INVRS

interactive networked virtual reality system

–

DRAFT: WRAPPING FUNCTIONALTY

Roland Landertshamer and Christoph Anthes

July 3, 2009

Contents

Chapter 1

Writing inVRs applications

1.1 Wrapping Functionality

In general it is possible to develop an *inVRs* application as described in the first tutorial – the *Medieval Town Tutorial*. Much of the code developed in that tutorial is generic and is already available in a so called `ApplicationBase` which is part of the *inVRs* SystemCore. The application developer can derive from this class to spare the implementation of the generic code parts. Besides the general existing `ApplicationBase` also a scene graph specific application base the `OpenSGApplicationBase` is available as an additional tool. This class will be used in this tutorial as a basis.

1.1.1 Using the ApplicationBase

Before we will start with the tutorial let's have a look at the generic `ApplicationBase` class. The key functions which are already provided in the `ApplicationBase` class are described in the following:

- `virtual bool preInit(const CommandLineArgumentWrapper& args)`

  This method is called before the initialization of the application base. It is the first method called in an *inVRs* application after the constructors are executed. The application developer can overwrite this method to insert code which has to be executed before all other parts of the application.

- `virtual void initCoreComponentCallback(CoreComponents comp)`

  This method is called by the `SystemCore` when the components of this class are initialized. It can be overwritten by the application developer to get notifications before each component is initialized.

- `virtual void initInputInterfaceCallback(ModuleInterface* moduleInterface)`

  This method is called by the `InputInterface` when the modules of this class are initialized. It can be overwritten by the application developer to get notifications before each module of the `InputInterface` is initialized.

- `virtual void initOutputInterfaceCallback(ModuleInterface* moduleInterface)`

  This method is called by the `OutputInterface` when the modules of this class are initialized. It can be overwritten by the application developer to get notifications before each module of the `OutputInterface` is initialized.

- `virtual void initModuleCallback(ModuleInterface* module)`

  This method is called by the `SystemCore` when the general modules are initialized. It can be overwritten by the application developer to get notifications before each module is initialized.

Other functions have to be implemented by the application developer:

- `virtual std::string getConfigFile(const CommandLineArgumentWrapper& args)`

  The main configuration file has to be passed to *inVRs*. It is typically loaded from a fixed path or passed as a command line argument.

- `virtual bool init(const CommandLineArgumentWrapper& args)`

  The method is called after the initialization of all *inVRs* components, interfaces and modules. It is intended to be used by the application developer for the initialization of the main application.

- `virtual void run()`

  This method is called by *inVRs* after all initialization steps were finished and the runnable components like the `EventManager` and the `Network` module were started. In this method the application developer should implement the main application loop. The implemented main loop has to call the `ApplicationBase::globalUpdate()` method every loop iteration in order to update the *inVRs* components.

- `virtual void display(float dt)`

  This method is called by the `ApplicationBase::globalUpdate()` method in order to update the main application. The application developer should prefer this method for updates compared to the implementation in the main loop because some important *inVRs* updates like the `Controller` or the `TransformationManager` were executed before this method is called.

- `virtual void cleanup()`

  This method should be implemented in order to clean up the application. It is called by the `ApplicationBase::globalCleanup()` method.

Other methods which have to be called by the application developer:

- `bool start(int argc, char** argv)`

  This method starts the application. It should be called out of the main method after an instance of the application object was created.

- `void globalUpdate()`

  The method updates the *inVRs* components and forwards the update-command to the inherited `display` method. It must be called out of the application main loop.

- `void globalCleanup()`

  The method cleans up the *inVRs* components. It must be called by the application before finishing. This method automatically forwards the command to the inherited `cleanup` method with which the user can clean up the main application.

Besides the provided methods several member variables can be used when inheriting from the `ApplicationBase`:

- `SceneGraphInterface* sceneGraphInterface`

  This member variable points to the used SceneGraphInterface. If no SceneGraphInterface is used the pointer value is NULL.

- `ControllerManagerInterface* controllerManager`

  This member variable is a pointer to the ControllerManager. If no ControllerManager is used the pointer value is NULL.

- `NetworkInterface* networkModule`

  This member variable points to the Network module. If no Network module is loaded the pointer valus is NULL.

- `NavigationInterface* navigationModule`

  This member variable points to the Navigation module. If no Navigation module is loaded the pointer points to NULL.

- `InteractionInterface* interactionModule`

  This member variable points to the Interaction module. If no Interaction module is loaded the pointer points to NULL.

- `User* localUser`

  This variable points to the User object for the local user.

- `User* localUser`

  This variable points to the User object for the local user.

- `CameraTransformation* activeCamera`

  This variable is a pointer to the camera transformation object of the local camera.

Using the `ApplicationBase` class allows for developing applications without having to care about the main *inVRs* components. Although this reduces the lines of code which have to be written there is still much to implement, e.g. for the window management, or the display methods for rendering the scene.

1.1.2 Using the OpenSGApplicationBase

To further simplify the application development the `OpenSGApplicationBase` was developed. This class is inherited from the basic `ApplicationBase` class and implements additional functionality for window management, rendering and input device support. In this helper class the decision whether immersive displays with the help of the CAVESceneManager are used for output or a simple GLUT window is used is taken.
The key functions which are already provided in the `OpenSGApplicationBase` or it's derived classes are described in the following:

- `virtual bool preInitialize(const CommandLineArgumentWrapper& args)`

  This method is called before the initialization of the application base. It is the first method called in an *inVRs* application right after OpenSG was initialized (via `osgInit()`). The application developer can overwrite this method to insert code which has to be executed before all other parts of the application. **NOTE**: Take care to not confuse this method with the `ApplicationBase::preInit()` method, since this one is implemented by the `OpenSGApplicationBase` and must NOT be overwritten!

- `virtual void initCoreComponentCallback(CoreComponents comp)`

  This method is called by the `SystemCore` when the components of this class are initialized. It can be overwritten by the application developer to get notifications before each component is initialized.

- `virtual void initInputInterfaceCallback(ModuleInterface* moduleInterface)`

  This method is called by the `InputInterface` when the modules of this class are initialized. It can be overwritten by the application developer to get notifications before each module of the `InputInterface` is initialized.

- `virtual void initOutputInterfaceCallback(ModuleInterface* moduleInterface)`

  This method is called by the `OutputInterface` when the modules of this class are initialized. It can be overwritten by the application developer to get notifications before each module of the `OutputInterface` is initialized.

- `virtual void initModuleCallback(ModuleInterface* module)`

  This method is called by the `SystemCore` when the general modules are initialized. It can be overwritten by the application developer to get notifications before each module is initialized.

- `virtual void cbGlutSetWindowSize(int w, int h)`

  This method is called whenever the size of the window is changed. The application developer can overwrite this functions if this information is needed by the application.

- `virtual void cbGlutMouse(int button, int state, int x, int y)`

  This method is called whenever a mouse button is pressed inside the application window. It can be overwritten to get notifications for mouse button presses. Note that it is recommended to get this information via the *inVRs* `ControllerManager` using the `GlutMouseDevice` instead!

- `virtual void cbGlutMouseMove(int x, int y)`

  This method is called whenever the mouse cursor is moved over the application window. It can be overwritten to get notifications for mouse motion. Note that it is recommended to get this information via the *inVRs* `ControllerManager` using the `GlutMouseDevice` instead!

- `virtual void cbGlutKeyboard(unsigned char k, int x, int y)`

  This method is called whenever a keyboard key is pressed. It can be overwritten to get notifications for keyboard input. Note that it is recommended to get this information via the *inVRs* `ControllerManager` using the `GlutKeyboardDevice` class.

- `virtual void cbGlutKeyboardUp(unsigned char k, int x, int y)`

  This method is called whenever a keyboard key is released. It can be overwritten to get notifications for keyboard input. Note that it is recommended to get this information via the *inVRs* `ControllerManager` using the `GlutKeyboardDevice` class.

Other functions have to be implemented by the application developer:

- `virtual std::string getConfigFile(const CommandLineArgumentWrapper& args)`

  This method must return the url to the main configuration file (usually called `general.xml`). This file is needed by *inVRs* in order to load and configure the core components, interfaces and modules. If the url to the configuration file should be passed via command line then this value can be obtained from the `CommandLineArgumentWrapper` parameter.

- `virtual bool initialize(const CommandLineArgumentWrapper& args)`

  This method can be used in order to initialize the application. The method is called after the *inVRs* components (core, interfaces and modules) were initialized. The parameter of type `CommandLineArgumentWrapper` can be used to read options passed via the command line.

- `virtual void display(float dt)`

  This method is called every application loop cycle and can be used to update the application. The paramater `dt` contains the elapsed time in milliseconds since the previous method call.

- `virtual void cleanup()`

  The method should be used in order to cleanup the application. It is called right before the application is terminated.

Other methods which must be called by the application developer:

- `void setRootNode(NodePtr root)`

  In this method the root node of the scenegraph is set in the used `SceneManager`. Depending on the configuration this is either the `SimpleSceneManager` or the `CAVESceneManager`.

Other methods which can be called by the application developer:

- `void setPhysicalToWorldScale(float scale)`

  This method is important when writing applications for VR installations like a CAVE. By calling this method you can set the scale-factor from the physical units provided by your tracking system to the world units used in the application. For example if your tracking systems provides centimeter values and your application is modeled in meters then you should pass 0.01 to this method. Don't forget to call this method when using tracking systems in order to provide a correct visualization.

- `void setNearClippingPlane(float nearPlane)`

  This method allows to set the near clipping plane of your application.

- `void setFarClippingPlane(float farPlane)`

  The method allows to set the far clipping plane of your application.

- `void setStatistics(bool onOff)`

  This method activates or deactivates the display of the `SceneManager` specific statistics.

- `void setWireframe(bool onOff)`

  This method allows to switch between normal and wireframe rendering.

- `void setHeadlight(bool onOff)`

  By calling this method the headlight (default light used in OpenSG) can be activated or deactivated.

- `bool setBackgroundImage(std::string imageUrl, int windowIndex = -1)`

  This method sets the background image for the windows with the passed index. When no window index is passed the image is used as background for all active windows. **NOTE:** Using an image background in OpenSG may reduce the overall performance of your application. Avoid using this method or only use it with low resolution images!

- `void setEyeSeparation(float eyeSeparation)`

  This method allows to set the eye separation when using the `CAVESceneManager` for output.

- `float getEyeSeparation()`

  The method returns the current eye separation.

Besides the provided methods several member variables which are inherited from the `ApplicationBase` class can be used:

- `SceneGraphInterface* sceneGraphInterface`

  This member variable points to the used SceneGraphInterface. If no SceneGraphInterface is used the pointer value is NULL.

- `ControllerManagerInterface* controllerManager`

  This member variable is a pointer to the ControllerManager. If no ControllerManager is used the pointer value is NULL.

- `NetworkInterface* networkModule`

  This member variable points to the Network module. If no Network module is loaded the pointer valus is NULL.

- `NavigationInterface* navigationModule`

  This member variable points to the Navigation module. If no Navigation module is loaded the pointer points to NULL.

- `InteractionInterface* interactionModule`

  This member variable points to the Interaction module. If no Interaction module is loaded the pointer points to NULL.

- `User* localUser`

  This variable points to the User object for the local user.

- `User* localUser`

  This variable points to the User object for the local user.

- `CameraTransformation* activeCamera`

  This variable is a pointer to the camera transformation object of the local camera.

In order to write your own *inVRs* application using OpenSG your application should inherit from the `OpenSGApplicationBase`.