OSG Virtual Planets User Guide

Info: http://gvsig.org/web/projects/gvsig-commons/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: 2008-12-22, 13:02
Revision: 1
Description: OSG Virtual Planets User Guide.

Contents

1

Introduction

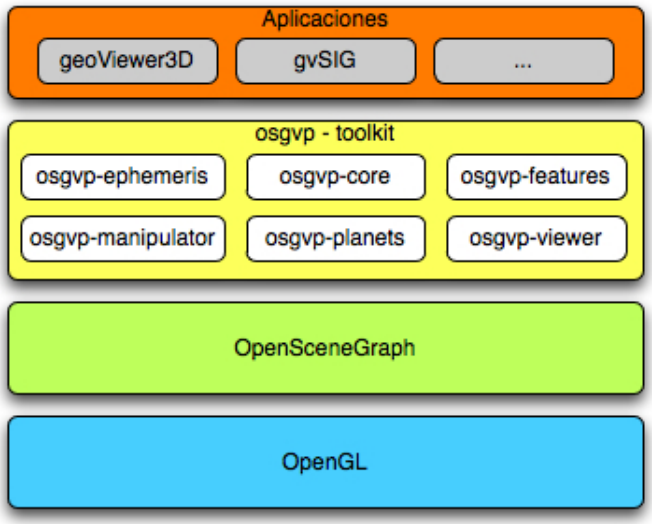
Welcome to the OSG Virtual Planets (osgVP) Users Guide. This manual serves as a reference to the osgVP library, and collecting documentation and examples provided with the code.

As you probably know, any software application requires some effort to learn. We've 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++).

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-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...
- osgVP-planets: this library allows developers to create Planet specific scene graphs and manages geometry generation and memory paging. Also controls texture and elevation layer.
- osgVP-features: serves to draw vectorial data such as text, points, lines, polygons, simple geometric figures and extruded figures.
- osgVP-manipulator: manages the edition of transformations associated to 3D objects. The library also manages modifications over Geometries.

1.1 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 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 XP, Ubuntu Linux (Hardy Heron 8.04 release) and Mac OS X (on Mac Intel: Mac Book Pro and iMac 20, both "core 2 duo").

1.2 Download and Installation

First of all, download and install OpenSceneGraph by following the instructions at http://www.openscenegraph.org Suggest that you also download the OpenSceneGraph data sets, follow the instructions for setting up your environment variables and verify that you can run the OpenSceneGraph examples before moving on.

Download osgVP SDK from Downloads section in the osgVP site. This package contains native libraries whose format depends on your Operative System (.so for linux, .dll for windows and .dylib for os X) and necessary jar files to compile and run your osgvp projects and compiled examples framework.

The SDK package contains:

- binaries directory (~/osgVP/binaries): Native binaries per platform and the required precompiled dependencies.
- lib directory (~/osgVP/lib): Jar files.
- product directory (~/osgVP/product): Compiled examples ready to be launched.

Resources required for carrying out osgvp projects:

- Jar files:

  - gluegen-rt-1.1.0.jar
  - jogl-1.1.0.jar
  - libNative-1.0.jar
  - libosgvp-core-2.x.x.jar
  - libosgvp-examples-2.x.x.jar
  - libosgvp-features-2.x.x.jar
  - libosgvp-manipulator-2.x.x.jar
  - libosgvp-planets-2.x.x.jar
  - libosgvp-viewer-2.x.x.jar

- Native libraries:

  - libjniosgvpcore.*
  - libjniosgvpmanipulator.*
  - libjniosgvpviewer.*
  - libosgvpfeatures.*
  - libosgvpplanets.*
  - libjniosgvpfeatures.*
  - libjniosgvpplanets.*
  - libosgvpcore.*
  - libosgvpmanipulator.*
  - libosgvpviewer.*

Once you have downloaded the libraries, there are several forms to include it on to your project, depending how you are developing.

If your are developing with Eclipse, maybe the easy-way is creating a “lib” directory inside your project and include 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>2.1.7</version>
        </dependency>
        <dependency>
                <groupId>org.gvsig.osgvp.libosgvp</groupId>
                <artifactId>libosgvp-features</artifactId>
             <version>2.1.7</version>
        </dependency>
        <dependency>
             <groupId>org.gvsig.osgvp.libosgvp</groupId>
             <artifactId>libosgvp-manipulator</artifactId>
             <version>2.1.7</version>
        </dependency>
        <dependency>
             <groupId>org.gvsig.osgvp.libosgvp</groupId>
             <artifactId>libosgvp-planets</artifactId>
             <version>2.1.7</version>
        </dependency>
        <dependency>
            <groupId>org.gvsig.osgvp.libosgvp</groupId>
            <artifactId>libosgvp-viewer</artifactId>
            <version>2.1.7</version>
        </dependency>
    </dependencies>
```

and also need to add a new remote repository in your pom.xml

```
    <repositories>
            ...
            <repository>
                    <id>gvsig-http-repository</id>
                <url>http://downloads.gvsig.org/pub/gvSIG-desktop/
                    maven-repository</url>
            </repository>
    </repositories>
```

The native binaries are not deployed in maven so you need to use the libraríes inside SDK.

2

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.

2.1 Running the examples

Once the osgVP SDK is installed the next step is to run the runexamples script located in ~/osgVP/product. If your OS is Windows you shall execute run-Examples.bat, in case you are running on linux or on Mac osX you must execute run-Examples.sh.

All binaries and precompiled dependencies are placed inside binaries directory, so if the installation was success 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 Figure 2.1 should be similar to your examples execution.

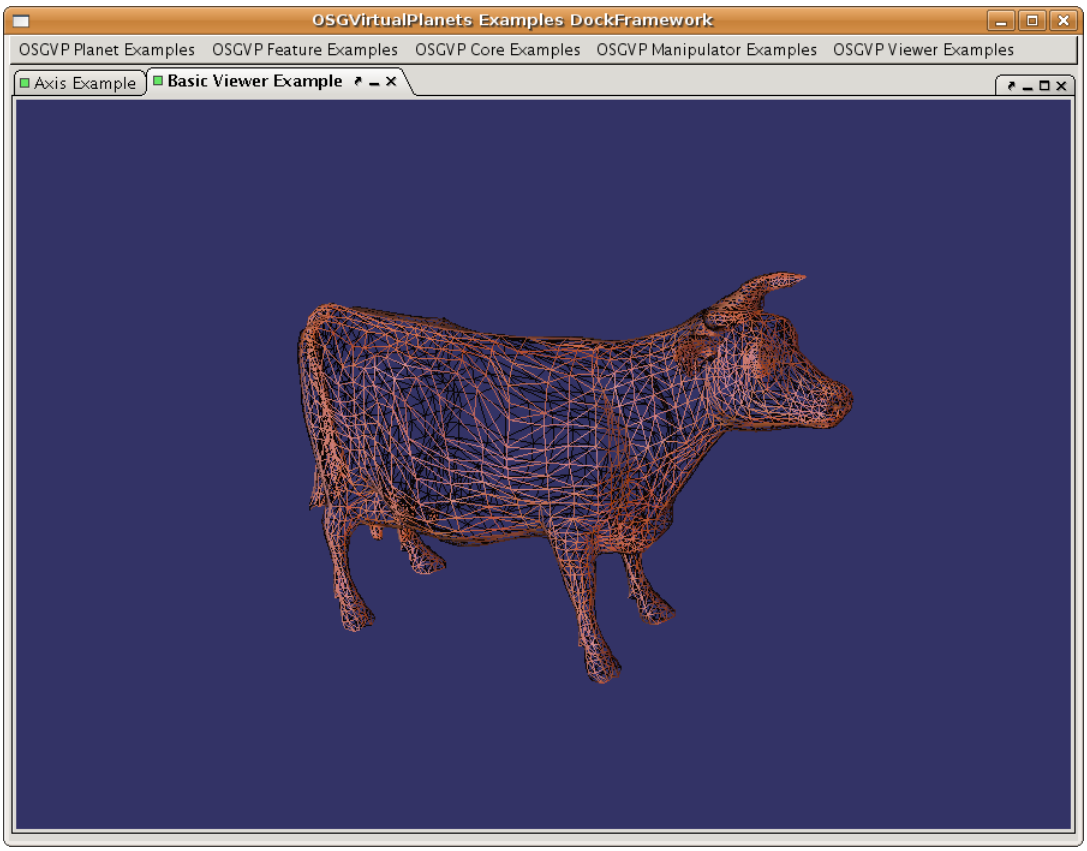


Figure 2.1: Examples framework. Running basicViewer Example in wired mode.

2.2 Configure Eclipse with osgVP

1. Create a new Java application with this simple example:

```
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.PositionAttitudeTransform;
import org.gvsig.osgvp.Vec3;
import org.gvsig.osgvp.osgDB;
import org.gvsig.osgvp.exceptions.node.NodeException;
import org.gvsig.osgvp.features.Text;
import org.gvsig.osgvp.planets.CustomTerrainManipulator;
import org.gvsig.osgvp.planets.Planet;
import org.gvsig.osgvp.planets.PlanetViewer;
import org.gvsig.osgvp.planets.CustomTerrainManipulator.
   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 Planet View Example");
    jFrame.setSize(600, 400);
    jFrame.setDefaultCloseOperation(JFrame.
EXIT_ON_CLOSE);

    /**
     * Create a planet viewer
     */
    PlanetViewer planetViewer = null;

    try {
            planetViewer = new PlanetViewer();
    } catch (NodeException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
    }

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

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

    /**
     * Add a planet to the scene data
     */
```

```
        Planet earth = null;
        try {
                earth = new Planet();
                planetViewer.addPlanet(earth);
        } catch (NodeException e2) {
                // TODO Auto-generated catch block
                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);
        planetViewer.setCamera(cam);

        /**
         * Add a cow in the north pole
         */

        double factor = earth.
getRadiusEquatorial() / 20.0;

        try {
                PositionAttitudeTransform transform =
        new PositionAttitudeTransform();
                transform.addChild(osgDB.
        readNodeFileFromResources("/cow.ive"));
                transform.setScale(new Vec3(factor,factor,
        factor));
                transform.setPosition(new Vec3(0,0,
        earth.getRadiusPolar()*1.1));
                planetViewer.addSpecialNode(transform);
        } catch (NodeException e1) {
```

```
                // TODO Auto-generated catch block
                e1.printStackTrace();
        }

        /**
         * Adding some information in the HUD
         */

        try {
                Text info = new Text();
                info.setText("Planet View Example");
                planetViewer.addNodeToHUD(info);
        } catch (NodeException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
        }

        /**
         * Customizing the manipulator
         */

        CustomTerrainManipulator manip =
(CustomTerrainManipulator) planetViewer.
getCameraManipulator();

        manip.addAzimButtonMask(MouseButtonMaskType.
LEFT_MOUSE_BUTTON, 'a');

        manip.setMinCameraDistance(earth.
getRadiusEquatorial());
        manip.setMaxCameraDistance(earth.
getRadiusEquatorial()*5.0);
        manip.setEnabledNorthOrientation(true);

        jFrame.setVisible(true);
        jFrame.addWindowListener(new WindowAdapter() {
```

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

}
```

1. Configure Java Build Path of the project. Add the jars of osgVP as external jars (~/osgVP/lib/*.jar).

2. Configure the Launcher (Open Run Dialog) 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=/home/myaccount/osgVP/binaries/linux32/lib`
     - **Windows:** `PATH=C:\osgVP\binaries\win32\bin`
     - **MacOSX:** `DYLD_LIBRARY_PATH=/home/myaccount/osgVP/binaries/mac/lib`

3. Run the new launcher, if all is ok, then you should see the image shown in the Figure 2.2.

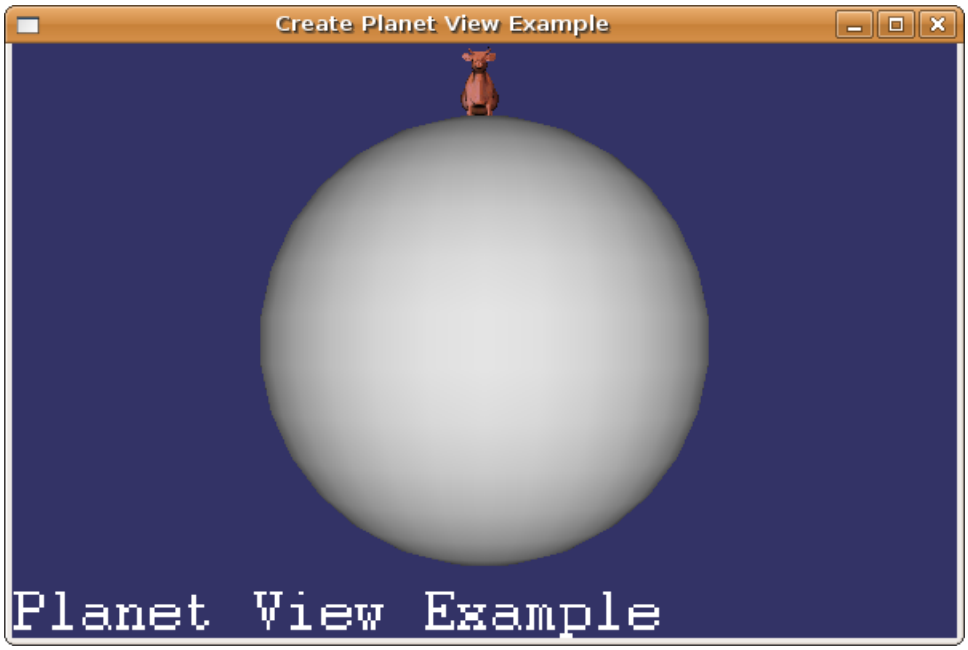


Figure 2.2: Simple example

3

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 appliying textures or materials, etc. The osgVP-core library also contains mathematical tools necessaries to manage the scenegraph.

3.1 Managing the scenegraph

Adding and/or removing nodes to the scene should be easy if you know the basics of scenegraphs. 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 neccessary know-how of scenegraphs. For more information you can look up in the Javadocs.

3.2 Loading and saving scenes

The osgVP-core library is capable to load and save scenes or nodes in the same format as OpenSceneGraphs does it. 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;
}
```

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

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

3.3 Mathematic Tools

Mathematic tools are included in this library. They are neccessary 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<T>, 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 acumulate rotations. It could be used in spherical interpolations,etc. The use of this class requires some advanced maths skills.
- EllipsoidModel: Serves to do changes between coordinate systems using an ellipsoid model to do the operations.

3.4 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 autoscaling to screen.

Let's see some code:

```
Group root= new Group();

//create node
AutoTransform at = new AutoTransform();
at.addChild(osgDB.readNodeFileFromResources("/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);
```

You can see some implementation in the example "Camera Matrices" of the examples framework.

3.5 Defining Geometries

This section covers some of the methods that can be used to create geometric primitives. There are several ways to deal with geometry objects: at the lowest level loosly wraps OpenGL primitives; an intermediate level using open scene graph basic shapes and at a higher level, loaded from files. This section covers the lowest level. This method provides the greatest flexibility and requies the most effort. Normally at the scene graph level geometry is loaded from files. Most of the effort of tracking vertices is handled by file loaders.

3.5.1 Overview

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 indices 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.

3.5.2 Geometry creation example

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 vertices 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 vertices andd 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.

3.6 StateSets

A scene graph manager traverses a scene graph to determine what geometry needs to be sent to the grapics 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 scenegraph. 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 OpenSceneGraph website or in the Javadoc of osgVP.

3.7 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
sphera.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.

3.7.1 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 BufferedImages to osgVP images. Coding first way should look like next lines.

```
try{
```

```
...
Texture2D tex = new Texture2D();
//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);
...
```

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

```
try{
...
Texture2D tex = new Texture2D();
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();
...
```

3.8 Updating a Node

Updating a Node is a very usual process in CG applications. Th scenegraph gives us the way to do it. Your main class must implement the UpdateNodeListener Interface. Implemeting 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();
        //Atention!! You must set the Listener
        ejes.setUpdateListener(this);
        ...

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

3.9 GLSL Programming

OpenGL Shading Language allows programers 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 writting shaders, but we explain how to apply these shaders to a Node of our scenegraph. 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 glShader-Object. 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 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);
```

4

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.

4.1 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 proccess. They are implemented with pbuffers. Composite Viewers are util when we need different views of the same scene.

4.2 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 *createViewer* 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 jContentPane = new JPanel();
jContentPane.setLayout(new BorderLayout());

OSGViewer Viewer = new OSGViewer();

_canvas3d = ViewerFactory.getInstance().createViewer(
        ViewerFactory.VIEWER_TYPE.CANVAS_VIEWER,
        Viewer);

jContentPane.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 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();
        }
});
```

4.3 Camera manipulators

A camera manipulator define 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 manipulator 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 manipualtor created and attached to viewer by deafult. The TrackballManipulator class receives updates of mouse events in the form of GUIEventAdapter instances.

4.4 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.

4.4.1 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);
```

4.4.2 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 acomplish 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.

4.5 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();
        ...
}
```

4.6 Printing utilities

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

```
printViewer = new OSGViewer();
```

```
printViewer.setUpViewerInBackground(0, 0, getCanvas3D().getWidth(),getCanv
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.

The class PrintUtilities makes for us this work, we can decide wich 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.

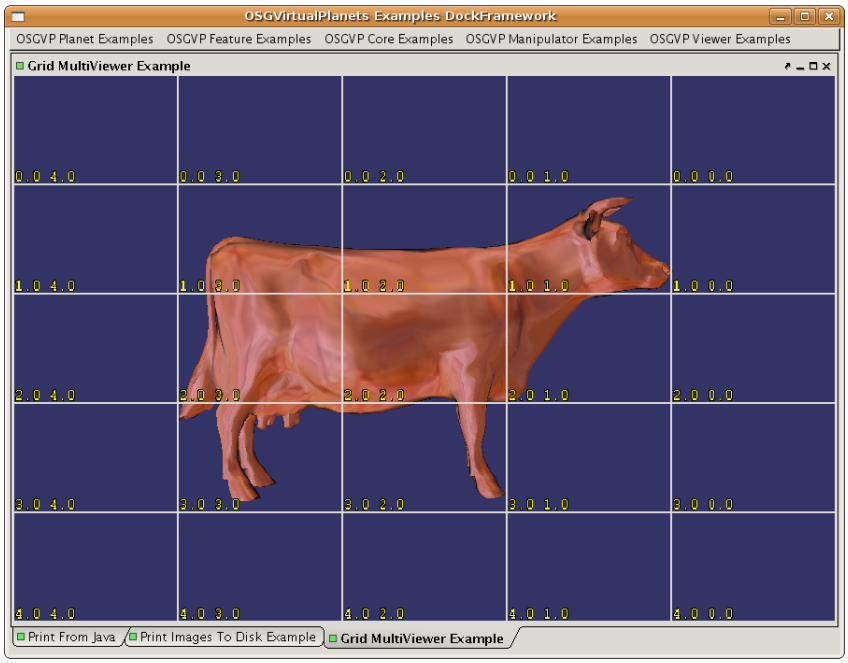


Figure 4.1: Tiled scene.

5

OSGVP Planets

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

5.1 The Planet View

Since a planet 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 planet 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 planet needs.

There is a specific *PlanetViewer* class inside the osgVP-planets 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 planet viewer for terrain visualization is highly recommended instead of the other OpenSceneGraph based viewers.

5.1.1 Create a planet viewer

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

```
private static IViewerContainer _canvas3d;
```

Later the *PlanetViewer* instance can be created and assigned to the canvas. The first parameter of the *createViewer* 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 jContentPane = new JPanel();
jContentPane.setLayout(new BorderLayout());

PlanetViewer planetViewer = new PlanetViewer();

_canvas3d = ViewerFactory.getInstance().createViewer(
        ViewerFactory.VIEWER_TYPE.CANVAS_VIEWER,
        planetViewer);

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

When a planet 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 planet viewer the scene data is composed by three void nodes: **planets, special nodes and hud nodes** with their set and get methods. The methods *setSceneData* and *getSceneData* are still available in the planet viewer but is recommended don't use them because you have to keep the structure of the scene graph. A default special manipulator for planet navigation called *CustomTerrainManipulator* 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 planet 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.

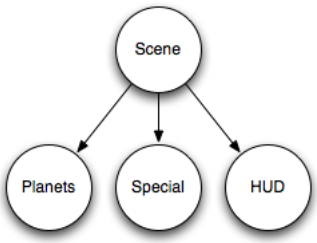
```
jFrame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
                ViewerFactory.getInstance().stopAnimator();
                _canvas3d.dispose();
        }
});
```

5.1.2 Set the scene data in a planet viewer

The scene data in a planet 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 **planets** node of the scene data because the planet viewer needs this nodes to compute some intersections with them. This intersections are used in the planet manipulator of the viewer and to compute the near and the far planes of the view.

The nodes added in the **special** 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 **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 planet with the default constructor (it builds the Earth) and add to the scene graph.

```
Planet earth = new Planet();
planetViewer.addPlanet(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);
planetViewer.setCamera(cam);
```

Now, we can put some 3D models in the planet 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.readNodeFileFromResources(
        "/cow.ive"));
transform.setScale(new Vec3(factor,factor,factor));
transform.setPosition(new Vec3(0,0,
        earth.getRadiusPolar()*1.1));

planetViewer.addSpecialNode(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("Planet View Example");
planetViewer.addNodeToHUD(info);
```

5.1.3 Using camera manipulators

A camera manipulator define how to move the camera in the scene. By default, a PlanetViewer set a *CustomTerrainManipulator*. This camera manipulator is specific for

terrain navigation. It defines three basic movements: **Zoom**, **Azimut** and **Roll**. The first one, let you to move closer or away from the planet surface. The second one, let you to change the angle of inclination and the last let you move around the planet 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 methods: *addAzimButtonMask*, *addRollButtonMask* and *addZoomButtonMask* and remove them with: *removeAzimButtonMask*, *removeRollButtonMask* and *removeZoomButtonMask*. The *mouseMask* parameter of this functions is defined by the *MouseButtonMaskType* class. For example, if you want to add a new combination of the left mouse button and the **a** key for change the **Azimut** you can use the following line:

```
manip.addAzimButtonMask(
        MouseButtonMaskType.LEFT_MOUSE_BUTTON, 'a');
```

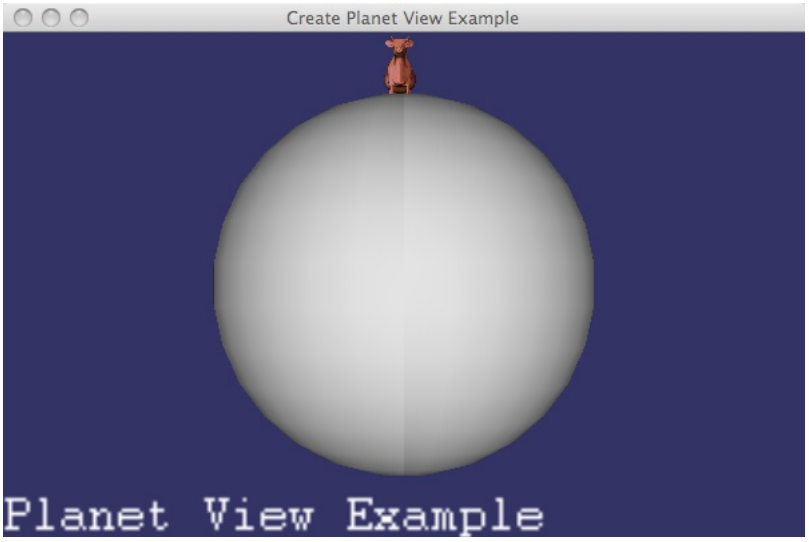
For remove the last combination you can use the remove method:

```
manip.removeAzimButtonMask(manip.getAzimButtonMaskSize()-1);
```

Aslo you can change the speed of the movement for the **Zoom** and **Roll** actions with the methods *setRollFactor* and *setZoomFactor* (there are are get methods for every set method). Furthermore, there are set and get methods for specify the minimum and maximum distance of the camera from the center of the planet: *setMinCameraDistance*, *setMaxCameraDistance*, *getMinCameraDistance* and *getMaxCameraDistance*. Finally, you have some methods to force the orientation of the planet to the North: *setEnabledNorthOrientation* or *forceNorthOrientation*.

```
manip.setMinCameraDistance(earth.getRadiusEquatorial());
manip.setMaxCameraDistance(earth.getRadiusEquatorial()*5.0);
manip.setEnabledNorthOrientation(true)
```

Finally your example should look like the following image.



5.2 Define a planet

When you create a planet 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 planets.

When you would define a planet 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.

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

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

```
Planet mars = new Planet();

mars.setPlanetName("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 differents planets. For example, you can created a projected view of the autonomous region of Valencia. First you must create a planet viewer with the methods explained in the previous section. Then you can add a new projected planet specifying the dimensions in the appropriate coordinate system and add to the planet viewer.

```
private static Planet _planet;

_planet = new Planet("Valencia",
        Planet.CoordinateSystemType.PROJECTED, "WKT",
        "EPSG:23030", 0, 4000000, 1000000, 5000000,
        6378137.0, 6356752.3142);
planetViewer.addPlanet(_planet);
```

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

```
double difx = planet.getExtent().getWidth() / 2.0d;
double dify = planet.getExtent().getHeight() / 2.0d;
double posx = planet.getExtent().getX() + difx;
double posy = planet.getExtent().getY() + 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);
planetViewer.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) {
                        _planet.setExtent(0, 4000000,
                                1000000, 5000000);
                } else if (e.getKeyCode() == KeyEvent.VK_2) {
                        _planet.setExtent(0, 4000000,
                                2000000, 5000000);
                }
        }
}});
```

Finally 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.

5.3 Layer management

The class *Planet* give you the necessary methods to afford the layer management. You can add raster layers like textures or elevations in your terrain. For add a layer, we need a planet before. You can create a default planet and add to the planet viewer with the methods explained in previous sections. Don't forget to place the camera in the correct position.

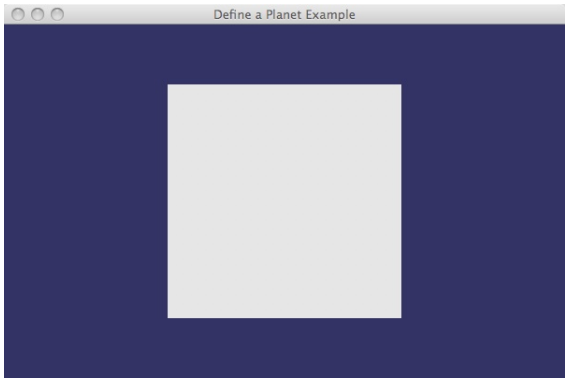


Figure 5.1: Projected planet sample with the extent of the autonomous region of Valencia.

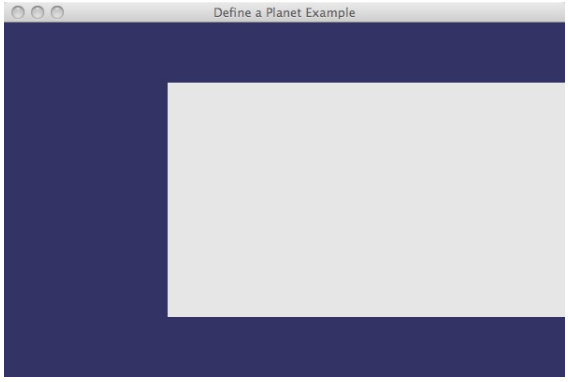


Figure 5.2: Projected planet sample after change the size of his extent.

5.3.1 Adding layers

The method *addTextureLayer* lets you to add layers to the planet. The data of this layer is used to set textures over the terrain. For example, you can add a key listener to add some texture layers when the key 1 is pressed. This methods require the extent of the layer like a *Rectangle2D* class. In a *Rectangle2D* the first and second parameters are the position of the top left corner (minimum x and y) and the third and fourth are the length of the x and y dimensions.

```
_canvas3d.addKeyListener(new KeyListener() {
        public void keyPressed(KeyEvent e) {
                if (e.getKeyCode() == KeyEvent.VK_1) {
                        Rectangle2D extent = new
                                Rectangle2D.Double(-180,
                                -90, 360, 180);
                        _planet.addTextureLayer(extent);
                }
        }
}});
```

If you would to use the raster data to change the elevation of the terrain, you must use the method *addHeightfield* instead of the other one. Add this method to the key listener.

```
if (e.getKeyCode() == KeyEvent.VK_2) {
        Rectangle2D extent = new Rectangle2D.Double(-180,
                -90, 360, 180);
        _planet.addHeightfieldLayer(extent);
}
```

Remember, when you add a layer to the planet, the planet put the layer in the top of the planet surface and his order number is the number of layers less one. The order numbers of texture and elevation has different numerations because the planet has separated lists of layer for each one.

5.3.2 Request layers

When you add a layer in the planet, you are only notifying the planet to request the layer data in other words this methods only adds the layer in the planet list of layer. Then the planet compute which regions are visible and request the necessary data for them. You must overwrite the methods provided in the *RequestLayerListener* and attach the listener to the planet. For example, you can put a test image when the planet request the layer data with the methods *setTexture* and *setHeightfield*. The first parameter of both methods is the pointer to the terrain node that request the data and the third is the number of the layer. You can get both of the *RequestLayerEvent*. The second one of the both methods is the path to the file that contains the requested data for the extension was given in the *RequestLayerEvent*

```
_planet.setRequestLayerListener(new RequestLayerListener() {

        public void requestElevationLayer(
                RequestLayerEvent evt) {
                _planet.setHeightfield(evt.getTerrainNode(),
                        "test.tiff", evt.getOrder());
        }

        public void requestTextureLayer(
                RequestLayerEvent evt) {
                _planet.setTexture(evt.getTerrainNode(),
                        "test.jpg", evt.getOrder());
        }

});
```

Alternatively, you can create a layer manager class that implements the methods of the *RequestLayerListener* and set to the planet later. Notice the importance of this methods. A planet send a event to the *RequestLayerListener* with the necessary information to give him the requested data. This information is stored int the *RequestLayerEvent*.

A *RequestLayerEvent* has methods to get some information about the layer and the terrain that requested the data.

- **getTerrainNode**: returns the pointer to the terrain node. You must use this pointer to send the data to the terrain node with the *setTexture* and *setHeightfield* methods.
- **getOrder**: returns the number of the layer. You must use to obtain the data of the correct layer and to send the data with the *setTexture* and *setHeightfield* methods.
- **getMinX, getMinY, getMaxX, getMaxY**: return the extent of the data requested. You have to implement or use some driver to get the data of this extent from remote services or local files and store in a cached file.
- **getLevel, getX, getY**: return the identifier of the terrain node. It's useful to use this identifier for the file name of the requested data to create a cache filesystem.

5.3.3 Removing layers

When a layer is no longer used you can remove it with the methods *removeTextureLayer* and *removeHeightfieldLayer*. You only have to specify the number of the layer that you would be removed. For example, you can add this methods to the key listener to remove the last layer added of each type.

```
if (e.getKeyCode() == KeyEvent.VK_3) {
        _planet.removeTextureLayer(_planet
                .getNumberLayerTexture()-1);
}

if (e.getKeyCode() == KeyEvent.VK_4) {
        _planet.removeHeightfieldLayer(_planet
                .getNumberLayerElevation()-1);
}
```

5.3.4 Reorder layers

Also you can move up and down the layers using specific methods called *reorderTextureLayer* and *reorderElevationLayer*. You must specify the actual position and the new one. For example, you can move the first layer to the top of the planet surface.

```
if (e.getKeyCode() == KeyEvent.VK_5) {
        _planet.reorderTextureLayer(0, _planet
                .getNumberLayerElevation()-1);
}

if (e.getKeyCode() == KeyEvent.VK_6) {
        _planet.reorderElevationLayer(0, _planet
                .getNumberLayerElevation()-1);
}
```

5.3.5 Visibility ranges

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

- **setMinTextureRange, setMinElevationRange**: they establish the minimum level of range necessary to see the layer.
- **setMaxTextureRange, setMaxElevationRange**: they establish the maximum level of range since the layer is no longer visible.
- **setMaxTextureResolution, setMaxElevationResolution**: 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.

The first parameter of this methods is always the order number of the layer and the second one is the level. Notice that the level is a integer that indicates the number of subdivisions of the mesh and represents the quality of the rendered data.

5.3.6 Other layer properties

There are some methods to change layer properties in the *Planet* class. The next methods are available for texture and elevations layers and you must specify the order number of the layer to apply them.

- **setEnabledTextureLayer, setEnabledHeightfieldLayer**: let you enable or disable the layer. When a layer is disabled, the planet doesn't request data for it and the layer it isn't visible. The second parameter of the method it's a boolean value set to true for enabling and false for disabling it.
- **invalidateTextureLayer, invalidateHeightfieldLayer**: you can use them to force the planet to reload all the data for the selected layer. They are useful if you are using cached files and one or more of them has been modified.

Finally, you can change the opacity only for the texture layer using the method *setTextureOpacityLayer*. The second parameter is a float between 0 and 1 and establish the transparency level. For elevations you can change the vertical exaggeration with the *setVerticalExaggerationLayer*. 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.

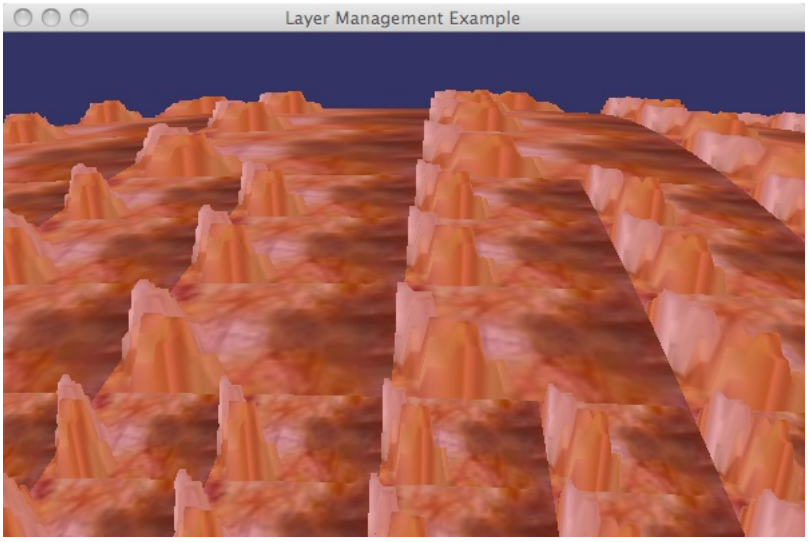
```
if (e.getKeyCode() == KeyEvent.VK_7) {
        _planet.setTextureOpacityLayer(0, 0.5f);
}

if (e.getKeyCode() == KeyEvent.VK_8) {
        _planet.setVerticalExaggerationLayer(0, 20.0f);
}
```

Finally, your example should look like the following image.

5.4 Planet utilities

The *Planet* 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 latitud, longitud 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.

- **setEnabledSkirts**: enable and disable the skirts of the subregions of the terrain. This skirts are used to prevent see holes in the terrain surface.
- **setEnabledSubdivision**: enable and disable the terrain subdivision in subregions of bigger data quality. If you disable the subdivision, you have poor data quality.
- **setSubdivisionFactor**: sets the subdivision factor of the mesh of the terrain. A higher value can collapse your system.

6

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.

6.1 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.

6.1.1 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.

6.1.2 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.

6.1.3 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 chidren 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 chid number i of the Manipulator.

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

Returns the dragger type.

6.2 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);
```

6.2.1 Example: 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.readNodeFileFromResources("/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.

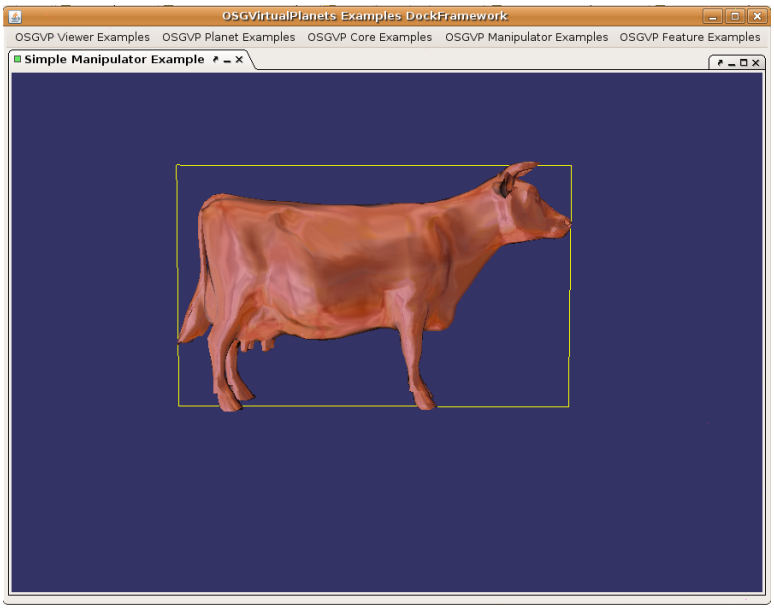


Figure 6.1: Node being Manipulated.

6.3 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);
```

6.3.1 Methods implemented by EM

```
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.

6.3.2 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 5.1:

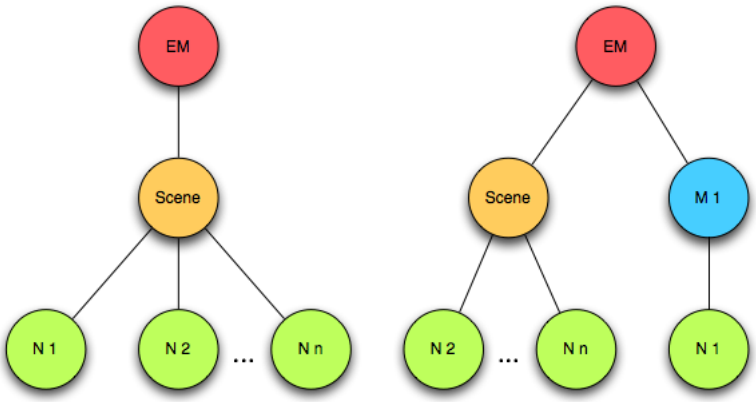


Figure 6.2: Adding a node to the EditionManager.

If the mouse is clicked and no intersections are computed, the listener makes all the manipulators dissapear. 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 5.1). 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.

6.4 The GeometryManipulator node

The Manipulator node transforms the objets 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.

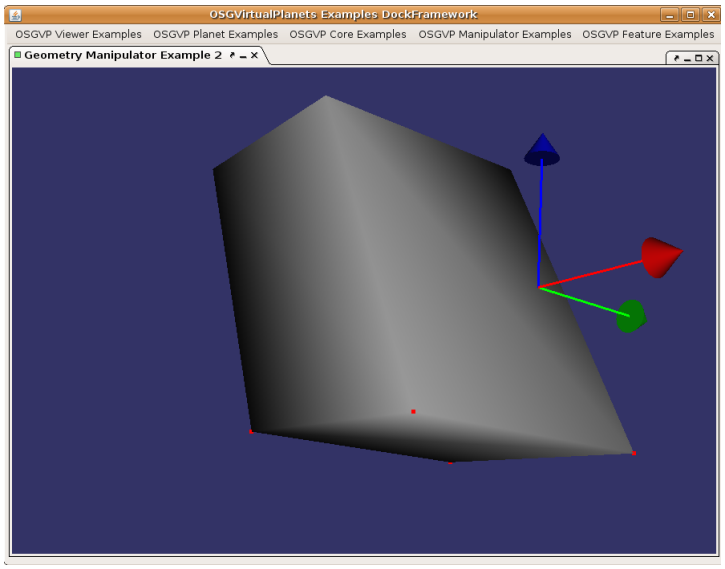


Figure 6.3: GeometryManipulator example.

7

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 vertices are too supported operations. Polylines ant PixelPoints could be antialiased.

Users can extrude simple geometric forms using different techniques. New features can be added wihtout modifying classes hierarchy.

7.1 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 implememted. 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 multipolygons. 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++.

7.2 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. dinamicly. 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.

```
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.

```
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);
```

7.3 Polylines

Drawing polylines should be easy with osgVP. The API let change the width, color, pattern or blending of the polyline. 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, Polylines 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);
```

7.4 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 vertices) 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 towards 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();
```

7.5 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 multiple 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");
        te1.setPosition(0f, 0f, 20f);
        text.setCharactherSize(3.0f);

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

7.6 Extruded Geometries

Extrusion classes in osgvpfeatures extends of OSGExtruder class, belonging osgvp-core library. This class is mapped against a generic extrusor, created in C++ from a J.Hidalgo Project(Deparment 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 determinated kind of geometry depending what was the input. See next example to view a polygon extrusion:

```
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());
```

You can look up the examples framework for further information.

8

Latest changes in OSGVirtualPlanets version 2.2

The next section explains how to migrate from osgvp 2.1.x libraries to the new version 2.2.0.

8.1 JAVA SIDE

Now, we explain the API changes in the Java side and how to use the new functionally added in the latest version.

8.1.1 OSGVPPlanets::TerrainViewer

- Rename PlanetViewer to TerrainViewer class.
- The method getScene() is deprecated. To obtain the scene you can use the getSceneData() defined in the OSGViewer.
- Rename addPlanet to addTerrain method.

- Rename removePlanet to removeTerrain method.
- Rename getPlanet to getTerrain method.
- Rename setPlanet to setTerrain method.
- Rename getPlanets to getTerrains method.
- Added setPlanets method.
- Rename setEnabledPlanets to setTerrainsEnabled method.
- Added getTerrainsEnabled method.
- Rename activePlanet to setTerrainActive method.
- Rename getActivePlanet to getTerrainActive method.
- Rename addSpecialNode to addFeature method.
- Rename removeSpecialNode to removeFeature method.
- Rename getSpecialNode to getFeature method.
- Added setFeature method.
- Rename getSpecialNodes to getFeatures method.
- Added setFeatures method.
- Rename setEnabledSpecialNodes to setFeaturesEnabled method.
- Added getFeaturesEnabled method.
- Rename addNodeToHUD to addNodeToCameraHUD method.
- Rename removeNodeFromHUD to removeNodeFromCameraHUD method.
- Rename getNodeFromHUD to getNodeFromCameraHUD method.
- Added setNodeToCameraHUD method.
- The method getHUD is deprecated, you must use the getCameraHUD method.
- Added setCameraHUD method.
- Rename setEnabledHUD to setCameraHUDEnabled method.

- Added getCameraHUDEnabled method.
- The method getCustomTerrainManipulator() is deprecated. The manipulator is getted or setted like other camera manipulators.
- Rename getOrCreatePlanetViewerHandler to getOrCreateEventHandler mehtod.

8.1.2 OSGVPPlanets::CustomCameraManipulator

Every custom camera manipulator must be implement this interface to obtain the customized keys settings.

8.1.3 OSGVPPlanets::TerrainCameraManipulator

- TerrainCameraManipulator replaces the CustomTerrainManipulator class.
- The static class MouseButtonMaskType is moved to CustomCameraManipulator interface.
- The methods to add, get or remove mouse and key masks are deprecated. Now you must implement the CustomCameraManipulator methods to let change the mouse and key masks.
- Rename the setMinCameraDistance to setMinimumDistance method.
- Rename the getMinCameraDistance to getMinimumDistance method.
- Rename the setMaxCameraDistance to setMaximumDistance method.
- Rename the getMaxCameraDistance to getMaximumDistance method.
- Rename the getNorthOrientation to getEnabledNorthOrientation method.

8.1.4 OSGVPPlanets::Terrain

- Rename PlanetNode to Terrain class.
- Rename getPlanetID to getTerrainID method.
- Rename getPlanetName to getTerrainName method.

- Rename setPlanetName to setTerrainName method.
- Rename getCSFormat to getCoordinateSystemFormat method.
- Rename setCSFormat to setCoordinateSystemFormat method.
- Rename setPlanetExtent to setExtent method.
- Rename getPlanetExtent to getExtent method.
- Rename getPlanetRadiusEquatorial to getRadiusEquatorial method.
- Rename getPlanetRadiusPolar to getRadiusPolar method.

9

Appendix

In this appendix you will find a extended compilation guide. The appendix describe the different ways to get your project working using osgVP libraries.

9.1 Compilation Requirements

Once you have dowloaded the libraries of (https://gvsig.org/web/projects/gvsig-commons/osgvp) there are several tools that you must have installed in your system. They are:

1. Subversion (Only to get development Version):
    - Linux: Depending on your distribution there are several ways to install subversion. For Ubuntu or Debian based distributions #apt-get install subversion.
    - Windows: download and install subversion command line version or TortoiseSVN from http://tortoisesvn.net
    - Mac osX: For Leopard install Developer tools from Leopard installation CD's. For Tiger download the command line version.

2. C++ Compiler:
    - Linux: Download and install g++. You can do it via apt if you use a Debian based Distribution. #apt-get install g++.
    - Windows: begginning from Visual Studio 2003 on, you can compile native packages of osgVP libraries.
    - Mac osX: Once you have installed developer tools, you are able tu compile osgVP native libraries with g++ or XCode.
3. CMake:
    - Linux: On debian based distributions #apt-get install cmake. Or download and install latest stable version from http://www.cmake.org
    - Windows: Download latest stable version from the same link than above.
    - Mac osX: Download latest stable version from the same link than above.

5. Java JDK: If you plan to use osgVP inside gvSIG you must install jdk 5.0 , but if your project is independent of gvSIG you are able to use jdk 6.0.
    - Linux: Download and install Java JDK. #apt-get install sun-javaX-jdk, where X are 5 or 6 depending on your project.
    - Windows: Download and install Java JDK from http://java.sun.com
    - Mac osX: Java JDK is installed with your mac osX Developer Tools packages.
6. Maven:
    - Linux: If you are using a debian based distribution like Ubuntu #apt-get install maven2. If not you must download the packages from http://maven.apache.org
    - Windows: Download and install maven from the link of above.
    - Mac osX: Maven comes with Developer Tools packages.

7. Ant:
    - Linux: If you are using a debian based distribution like Ubuntu #apt-get install ant.
    - Windows: Download and install ant from http://ant.apache.org.
    - Mac osX: ant comes with the basic packages of Developer Tools
8. Python (Only if you want assisted compilation and precompiled dependency management):
    - Linux: #apt-get install python
    - Windows: download and install python from http://www.python.org
    - Mac osX: python comes with clean installation of osX.

9.2 Stable Version Build Guide (osgVP-2.1.7)

The osgVP has a range of dependencies which could be managed automatically or by user. So there are two ways to compile source code, with or without assisted compilation.

9.2.1 Compiling with assisted compilation

Build Manager (BuildMan) is a set of extensible Python scripts that helps in the management of binary dependencies and automated build systems. Experts readers could ask themselves if ANT and MaVen exits why not use it?

1. Ant is a really good choice, but you need to have Java installed.
2. Ant is extensible too, but you need to do it in Java, and this implies to compile. We want quick development, and only define a simple config file or create a python script that only is installed on buildman plugins path or $HOME/.buildman/plugins.
3. Maven is useful to manage dependencies, but only for Java, and Maven is created for the same reason.. Ant is powerful but to dispatch tasks, not to manage complex dependency systems.

Working with buildman should be very easy, the procedure is:

1. Download sources from Downloads page
2. Enter to the directory osgVP
3. Execute $ant inside the directory osgVP

There are two more options to execute ANT, you should use it instead of the default if you meet the requirements:

- "ant ati" This option should be used if you are under a Linux OS and using the ATI restricted driver.
- "ant vs8" This option should be used if you are under a Windows OS and want to compile with Visual Studio 8.

The previous command execute buildman to download required native dependencies from ai2 servers, then executes CMake and prepares the Makefiles ready to compile with gcc (in Linux and Mac OS X), or prepares the projects to compile with Visual Studio 7 (in Windows. The use of VS7 is a requirement of the gvSIG project, so it tries to use the VS7 Generator with CMake) . It also prepares the Java projects that can be imported with Eclipse.

9.2.2 Executing with assited compilation

1. Once finished compilation, check if inside osgVP/binaries/{platform}/{lib/bin} directory you have the native libraries (.so or .dll or .jnilib/.dylib).
2. Open Eclipse and select as workspace the directory osgVP.
3. Import projects with eclipse assistant from the osgVP directory.
4. Open the launcher assistant with Open Run Dialog inside Run Menu or Run As bar button
5. Create a new Java Application Launcher:
    - Name: ExamplesLauncher
    - Project: osgvp-examples
    - Main Class: org.gvsig.ExamplesLauncher

Before Launch, go to the Environment Tab Button and add a new Environment Variable:

- **Windows:**
  ```
  PATH=${workspace_loc}\binaries\win32\bin;
       C:\documents and Settings\{username}\depman\bin
  ```
- **Linux:**
  ```
  LD_LIBRARY_PATH=${workspace_loc}/binaries/linux32/lib:
       /home/{username}/.depman/lib
  ```
- **MacOsX:**
  ```
  DYLD_LIBRARY_PATH=${workspace_loc}/binaries/mac/lib:
       /Developer/DepMan/lib
  ```

Run the application, and if all is ok you should see the examples framework.

9.2.3 Compile without assisted compilation

If you rather work without assisted compilation, you must set up your project with native dependencies installed correctly. A priori, the libraries you need are: OpenSceneGraph-2.2, jogl and gdal.

1. Download sources from Downloads page.
2. Enter to the directory osgVP/libosgvp/libjni-osgvp
3. Execute CMake and solve native dependencies stuff. Cmake will ask you for them.
4. Compile calling install target (make install or in VS/XCode selecting INSTALL to compile)
5. Once finished compilation, check if inside osgVP/binaries/{platform}/{lib/bin} directory you have the native libraries (.so or .dll or .jnilib/.dylib).
6. Enter to the directory osgVP/
7. Prepare eclipse workspace and eclipse projects:
   - $ mvn install -Dmaven.test.skip=true
   - $ mvn eclipse:eclipse
   - $ mvn eclipse:add-maven-repo -Declipse.workspace="."

8. To Develop with Eclipse Plugins:
    - $ cd eclipse-plugins
    - $ mvn eclipse:eclipse
9. To Develop with Eclipse RCPs:
    - $ cd applications/eclipse-applications
    - $ mvn eclipse:eclipse

9.2.4 Executing without assited compilation

Configure de launcher exactly equals to assited compilation except for environment variables, where OSG_DIR is the path to your compiled version of OpenSceneGraph-2.2.0, GDAL_DIR is the path to your installation of GDAL (required by OSG) and finally JOGL_DIR is the installation path of jogl.

- **Windows:**

```
PATH=${workspace_loc}\binaries\win32\bin;
     %OSG_DIR%\bin:%GDAL_DIR}%\bin:%JOGL_DIR%\bin
```

- **Linux:**

```
LD_LIBRARY_PATH=${workspace_loc}/binaries/linux32/lib:
     ${OSG_DIR}/lib:${GDAL_DIR}/lib:${JOGL_DIR}/lib
```

- **MacOsX:**

```
DYLD_LIBRARY_PATH=${workspace_loc}/binaries/mac/lib:
     ${OSG_DIR}/lib:${GDAL_DIR}/lib:${JOGL_DIR}/lib
```

Run the application, and if all is ok you should see the examples framework.