surface. The second one, let you to change the angle of inclination and the last let you move around the terrain surface.

By default, there are a combination of button mouse and keys for each movement. You can add more keys and mouse combinations for movement with the method *registryActionMask* of the manipulator, where the first parameter indicates the movement type, the second one the mouse mask and the last the key mask associated. For example, if you want to add a new combination of the left mouse button and the **a** key for change the **Azimuth** you can use the following line:

```
manip.registryActionMask("AZIMUT",
    MouseButtonMaskType.LEFT_MOUSE_BUTTON, 'a');
```

For remove the last combination you can use the *unregistry* method:

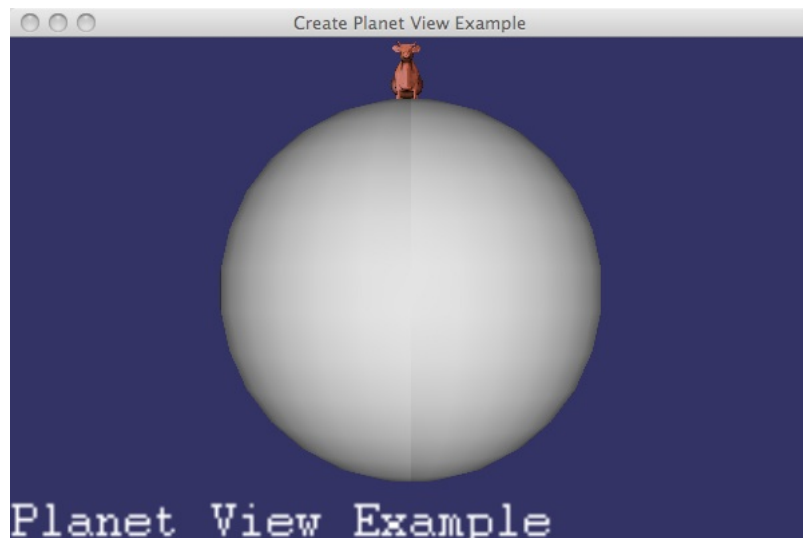
```
manip.unRegistryActionMask("AZIMUT",
    MouseButtonMaskType.LEFT_MOUSE_BUTTON, 'a');
```

Aslo you can change the speed of the movement for the **Zoom** and **Roll** actions with the methods *setRollFactor* and *setZoomFactor*. Furthermore, there are set and get

methods for specify the minimum and maximum distance of the camera from the center of the planet: *setMinimumDistance*, *setMaximumDistance*, *getMinimumDistance* and *getMaximumDistance*. Finally, you have some methods to force the orientation of the planet to the North: *setEnabledNorthOrientation* or *forceNorthOrientation*.

```
manip.setMinimumDistance((float) earth.getRadiusEquatorial());  
manip.setMaximumDistance((float) earth.getRadiusEquatorial()*5.0f);  
manip.setEnabledNorthOrientation(true);
```

This sample code is available in the package `UserGuide` of `osgvp-examples` and should look like the following image.



Define a terrain

When you create a terrain with the default constructor it build an ellipsoidal Earth planet in a cartesian coordinate system. But you can specify some parameters in the constructor to create a lot of different terrains.

When you would define a terrain you can specify his name in the first parameter of the constructor. The second one specify the type of the coordinate system, it could be **Geocentric** for ellipsoidal terrains in cartesian coordinate system, **Geographic** for plane terrains in cartesian coordinate systems and **Projected** for plane terrains in cartesian or UTM coordinate systems. The third parameter define the format of the next

parameter (the coordinate system name) and usually is set to *WKT*. The next parameter is the coordinate system name, in a **Geocentric** or **Geographic** terrain is usually set to "EPSG:4326" and in a **Projected** terrain it must be set in **UTM** coordinates like "EPSG:23030". After the coordinate system name you have to set four parameters that indicates the extent of the terrain (always in the units of the coordinate system specified in the coordinate system name). Finally the two last parameters are the equatorial and the polar radius of the planet respectively.

```
Terrain mars = new Terrain("Mars",
    CoordinateSystemType.GEOCENTRIC, "WKT", "EPSG:4326",
    -180.0, -90.0, 180.0, 90.0, 3396200, 3376200)
```

Alternatively, you can define a default terrain and use the set and get methods to change some parameters. The following code is equivalent to the code of the last paragraph.

```
Terrain mars = new Terrain();

mars.setTerrainName("Mars");
mars.setCoordinateSystemType(CoordinateSystemType.GEOCENTRIC);
mars.setCoordinateSystemFormat("WKT");
mars.setCoordinateSystemName("EPSG:4326");
mars.setExtent(-180.0, -90.0, 180.0, 90.0);
mars.setRadiusEquatorial(3396200);
mars.setRadiusPolar(3376200);
```

Using this methods you can create a lot of different planets. For example, you can create a projected view of the autonomous region of Valencia. First you must create a terrain viewer with the methods explained in the previous section. Then you can add a new projected terrain specifying the dimensions in the appropriate coordinate system and add to the terrain viewer.

```
private static Terrain _terrain;

_terrain = new Terrain("Valencia", Terrain.CoordinateSystemType
    .PROJECTED, "WKT", "EPSG:23030", 0, 4000000, 1000000,
    5000000, 6378137.0, 6356752.3142);
terrainViewer.addTerrain(_terrain)
```


Don't forget to put the camera in the correct position to see all the terrain.

```
double difx = (_terrain.getExtent().xMax()
    -_terrain.getExtent().xMin()) / 2.0d;
double dify = (_terrain.getExtent().yMax()
    -_terrain.getExtent().yMin()) / 2.0d;
double posx = _terrain.getExtent().xMin() + difx;
double posy = _terrain.getExtent().yMin() + dify;
double height = Math.sqrt(difx * difx + dify * dify) * 4.0f;

Camera cam = new Camera();
cam.setViewByLookAt((float) posx, (float) posy, (float) height,
    (float) posx, (float) posy, 0f, 0f, 1f, 0f);
terrainViewer.setCamera(cam);
```

All of the explained methods can be changed in real time but you have to keep in mind the correlation between the coordinate system and the units of the extent. For example, you can add a key listener to the example for change the size of the terrain extent.

```
_canvas3d.addKeyListener(new KeyListener() {
    public void keyPressed(KeyEvent e) {
        if (e.getKeyCode() == KeyEvent.VK_1) {
            _terrain.setExtent(0, 4000000,
                1000000, 5000000);
        } else if (e.getKeyCode() == KeyEvent.VK_2) {
            _terrain.setExtent(0, 4000000,
                2000000, 5000000);
        }
    }
});
```

Finally, this sample code is available in the package `UserGuide` of `osgvp-examples`. Your example should look like the figure 4.1 when the key 1 is pressed and like the figure 4.2 when the key 2 is pressed.

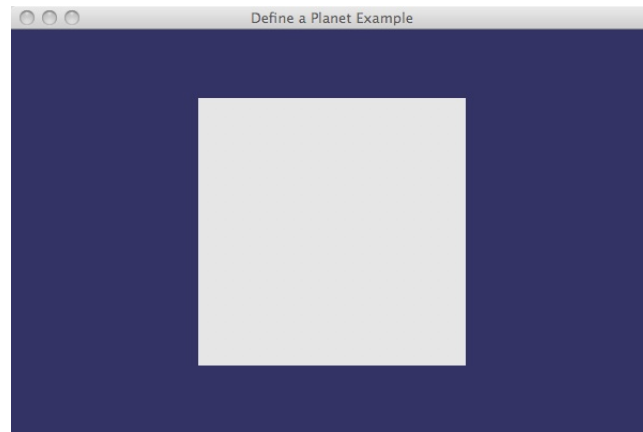


Figure 16: Projected terrain sample with the extent of the autonomous region of Valencia.

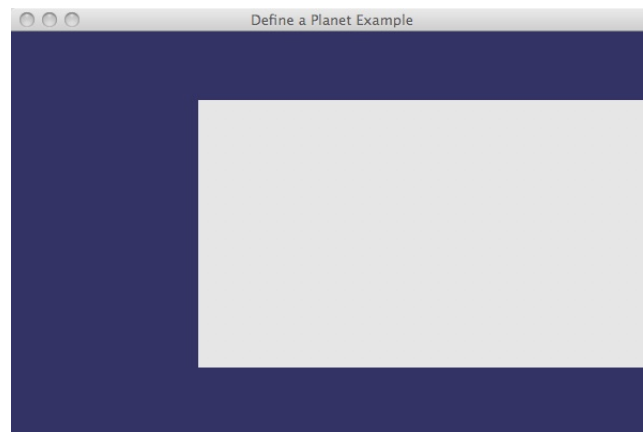


Figure 17: Projected terrain sample after change the size of his extent.

Layer management

The class *LayerManager* give you the necessary methods to afford the layer management. You can add raster layers like textures or elevations in your terrain. One *LayerManager* can be shared by two different terrains but in this case any change of layer parameters is reflected in both terrains.

Adding a layer manager

The method `setLayerManager` of the `Terrain` class, allow to set a layer manager who manages all of the terrain layers.

```
LayerManager manager = new LayerManager();
terrain.setLayerManager(manager);
```

Adding layers

To add a raster layer into the layer manager, we need to use the class `RasterLayer` and set the necessary parameters like the layer extent. After this, we can add this layer to the layer manager and it will be applied to the terrain automatically. We also can modify the layer parameters in any time.

```
RasterLayer layer0 = new RasterLayer();

layer0.setNodeName("Layer 0");
layer0.setExtent(new Extent(Math.toRadians(-180.0),
    Math.toRadians(-90.0),
    Math.toRadians(180.0),
    Math.toRadians(90.0)));
layer0.setEnabled(true);
layer0.setOpacity(1.0f);

manager.addLayer(layer0);
```

Also, we can add a `HeightfieldLayer` for elevation data:

```
Heightfield layer1 = new Heightfield();

layer1.setNodeName("Layer 1");
layer1.setExtent(new Extent(Math.toRadians(-180.0),
    Math.toRadians(-90.0),
    Math.toRadians(180.0),
    Math.toRadians(90.0)));
layer1.setEnabled(true);
layer1.setVerticalExaggeration(1.0f);
```

```
manager.addLayer(layer1);
```

And finally, add a `VectorLayer` to manage features with multiresolution:

```
VectorLayer layer2 = new VectorLayer();

layer2.setNodeName("Layer 2");
layer2.setExtent(new Extent(Math.toRadians(-180.0),
    Math.toRadians(-90.0),
    Math.toRadians(180.0),
    Math.toRadians(90.0)));
layer2.setEnabled(true);
layer2.setDensity(1.0f);

manager.addLayer(layer2);
```

By default when a layer is added, it is pushed at the end of the list (top on visualization) but the method `addLayer` lets us to give a specific order.

```
manager.addLayer(layer2, 3);
```

Creating data drivers

Each layer needs to retrieve data from a source. The class that manages each request and how to get the necessary data from the source is called `DataDriver`. By the way, the same data driver can retrieve data for one or more layers, and get it from one or more sources, depending on the implementation. In the case of Java applications, `osgVP-terrain` offers a `JavaDataDriver` class which exports the necessary functionality for retrieving data from Java sources. Each data driver has a `DataLoader` interface where the user can process the retrieving events and get the data from any source.

The `JavaDataDriver` needs to inject the data using the OSG viewer update traverse. If we don't add the `JavaDataDriver` to the same viewer of the terrain, the data never arrives to the tiles of the terrain.

```
// Data driver
JavaDataDriver jdd = new JavaDataDriver();
_canvas3d.getOSGViewer().addUpdateOperation(jdd);
```

After add the driver to our viewer, we need to implement the data loader interface, and define which is our data source depending of the parameters of the request event. Notice, in this example we return diferent data for each type of layer, but other interesting parameters like the layer identifier or the extent are available to improve the functionality of the driver and retrieve more specifically data from source.

```
// Set the data loader
jdd.setDataLoader(new DataLoader() {
    public UpdateDataEvent loadData(RequestDataEvent rde)
    {
        /// Build the update event
        UpdateDataEvent ude = new UpdateDataEvent();
        ude.copyDataFromRequest(rde);

        if(rde.getLayer().getLayerType()
            == Layer.LayerType.HEIGHTFIELDLAYER)
        {
            ude.setHeightfieldData(_elevation.getPath(),
                "tiff");
        }

        if(rde.getLayer().getLayerType()
            == Layer.LayerType.RASTERLAYER)
        {
            ude.setRasterData(_texture.getPath(),
                "jpg");
        }

        if(rde.getLayer().getLayerType()
            == Layer.LayerType.VECTORLAYER)
        {
            ude.setVectorData("cow.ive", "ive");
        }

        return ude;
    }
});
```

Finally we can add our driver to each layer. Don't forget to associate a driver for each layer because it defines how to retrieve the data. We are going to use the same driver for each layer in this example. Of course, you can define different drivers instead of only one.

```
layer0.setDataDriver(jdd);  
layer1.setDataDriver(jdd);  
layer2.setDataDriver(jdd);
```

Removing layers

When a layer is no longer used you can remove from layer manager.

```
manager.removeLayer(layer0);
```

We can remove a layer given its order too.

```
manager.removeLayer(3);
```

Reorder layers

The manager let us move layers to a different position using the `moveLayer` method and specifying the old position and the new ones.

```
int oldposition = 0;  
int newposition = 3;
```

Get a layer

Since we can add the same layer in different positions (without create a new layer), we can retrieve the current layer positions in the layer manager using the following method:

```
Vector<Integer> orders = manager.getOrder(layer0);
```

Also we can get a layer given a current position:

```
Layer layer = manager.getLayer(0);
```

Forcing to retrieve data

When some of the data isn't displayed correctly, we can force to retrieve all the data again for a layer using the method **invalidateLayer**:

```
manager.invalidateLayer(layer);
```

Changing layer properties

All layers has general properties like the one that indicates if a layer is enabled or disabled. You can change a layer to disabled to avoid retrieving data and don't visualize it (notice the retrieved data until that its kept in the memory).

```
layer.setEnabled(false);
```

Sometimes is interesting only show a layer when you are really close to the terrain or when you are far of it. You can establish some visibility ranges to reproduce this behavior with this methods:

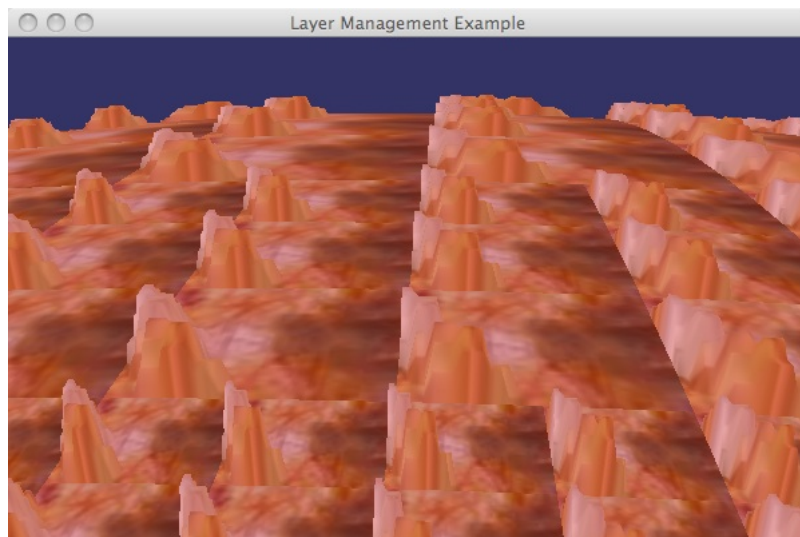
- **setMinLevel**: establish the minimum level of range necessary to see the layer.
- **setMaxLevel**: establish the maximum level of range since the layer is no longer visible.
- **setMaxResolution**: they are useful to set the maximum level of the data resolution. Since this level no more data is requested but the textures are propagated to the higher resolution ranges.

Notice that the level is a integer that indicates the number of subdivisions of the mesh and represents the quality of the rendered data. Tip: When a layer is retrieved the level is indicated in the retrieve event.

Also, each layer type define some specifically parameters. For example, you can change the opacity only for the RasterLayer using the method *setOpacity*. For elevations you can change the vertical exaggeration with the *setVerticalExaggeration*. You must put values between 0 and 1 to reduce the elevations or put values bigger than 1 to exaggerate them. You can add this methods to the key listener of your example to change the properties of the first layer.

```
if (e.getKeyCode() == KeyEvent.VK_7) {  
    layer0.setOpacity(0.5f);  
}  
  
if (e.getKeyCode() == KeyEvent.VK_8) {  
    layer1.setVerticalExaggeration(20.0f);  
}
```

Finally, a full implementation of this code is available on package `UserGuide` of the `osgvp-examples`, and shows like the next figure:



Other examples more complex are included in the examples framework. See them for more information.

Terrain utilities

The *Terrain* class has other useful methods. In this section we explain them to give more functionality to your application.

- **getZoom, getLongitude, getLatitude, getAltitude:** they give you the position of the camera in the unities of the coordinate system of the planet.

- **convertXYZToLatLongHeight, convertLatLongHeightToXYZ:** you can convert coordinates between the XYZ and latitude, longitude and height using the real ellipsoid of the planet. It's very important to set the correct radius in all of the type planets for a good performance of these methods. They are very useful to put some 3D objects over the planet surface because all of the nodes put in the special node scene data are in XYZ coordinate system.

OSGVP Features

The osgvpfeatures library is capable to draw vectorial data such as text, points, lines, polygons, simple geometric figures and extruded figures in a three-dimensional way.

All of this features could be edited in the same way. Text and shapes lacks some geometric values which are essential to edit them.

Blending and color changes are supported for all the features. Adding or removing vertexes are too supported operations. Polylines and PixelPoints could be anti-aliased.

Users can extrude simple geometric forms using different techniques. New features can be added without modifying classes hierarchy.

Overview

In a GIS, geographical features are often expressed as vectors, by considering those features as geometrical shapes. Different geographical features are best expressed by different types of geometry. Each of these geometries are linked to a row in a database that describes their attributes. This information can be used to make a map to describe a particular attribute of the dataset.

Vector data can be displayed as vector graphics used on traditional maps, whereas raster data will appear as an image that may have a blocky appearance for object boundaries. Vector data can be easier to register, scale, and re-project. This can simplify combining vector layers from different sources. Vector data are more compatible with relational database environment. They can be part of a relational table as a normal column and processes using a multitude of operators.

The visualization of vectorial data will be faster if we use the facilities that scene graphs give us. An API that allow developers showing vectorial data with some guarantee must be implemented. This API will offer support to draw geometric figures and to do basic vector operations.

Basic and common elements in a lot of GIS systems are: points, lines, polylines, polygons and multi-polygons. This entities are those which the library could represent.

Looking for simplicity we decided to implement an abstraction layer over OpenSceneGraph. The mapping isn't direct between OSG and Java classes(like in osgvp-core).This classes remains over osgvpfeatures library, implemented in C++.

Points

In GIS is useful to show points in different metric units, at least pixels and meters. The osgVP API for Points lets the user to change the size, color, transparency, etc. dynamically. It works in the same way of GL_POINTS, one primitive could have several points. There are two main classes for drawing points:

- PixelPoint: represents points in pixels.
- QuadPoint: represents points in meters with a quad geometry where the real position of the point is the center of the quad.

Let's see some code example. A full implementation of this example is available on "Points Example" of the examples framework. See the PointsExample class in examples package of source code.

```
PixelPoint points;
g = new Group();
try {
    points = new PixelPoint();
} catch (NodeException e) {
    e.printStackTrace();
}
points.setPointSize(5.0f);
points.setEnabledSmoothing(false);
for (int i = 0; i < 1000; i++) {
```

```
Vec3 position1 = new Vec3(Math.random() * 100,
    Math.random() * 100, Math.random() * 100);
Vec4 color1 = new Vec4(Math.random(), Math.random(),
    Math.random(), 1);
points.addPoint(position1, color1);
}
g.addChild(points);
```

If you want to use `QuadPoints` you can set `billboarding enabled` to rotate the quads to the screen. A full implementation of this example is available on “`QuadPoints Example`” of the examples framework. See the `QuadPointsExample` class in `examples` package of source code.

```
QuadPoint points;
g = new Group();
try {
    points = new QuadPoint();
} catch (NodeException e) {
    e.printStackTrace();
}
points.setPointSize(5.0f);
points.setBillboardingEnabled(true);
for (int i = 0; i < 1000; i++) {
    Vec3 position1 = new Vec3(Math.random() * 100,
        Math.random() * 100, Math.random() * 100);
    Vec4 color1 = new Vec4(Math.random(), Math.random(),
        Math.random(), 1);
    points.addPoint(position1, color1);
}
g.addChild(points);
```

A shape point class can be used to replace points for another geometry defined by the user as shown in the following code. A full implementation of this example is available on “`Shape Point Example`” of the examples framework. See the `ShapePointExample` class in `examples` package of source code.

```
private ShapePoint shape;
```

```
Sphere esfera = new Sphere();
esfera.setColor(new Vec4(1.0, 0.5, 0.0, 1.0));
esfera.setDetailRatio(0.9f);

Box box = new Box();
box.setCreateTop(false);

Cylinder cylin = new Cylinder();
cylin.setCreateBody(true);

CompositeShape compostela = new CompositeShape();
compostela.addChild(esfera);
compostela.addChild(cylin);
compostela.setColor(new Vec4(1.0, 1.0, 0.0, 0.5));

shape = new ShapePoint(compostela);
for (int i = 0; i < 1000; i++) {
    Vec3 position1 = new Vec3(Math.random() * 100,
        Math.random() * 100, Math.random() * 100);
    Vec4 color1 = new Vec4(Math.random(), Math.random(),
        Math.random(), 1);
    shape.addPoint(position1, color1);
}
```

Polylines

Drawing polylines should be easy with osgVP. The API let change the width, color, pattern or blending of the polylines. It works with lines like OpenGL does.

```
Group g = new Group();
Polyline lines;
try {
    lines = new Polyline();
} catch (NodeException e) {
    e.printStackTrace();
}
```

```
lines.setWidth(100);
//Like points, Polyline can be antialias
lines.setEnabledSmoothing(true);
lines.setEnabledBlending(true);
for (int i = 0; i < 1000; i++) {
    Vec3 position1 = new Vec3(Math.random() * 10,
        Math.random() * 10, Math.random() * 10);
    Vec4 color1 = new Vec4(Math.random(), Math.random(),
        Math.random(), Math.random());
    lines.addVertex(position1, color1);
}
g.addChild(lines);
```

To change the pattern of a polyline you must set the pattern as a 16 bits variable which is repeated as necessary along line feature. Then you should set the factor, it serves to scale pattern and must be in range [1,255].

Samples:

```
polyline.setpattern((short)0XAAAA);
polyline.setFactor(3);
```

A full implementation of this example is available on “Polyline Example” of the examples framework. See the PolylineExample class in examples package of source code.

An extension of Polyline class is the InteractivePolyline ones, which allow to draw a line interactively adding points one by one clicking with the mouse over a geometry surface as shown in the next example:

```
Group g = new Group();
InteractivePolyline polyline = new InteractivePolyline();

g.addChild(osgDB.readNodeFromFileFromResources("/cow.ive"));
g.addChild(polyline);

getCanvas3D().addMouseListener(this);
getCanvas3D().addMouseMotionListener(this);
```

```
float factor=0.1f;
boolean clicked;

...

public void mousePressed(MouseEvent arg0) {

    if (arg0.getButton() == MouseEvent.BUTTON3) {

        clicked = true;
        Vec3 normal = hits.getFirstIntersection()
            .getIntersectionNormal();
        normal.normalize();
        polyline.setMouseCoords(hits.getFirstIntersection()
            .getIntersectionPoint().sum(normal
            .escalarProduct(factor)));
    }
    try {
        polyline.update();
    } catch (OSGVPEException e) {
        e.printStackTrace();
    }
    polyline.setEnabledNode(true);

}

public void mouseMoved(MouseEvent arg0) {

    if (clicked) {
        Intersections hits = getCanvas3D().getOSGViewer()
            .rayPick(arg0.getX(), arg0.getY());
        Vec3 normal = hits.getFirstIntersection()
            .getIntersectionNormal();
        normal.normalize();
        polyline.setMouseCoords(hits.getFirstIntersection()
            .getIntersectionPoint().sum(normal
            .escalarProduct(factor)));
    }
}
```



```
        try {
            polyline.update();
        } catch (OSGVPEException e) {
            e.printStackTrace();
        }
    }
}

public void mouseReleased(MouseEvent arg0) {

    Intersections hits = getCanvas3D().getOSGViewer()
        .rayPick(arg0.getX(), arg0.getY());

    if (arg0.getButton() == MouseEvent.BUTTON3) {

        Vec3 normal = hits.getFirstIntersection()
            .getIntersectionNormal();
        normal.normalize();
        polyline.setMouseCoords(hits.getFirstIntersection()
            .getIntersectionPoint().sum(normal
            .escalarProduct(factor)));
        polyline.addVertex(hits.getFirstIntersection()
            .getIntersectionPoint().sum(normal
            .escalarProduct(factor)));

        try {
            polyline.update();
        } catch (OSGVPEException e) {
            e.printStackTrace();
        }
    }
}
```

A full implementation of this example is available on “InteractivePolyline Example” of the examples framework. See the InteractivePolylineExample class in examples package of source code.

Polygons

The class Polygon involves all the polygon functionality (patterns, textures...). You can define what kind of polygon you want to renderize: empty (only borders), filled, pattern or point(only vertexes) polygons.

Is important when you define a polygon to specify coords for polygons into a anti-clockwise direction for their front face to be pointing toward your, get this wrong and you could find back face culling removing the wrong faces of your model. A texture could be applied to the polygon, and the user can rotate, scale or translate this texture.

For convex polygons, those that OpenGL can't paint without errors, the user will use TessellablePolygon to triangulate convex geometries.

Example:

```

Polygon _rect;
g = new Group();
Vec4 color = new Vec4(1.0, 1.0, 0.0, 1.0);
try {
    _rect = new Polygon();
} catch (NodeException e1) {
    e1.printStackTrace();
}
_rect.setType(Polygon.PolygonType.FILLED_POLYGON);
//vertices
_rect.addVertex(new Vec3(-5, 0, 0), color);
_rect.addVertex(new Vec3(0, 5, 0), color);
_rect.addVertex(new Vec3(-5, 10, 0), color);
_rect.addVertex(new Vec3(10, 10, 0), color);
_rect.addVertex(new Vec3(5, 5, 0), color);
_rect.addVertex(new Vec3(10, 0, 0), color);
//normals
Vector<Vec3> normalarray = new Vector<Vec3>();
Vector<Vec3> normalarray1 = new Vector<Vec3>();
normalarray.add(new Vec3(0, 0, -1));
_rect.setNormalArray(normalarray);
_rect.setNormalBinding(GeometryFeature.
    AttributeBinding.BIND_OVERALL);

```

```
g.addChild(_rect);
```

If tessellable polygon is needed:

```
_rect1 = new TessellablePolygon();
_rect1.setType(Polygon.PolygonType.FILLED_POLYGON);
_rect1.addVertex(new Vec3(-5, 0, 5), color);
_rect1.addVertex(new Vec3(-5, 10, 5), color);
_rect1.addVertex(new Vec3(10, 10, 5), color);
_rect1.addVertex(new Vec3(20, 5, 5), color);
_rect1.addVertex(new Vec3(10, 0, 5), color);
_rect1.setNormalArray(normalarray1);
_rect1.setNormalBinding(GeometryFeature.
    AttributeBinding.BIND_PER_VERTEX);
File texture = Util.extractFromURL(this.getClass().getResource(
    "/earth.gif"));
_rect1.setTexture(texture.getPath());
_rect1.setEnabledBlending(true);
_rect1.tessellate();
```

A full implementation of this example is available on “Polygon Example” of the examples framework. See the PolygonExample class in examples package of source code.

Text

Text has several methods that control its size, appearance, orientation, and position. The following section describe how to control some of these parameters. To use osgText in your application, you usually need to perform three steps:

1. To display multiple text strings using the same font, create a single Font object that you can share between all Text objects.
2. For each text string to display, create a Text object. Specify options for alignment, orientation, position, and size. Assign the Font object you created in step 1 to the new Text object.
3. Add your Text objects to a Geode using addDrawable(). You can add multiple Text objects to a single Geode or create multi-

ple Geode objects, depending on your application requirements.
Add your Geode objects as child nodes in your scene graph.

```
Text text = null;
    try {
        text = newText();
    } catch (NodeException e) {
        e.printStackTrace();
    }
    text.setFont("arial.ttf");
    text.setText("example");
    t1.setPosition(0f, 0f, 20f);
    text.setCharacterSize(3.0f);

    try {
        g.addChild(t);
        ...
    }
```

A full implementation of this example is available on “Text Example” of the examples framework. See the `TextExample` class in examples package of source code.

Extruded Geometries

Extrusion classes in `osgvpfeatures` extends of `OSGExtruder` class, belonging `osgvp-core` library. This class is mapped against a generic extruder, created in C++ from a J.Hidalgo Project (Department of Computer System & Computation of UPV). It works over a stack matrix system. From this class we can create specific extruders depending the geometry we want to extrude. So, there are three specific extruders: `PointExtruder`, `PolylineExtruder` and `PolygonExtruder`. They three work in a very similar way. But the extruder returns a determined kind of geometry depending what was the input.

The following example show how to extrude a point feature. A full implementation of this example is available on “Point Example” of the examples framework. See the `PointExtrusionExample` class in examples package of source code.

```
Group g = new Group();
```

```
Geode ge = new Geode();
PixelPoint shape = new PixelPoint();

Vec4 color = new Vec4(1.0, 0.0, 0.0, 1.0);
Vec4 color1 = new Vec4(1.0, 1.0, 0.0, 1.0);
Vec4 color2 = new Vec4(1.0, 0.0, 1.0, 1.0);

shape.addVertex(new Vec3(0, 0, 0), color);
shape.addVertex(new Vec3(0, 0, 0), color1);
shape.addVertex(new Vec3(0, 0, 0), color2);
shape.addVertex(new Vec3(0, 0, 0), color1);
shape.addVertex(new Vec3(0, 0, 0), color);
shape.addVertex(new Vec3(0, 0, 0), color2);
shape.addVertex(new Vec3(0, 0, 0), color1);

PointExtruder extruder = new PointExtruder();
Vector<Vec3> vec = new Vector<Vec3>();
Vector<Double> vecdou = new Vector<Double>();

vec.add(new Vec3(0, 0, 1));
vec.add(new Vec3(0, 1, 0));
vec.add(new Vec3(1, 0, 0));
vec.add(new Vec3(0, 0, -1));
vec.add(new Vec3(0, -1, 0));
vec.add(new Vec3(-1, 0, 0));
vec.add(new Vec3(1, 1, 0));
vecdou.add(5.0);
vecdou.add(10.0);
vecdou.add(5.0);
vecdou.add(10.0);
vecdou.add(5.0);
vecdou.add(10.0);
vecdou.add(15.0);

ShapePoint cero = new ShapePoint();
Sphere sphera = new Sphere();

try {
```

```
        cero.setShape(sphera);
    } catch (ChildIndexOutOfBoundsExceptions e2) {
        e2.printStackTrace();
    } catch (NodeException e2) {
        e2.printStackTrace();
    }
}

cero.addPoint(new Vec3(0, 0, 0), new Vec4(1, 0, 1, 1));
try {
    g.addChild(cero);
} catch (NodeException e1) {
    e1.printStackTrace();
}

try {
    ge.addDrawable(extruder.extrude(shape, vec, vecdou));
} catch (NodeException e3) {
    e3.printStackTrace();
}

try {
    ge.getOrCreateStateSet().setLightingMode(
        Node.Mode.OFF | Node.Mode.PROTECTED);
} catch (InvalidValueException e2) {
    e2.printStackTrace();
}

try {
    g.addChild(ge);
} catch (NodeException e1) {
    e1.printStackTrace();
}
}
```

Extruding polyline features is possible too, as shown in next example. A full implementation of this example is available on “Polyline Extrusion Example” of the examples framework. See the `PolylineExtrusionExample` class in `examples` package of source code.

```
Group g = new Group();
Geode ge = new Geode();
Polyline shape = new Polyline();

Vec4 color = new Vec4(1.0, 0.0, 0.0, 1.0);
Vec4 color1 = new Vec4(1.0, 1.0, 0.0, 1.0);
Vec4 color2 = new Vec4(1.0, 0.0, 1.0, 1.0);

shape.addVertex(new Vec3(0, 0, 0), color);
shape.addVertex(new Vec3(5, 0, 0), color1);
shape.addVertex(new Vec3(5, 5, 5), color2);
shape.addVertex(new Vec3(5, 10, 0), color1);
shape.addVertex(new Vec3(0, 5, 0), color);
shape.addVertex(new Vec3(-5, 5, 0), color2);

PolylineExtruder extruder = new PolylineExtruder();
ShapePoint cero = new ShapePoint();
Sphere sphera = new Sphere();

try {
    cero.setShape(sphera);
    cero.addPoint(new Vec3(0, 0, 0), new Vec4(1, 0, 1, 1));
    g.addChild(cero);
    ge.addDrawable(extruder.extrude(shape,
        new Vec3(0.0, 0.0, 1.0), 10));
    ge.getOrCreateStateSet().setTwoSidedLighting(true,
        Node.Mode.ON);
    g.addChild(ge);
} catch (ChildIndexOutOfBoundsException e1) {
    e1.printStackTrace();
} catch (NodeException e1) {
    e1.printStackTrace();
}
}
```

See next example to view a polygon extrusion. A full implementation of this example is available on “Polygon Extrusion Example” of the examples framework. See the PolygonExtrusionExample class in examples package of source code.

```
PolygonExtruder pol = new PolygonExtruder();
Polygon shape= null;
try {
    shape = new Polygon();
} catch (NodeException e) {
    e.printStackTrace();
}

shape.addVertex(new Vec3(2, 0, 0), color);
shape.addVertex(new Vec3(2, 0, 5), color);
shape.addVertex(new Vec3(2, 5, 5), color);
shape.addVertex(new Vec3(2, 5, 0), color);

pol.extrude(shape, 10);
Geode ge = new Geode();
ge.addDrawable(pol.getGeometry());
```

Particles

A system particle can be used like a feature as shown in the next example:

```
ParticleSystem ps = new ParticleSystem();

root.addChild(ps.getGroup());
root.addChild(ps.getPat());
```

A full implementation of this example is available on “Particle Example” of the examples framework. See the `ParticleSystemExample` class in examples package of source code.

You can look up the examples framework for further information. We explained the most common features available in this package, by the way other helpful classes are available in this library.

OSGVP Manipulator

In this section, you will learn how to use the classes provided by the library `osgVP-manipulator`. You will learn how to add a manipulator to a node of the scene-graph and how to transform it, as well how to manage all the manipulators present in the scene. We provide the same manipulators implemented in OSG, as well as a new type of manipulator that allows the transformation of individual vertex of a given geometry.

The Manipulator node

To add a **Manipulator** to an existing node is a very simple task. First of all, you have to create an instance of the class `Manipulator`. There are two possibilities to create a `Manipulator`.

```
public Manipulator();  
public Manipulator(String draggerType);
```

The only difference between these two constructors is what type of manipulator will be created. The argument *draggerType* specifies this type. If no argument is passed, the default manipulator will be created.

Types of dragger

Here is a list with all the draggers available in this version of the library:

Scale1DDragger: Scales the object over an axis.

Scale2DDragger: Scales the object over two given axis.

ScaleAxisDragger: Scale the object over the three axis.

TabBoxDragger: Scales the object through the eight corners of a box containing the object. Also permits to translate the object picking on one of the six planes which form this box. This is the default manipulator.

TabPlaneDragger: Scales and translates the object through a plane.

TabPlaneTrackballDragger: Scales and translates the object through a plane. Also, rotates the object through a sphere.

TrackballDragger: Rotates the object through a sphere that contains it.

Translate1DDragger: Translates the object over an axis.

Translate2DDragger: Translates the object over two given axis.

TranslateAxisDragger: Translates the object over the three axis.

TranslatePlaneDragger: Translates the object over a plane.

Adding a Node

Once a Manipulator has been created, the method

```
public boolean addChild(Node child);
```

inserts the given node inside the manipulator. This method can be used as many times as wanted, therefore a Manipulator can transform several objects at the same time.

Other available methods

```
public void setDragger(String draggerType);
```

Changes the dragger type.

```
public Node getChild(int i);
```

Returns the node at the position i .

```
public int getNumChildren();
```

Returns the number of nodes being manipulated at the moment.

```
public boolean removeChild(Node child);
```

Removes the child

```
public boolean removeChildren();
```

Removes all the children inside the Manipulator.

```
public Group getSelection();
```

Returns the subgraph with all the objects transformed.

```
public boolean removeChild(int i);
```

Removes the child number i of the Manipulator

```
public void setChild(int i, Node node);
```

Puts the node as the child number i of the Manipulator.

```
public String getDragger();
```

Returns the dragger type.

Setting the Manipulator Handler

Once one or more nodes have been added to a manipulator, you can instantiate the class `ManipulatorHandler` in order to interact with them.

```
ManipulatorHandler();
```

Once the new class has been created, it must be added as an `EventHandler` to the `OSGViewer`.

```
getCanvas3D().getOSGViewer().addEventHandler(_hand);
```

Now, the object can be transformed depending on the dragger type selected. If your application wants to enable or disable this handler, use the method:

```
public void setActive(boolean active);
```

Manipulate an object

Let's see an example that shows the use of the Manipulator node. (Note that this is not exactly the same code present in the library examples. It has been simplified to give a better understanding of the functionality. For example, the try/catch clauses have been removed)

```
private ManipulatorHandler _hand;
private Manipulator _manip;
private Node _cow;

_cow = osgDB.readNodeFromFileFromResources("/cow.ive");
_manip = new Manipulator(Manipulator.DraggerType
    .TRANSLATE_PLANE_DRAGGER);
_manip.addChild(_cow);
_hand = new ManipulatorHandler();
getCanvas3D().getOSGViewer().addEventHandler(_hand);
```

In this example, the 3D model stored in the file *cow.ive* is loaded as a node. The next step is to create an instance of Manipulator. In this case, we have chosen the *TRANSLATE_PLANE_DRAGGER*, which draws a plane around the object and allows to translate the objects clicking on it. Later, we add the loaded node as a child of the Manipulator. Instantiating a ManipulatorHandler and adding it to the OSGViewer ends the process.

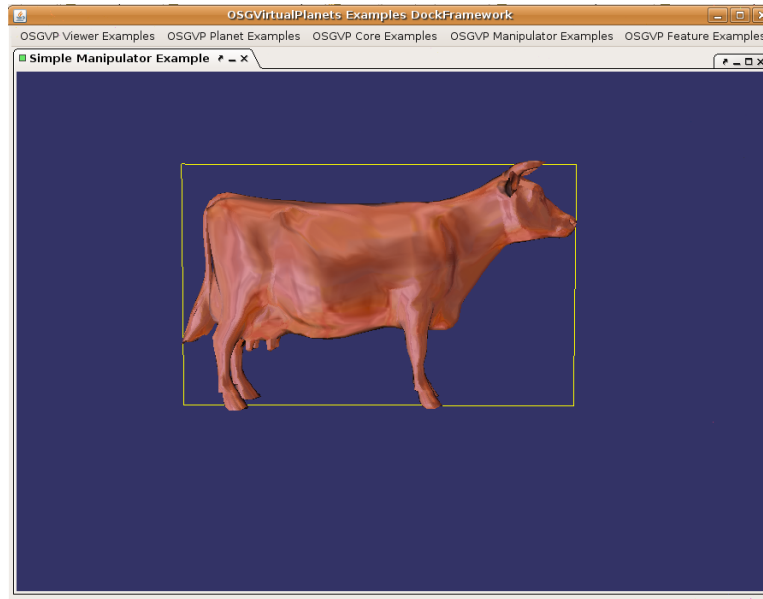


Figure 18: Node being Manipulated.

A full implementation of this example is available on “Simple Manipulator Example” and “Feature Manipulation Example” of the examples framework. See the `SimpleManipulator` and `FeatureManipulationExample` class in `examples` package of source code.

Managing the Scene with `EditionManager`

`EditionManager` is a class created to help the developers to build a more advanced edition systems. It allows the creation and management of many Manipulators at the same time, and even permits the interaction between them. Its use is very similar to the `Manipulator` class. The following example shows its use:

```
_manager = new EditionManager(scene);
getCanvas3D().addKeyListener(new ViewerStateListener(
    getCanvas3D().getOSGViewer()));
getCanvas3D().addKeyListener(this);
_hand = new ManipulatorHandler();
getCanvas3D().getOSGViewer().addEventHandler(_hand);
getCanvas3D().addMouseListener(this);
```

A full implementation of this example is available on “Edition Manager Example” of the examples framework. See the `EditionManager` class in `examples` package of source code.

Methods implemented by Edition Manager

```
public Node getScene()
```

Gets the part of the scene not manipulated.

```
public Group getTransformedScene()
```

Gets the whole transformed scene without the draggers.

```
public void setScene(Node node)
```

Changes the whole scene in the `EditionManager`.

```
public int getNumChildren()
```

Returns the number of children.

```
public Manipulator addNode(int i)
```

Creates a manipulator for the node at the position i in the scene branch of the EM.

```
public Node removeNode(Node object)
```

Extracts the node from the manipulator and returns it to the scene branch.

```
public void removeAllNodes()
```

Extract all transformed nodes and put them in the scene branch.

```
public void deleteSelectedNodes()
```

Deletes all the Manipulators and the nodes inside them.

```
public void changeDragger(String draggerType)
```

Changes the type of all the active draggers.

```
public String getDraggerType()
```

Returns the type of the dragger.

```
public void group()
```

Gets all manipulators in the scene and creates one with all of them.

```
public void ungroup()
```

Separates different nodes present in a manipulator and makes one manipulator for each of them.

Implementing the picking functionality

To interact with the `EditionManager` class, your Java application must implement at least one `MouseListener`. The following code shows one simple example.

```
public void mouseClicked(MouseEvent arg0) {
    if (arg0.getButton() == MouseEvent.BUTTON1) {
        Intersections polytopeHits =
            getCanvas3D().getOSGViewer()
                .rayPick(_manager, arg0.getX(),
                    arg0.getY(),
                    Manipulator.NEG_MANIPULATOR.NODEMASK);
        if (polytopeHits.containsIntersections()) {
            Intersection hit = polytopeHits.
                getFirstIntersection();
            Node nodeHit = (Node) (hit.getNodePath().get(2));
            int k;
            k = nodeHit.getParent(0).getChildIndex(nodeHit);
            AddSelectionCommand addCommand =
                new AddSelectionCommand(k, _manager);
```

```
addCommand.execute();
_commands.add(addCommand);

} else {

RemoveAllSelectionCommand removeAllCommand =
    new RemoveAllSelectionCommand(_manager);
removeAllCommand.execute();
_commands.add(removeAllCommand);

}

}

}
```

In this example, when the left button of the mouse is clicked, it computes the intersections with the objects present in the scene and stores the first object that intersects. Later, the subgraph containing the object is included in a new Manipulator. This process is shown in the figure next figure:

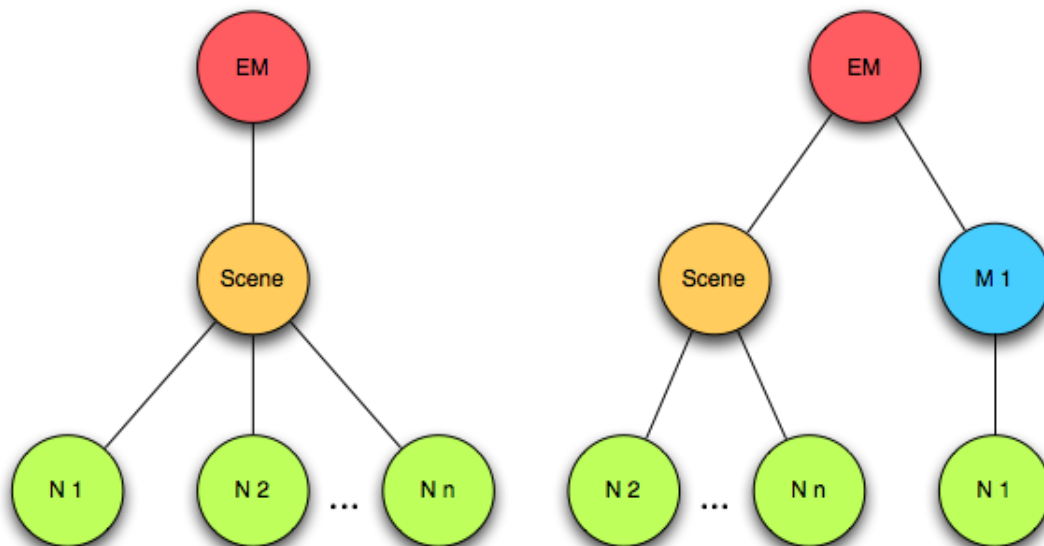


Figure 19: Adding a node to the EditionManager.

If the mouse is clicked and no intersections are computed, the listener makes all the manipulators disappear. One important part is how to extract the node to add to the EditionManager. In this example, we are assuming that the EM is the root node of the scene, therefore the nodepath level selected is 2 (as can be seen in the figure). If the EM is in a sublevel of the graph, the nodepath level selected must be changed. For example, if the EM is a child of the root of the scene, the nodepath level selected must be 3.

A full implementation of this example is available on “Picking Example” of the examples framework. See the PickingExample class in examples package of source code.

The GeometryManipulator node

The Manipulator node transforms the objects through a MatrixTransform in the scene-graph. In this way, the transformations are applied to the whole subgraph below it. To achieve a more advanced functionality that allows to edit each vertex of the object individually, we have created the GeometryManipulator node. The use of this node is very similar to the Manipulator. First of all, you have to instantiate the class GeometryManipulator with the following method:

```
public GeometryManipulator(Geometry geo,  
                           Vector<Integer> indexArray);
```

The first argument is the Geometry to be manipulated, the second is a vector that contains the selected vertex indices of that geometry. The ManipulatorHandler class must be instantiated and added to the OSGViewer to transform the vertices too. In the next figure you can see a geometry with some of its vertices being manipulated.

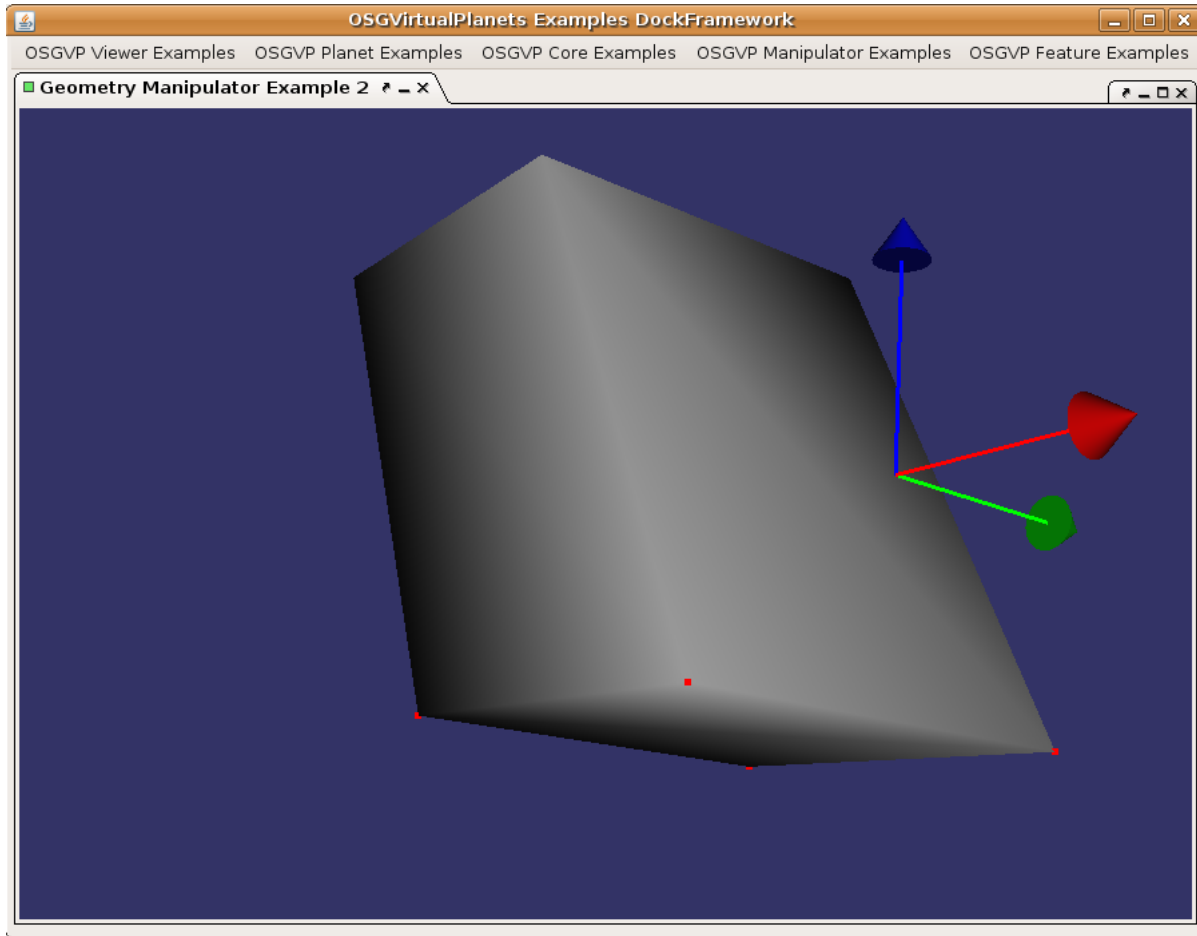


Figure 20: GeometryManipulator example.

OSGVP Symbology

In this section, you will learn how to use the classes provided by the library `osgVP-symbology`. In a GIS, the symbols are used to represent a feature like a point, with an object that represents this point, for example with an image. With the `osgVP-symbology` library, we can change how a 3D GIS feature (defined with `osgVP-features`) is rendered in the screen and change these symbols interactively.

The symbol visitor

To replace the current features for the symbols in the scene-graph, we have defined a symbol visitor to traverse it and replace the geometry for the current one. There are specific visitors for each type of symbol, however the class `Symbol3DVisitor` can manage all of them and apply the correct one.

Basic symbols

We provide the basic symbols like boxes, circles, cones, cylinders, text, spheres, etc. for rendering features. For example, if we have defined a set of points in our scene-graph, we can represent them with boxes as shown in the next example code:

```
Group root = new Group();
BoxSymbol bs = new BoxSymbol();
Symbol3DVisitor sv = new Symbol3DVisitor();

// Point features
```

```
for (int i = 0; i < 1000; i++) {

    Point3D p = new Point3D();
    p.setX(Math.random() * 100);
    p.setY(Math.random() * 100);
    p.setZ(Math.random() * 100);
    Vec4 color = new Vec4(Math.random(), Math.random(),
        Math.random(), 1 - Math.random());

    bs.addPoint(p, color);
}

bs.setPointSize(2);
bs.setSizeMetric(PointSymbol3D.SizeMetric.PIXELS);

// Replace the points by symbols
bs.traverse(sv);

// Add the result to scenegraph
root.addChild(sv.getSceneData());
```

A full implementation of this example is available on “BoxSymbol Example” of the examples framework. See the `BoxSymbolExample` class in `examples` package of source code. Other basic symbols examples are available in this framework too.

Composite symbols

In addition to the basic symbols, the composite symbol can represent a feature like a group of basic symbols as shown in this example:

```
Group root= new Group();
CircleSymbol ps = new CircleSymbol();
PolylineSymbol3D polylinesym = new PolylineSymbol3D();
FillSymbol3D fillsym = new FillSymbol3D();
CompositeSymbol3D composite = new CompositeSymbol3D();
Symbol3DVisitor sv= new Symbol3DVisitor();
```

```
//Add features
for (int j =0; j <10 ; j++)
{
    Polyline3D polyline = new Polyline3D();
    Polygon3D poli = new Polygon3D();
    Vec4 color = new Vec4(Math.random(), Math.random(),
        Math.random(), 1 - Math.random());
    Vec4 color1 = new Vec4(Math.random(), Math.random(),
        Math.random(), 1 - Math.random());
    for (int i = 0; i < 3; i++) {
        Point3D p = new Point3D();
        p.setX(Math.random() * 100);
        p.setY(Math.random()* 100);
        p.setZ(j*50);

        Vec3 direction = new Vec3(Math.random(),
            Math.random(), Math.random());
        double height = Math.random()*10;
        //cs.addPointExtrusion(p, color,height,direction);

        polyline.addVertex(p);
        poli.addVertex(p);
    }
    polylinesym.addPolyline(polyline, color1);
    fillsym.addPolygon(poli,color);
}

polylinesym.setStipple(1, (short) 0XAAAAA);
polylinesym.setWidth(3f);
polylinesym.setLineLoop(true);

composite.addSymbol(fillsym);
composite.addSymbol(polylinesym);
composite.traverse(sv);

root.addChild(sv.getSceneData());
```

A full implementation of this example is available on “Composite Symbol Example” of the examples framework. See the `CompositeSymbolExample` class in examples package of source code.

Extruded symbols

Also, the library provides the mechanism to extrude points, polylines and polygons like symbols:

```
Group root = new Group();
PointExtrusionSymbol cs = new PointExtrusionSymbol();
Symbol3DVisitor sv = new Symbol3DVisitor();

// Add features
for (int i = 0; i < 1000; i++) {
    Point3D p = new Point3D();
    p.setX(Math.random() * 100);
    p.setY(Math.random() * 100);
    p.setZ(Math.random() * 100);
    Vec4 color = new Vec4(Math.random(), Math.random(),
        Math.random(), 1 - Math.random());
    Vec3 direction = new Vec3(Math.random(), Math.random(),
        Math.random());
    double height = Math.random()*10;
    cs.addPointExtrusion(p, color,height,direction);
}

cs.traverse(sv);
root.addChild(sv.getSceneData());
```

A full implementation of this example is available on “Point Extrusion Symbol Example” of the examples framework. See the `PointExtrusionSymbolExample` class in examples package of source code. A example for polygon and polyline extrusion is also available.

Node symbols

Finally, we can use any OSG geometry like a symbol using the node symbol class:

```
Group root = new Group();
NodeSymbol3D cs = new NodeSymbol3D();
Symbol3DVisitor sv = new Symbol3DVisitor();

// Add features
for (int i = 0; i < 1000; i++) {
    Point3D p = new Point3D(0,0,0);
    p.setX(Math.random() * 100);
    p.setY(Math.random() * 100);
    p.setZ(Math.random() * 100);
    Vec4 color = new Vec4(Math.random(), Math.random(),
        Math.random(), 1 - Math.random());
    cs.addPoint(p, color);
}

cs.setPointSize(30);
cs.setNode(osgDB.readNodeFromFileFromResources("/cow.ive"));

cs.traverse(sv);
root.addChild(sv.getSceneData());
```

A full implementation of this example is available on “Node Symbol Example” of the examples framework. See the `NodeSymbolExample` class in `examples` package of source code.