Modifying TraplasGUI

Herbert de Vos, Willem Drost

June 19, 2006

Contents

1 Introduction **3**

2 Using traplasGUI **3**
2.1 keybindings and buttons . . . 3

3 Adding messages **4**
3.1 FileReader . . . 4
3.2 DbusConnection . . . 4
3.3 Dispatcher . . . 4
3.4 StateControl . . . 5

4 Adding communication modes **5**
4.1 Comm . . . 5
4.2 MessageController . . . 5
4.3 Dispatcher . . . 5
4.4 traplasGUI.cpp . . . 5

5 OpenSceneGraph **6**

6 Models **7**
6.1 Mappings . . . 7
6.2 Adding models . . . 7

A The osg data structure in StateControl **8**
A.1 Visualisation . . . 8
A.2 HUD menu . . . 9

1 Introduction

This is a short manual on how to use and modify the traplasGUI application. It also contain some pointers where certain functionality can best be added.

For problems installing D-Bus or OpenSceneGraph it might be usefull to check #dbus and #openscenegraph at irc.freenode.net.

2 Using traplasGUI

Generating documentation

Call `make` in the root of the traplasGUI directory structure. For latex documentation, call `make` in the directory of the document you wish to generate.

compilation

Call `make` in the `src\` directory

The application comes with a commandline help. just type `traplasGUI -h`. If communications using dbus is selected, the application should be started before traplas.

2.1 keybindings and buttons

Spacebar: Resets the camera to the default position, looking straight down on the visualised transport network. At the moment it is necessary to reset the camera at the start of the visualisation.

'p': The 'p' key will pause/unpause the simulation.

'Esc': Exits the visualisation.

Quit button: Exits the visualisation.

Pause button: Pauses/unpauses the visualisation.

- button: Decreases the speed of the visualisation.

+ button: Increases the speed of the visualisation.

Middle mouse button: Selects/deselects object.

Middle mouse button + dragging: Swivels the camera.

Left mouse button + dragging: Rotates the camera around the visualised transport network.

Right mouse button + draggin up/down: Zoom out/in.

3 Adding messages

To add a message to traplasGUI there are a few places that need to be modified. Starting at the communications side, we will walk through the classes that need modification in the order of the control flow.

3.1 FileReader

The `FileReader` class offers two methods of reading a file. A 'dumb' reader that will convert any line in a file to a vector containing string pointers and passes that on to message control, and a parser that is based on flex. The dumb reader does not need modifications as it will read practically anything.

The flex based reader however will have to be given the syntax of the message. If a message matches the syntax given in `src/Messages.flex`, the function `send(char*)` is called. To add a message or change the signature of a message, `Messages.flex` will have to be modified.

The syntax of `Messages.flex` is pretty straightforward. It can contain regular expressions matching the message followed by the action taken when that message is matched (in this case, invocation of `send(char*)` with the matched char* as its argument).

3.2 DbusConnection

`DbusConnection` only receives a string array and creates a vector containing string pointers that is placed in the queue of `MessageController`. It needs no modification to add messages.

3.3 Dispatcher

`Dispatcher` parses and construct messages. It matches incoming messages to several private functions that convert the vector containing strings to the datatypes that are used as parameters for the corresponding functions in `StateControl`, and calls the corresponding function.

To add a parsing function, it is neccesary to add the message identifier of the new message to `Dispatcher::dispatchMessage`.

For outgoing messages a function can be added to `Dispatcher()` to construct a vector with string pointers based on its parameters. See for example `Dispatcher::order()`.

3.4 StateControl

Here, the state of the osg representation is kept. The parse functions from `Dispatcher` call functions here. If a new message and parse function is made, simply add a function to `StateControl` that can be called. See also appendix A for the way the osg datastructure and the traplasobjects are organised.

4 Adding communication modes

At the moment there are two modes in which traplasGUI can work. In one it reads from file, either by means of the dumb file reader or flex parser, or it communicates with Traplas over DBus. It is however possible to add extra communication modes to the application.

4.1 Comm

`Comm` is a prototype for communications classes. Any new communications class should inherit from it.

4.2 MessageController

`MessageController` maintains references to the currently used communications class. To add a new type of communications, an extra instance of the function `MessageController::setCommunicationMode()` can be added or one of the old ones modified.

4.3 Dispatcher

Pretty much the same as the above. `Dispatcher` gets the selected communications mode from the `main` function and passes it on to `MessageController`.

4.4 traplasGUI.cpp

`traplasGUI.cpp` contains the main function and a parser for command line argument. The method of communication is also given as a command line argument so the parse function int `traplasGUI.cpp` will need to be adapted if the mode is to be accesible from the command line.

5 OpenSceneGraph

For examples on OpenSceneGraph: there are examples for download at the osg website, www.OpenSceneGraph.org. They are also included with the sources for the installer.

On http://www.nps.navy.mil/cs/sullivan/osgtutorials/ are also several OpenScenGraph tutorials. It should be noted however that most of these tutorials are for osg version <1.0. The `osg::CameraNode` used for picking has only recently been added and is not part of these tutorials.

For help with osg it may be usefull to check out #OpenSceneGraph on irc.freenode.net.

6 Models

Right now there are 5 tiletypes and two models that make up the simulation. In the future this will probably be expanded to cover a greater variety of transport networks and situations.

6.1 Mappings

The models that are loaded in response to a LOCATION, TRNEW or NEW-CARG message can be found in the models directory. The mapping that specifies what model should be loaded on which action can be found in the `location`, `transport` and `cargo` text files in the root directory of traplasGUI. These the path and names of models that can be loaded.

transport and cargo files

Both `transport` and `cargo` contain the path and name of a model relative to the location of the mapping files themselves. Separated from that by a space there is the scaling of the model. This deteremines how large the object wil appear in the visualisation.

location file

The `location` file starts with the scaling factor for the osg models. Each line after that is a mapping containing an integer, denoting the tiletype, and the path to the model in same format as the transport and cargo models.

The relation between the tiletype number and the models is given in appendix A of the Interface document.

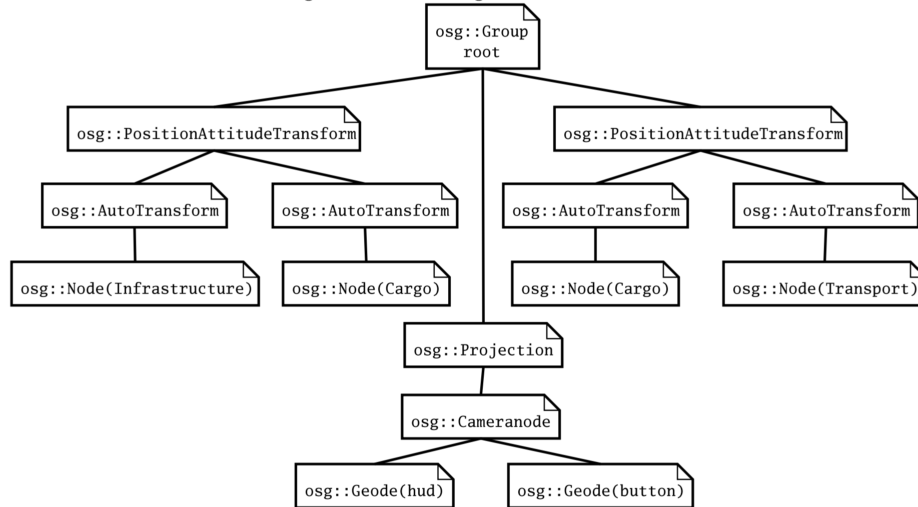
6.2 Adding models

Included in the models directory is an archive named `osgexporter-2.37.tar`. This is an osg model exporter plugin for blender (www.blender3d.org) that allows creation of osg models for usage in the visualisation. N.B.: All the models that are currently used have the convention that the positive y-axis is 'down'.

It is also possible to edit the osg files using a text editor.

A The osg data structure in StateControl

Figure 1: The osg data structure



The osg structure that is managed by `StateControl` consists of two major parts. The first one is the visualisation part that contains the representations of the objects in traplas. The second one is the HUD containing the menu. Both are contained in an osg::Group called osgRoot, which is an attribute of `StateControl`.

A.1 Visualisation

Each object in traplas is represented by a TraplasObject and an `osg::Node`. The TraplasObject contains properties such as speed, loaded cargo, description of the object, etc. The osg::Node contains a model that is used for visualising the TraplasObject. These are all grouped in `StateControl::osgRoot`. They have as parents an `osg::AutoTransform` for scaling and an `osg::PositionAttitudeTransform` for their position in the world. Both infrastructure resources and transports are placed under the osg-root. Cargo has its own `osg::AutoTransform` which is placed under the `osg::PositionAttitudeTransform` of an infrastructure resource of a transport depending on its location.

Animation

Each `osg::PositionAttitudeTransform` can be used for animation by means of `osg::AnimationPath`. This is a kind of interpolator. An `osg::AnimationPath` is created and control points are added to it, together with the time at which the moving object should be at this control point.

`osg::AnimationPath`s are created in `StateControl::drv()`. Here, they are added to an instance of `AnimationPathTimedCallback`, which is given the correct paramters for the speed and timeoffset of the animation. The instance of `AnimationPathTimedCallback` is then added to the `osg::PositionAttitudeTransform` associated with the transport resource that is involved in the drive action. Each time a frame is drawn the `AnimationPathTimedCallback` updates the position of the node it is associated with.

Each TraplasObject also has a label describing the object in the visualisation. This label is also child of the `osg::AutoTransform`.

A.2 HUD menu

The HUD and menu are managed by the `UserInterface` class. The HUD and menus are children of an `osg::CameraNode` that is separate from the world to enable picking in both the hud and the visualisation at the same time. This node has an `osg::Projection` as parent and this parent is also part of the `osg::Group` osgRoot.