Date: 26.11.02

Author: R. Glaschick C-LAB Paderborn

5

Subject: glGUI widgets as AMIRE components

glGUI structure

glGUI has a separate rendering interface that can and must be tailored to the application environment. So there is the required flexibility to integrate it into many applications. The example rendering modules are a good basis.

However, glGUI is a subsystem on its own, designed to structure the GUI widgets. It

- maintains a database of the GUI elementes (widgets)

- receives keyboard and mouse events

- determines which widget is concerned

- if a widget is hit, a callback tells its number

glGUI uses a single central object of the class CUI_UI, which provides the interfaces for keyboard and mouse events and triggers the callback event, if a widget is hit.

AMIRE framework relations

In the AMIRE framework, this should be wrapped by a single invisible component, that contains the reference to this CUI_UI object. This component will receive all keyboard and mouse events.

A first version could register itself for keyboard an mouse events and neither use in- nor outslots.

In the final version, another component should pick all keyboard events and send them via outslot to whatever component would like to hear about it, in particlular to the main glGUI component.

This main CUI_UI object contains a simple registry for colors, textures and the like, used in creating and modifying widgets.

Widget components

Each widget is a new component. This is clearly necessary for visible objects, e.g. a button. Once created, it has to be attributed and linked into the whole UI structure. <u>So we have a 1-to-1 correspondence; each UI object will have a corresponding component that creates the</u>

5   <u>UI object and stores its reference in its private variables</u>.

For setting attributes, several interfaces to the registry in the main UI object are necessary, e.g. to obtain a reference to a color shader. Changing a color means:

- to change the name of the color as a property of the component,

- to lookup the corresponding shader object in the registry of the main object,

10   - to register this shader object with the glGUI button object.

So each GUI component needs at least a link to the main UI object. Therefor, it should keep this reference in a private pointer variable, set on creation. As it will never change, a mandatory parameter of the constructor call is the right solution. The authoring should create the main UI object and pass its reference to the main UI (invisible) component; this

15   is the only one that does not create the UI object itself.

The user interface at a certain state of the application consists of a number of UI objects linked together in a widget tree. This is done via an interface of the parent UI object to register the dependent UI objects.

Unless the authoring system itself keeps track of the components, each component needs

20   an additional interface for this purpose:

- The dependet component has a method to link it to a container widget, i.e. the opposite direction as in the UI. The call of this method has a reference to the container component. With this, the dependent component uses its privat reference to its own UI object to call a method in the container component (given via parameter) which relays

25   the call to its own UI object.

Alternatively, the pointer to the corresponding UI object could be public, and the authoring system must do the call.

Keyboard and Mouse actions

This version of glGUI uses a global callback routine to signal any hit of a keyboard or mouse

30   event for a widget, delivering a widget code number. In the AMIRE concept, the widget's component should send a message via an outslot once the widget is hit. To achieve this, the main component

- registers itself to obtain keyboard and mouse events,

- passes these to its main UI object,

- this main UI object will call a method of the main component to tell if and what UI object is hit; this will be noted,

5 - on return from the main UI objects keyboard event method, the marker of the previous step will be inspected,

- if a hit occured, on return to the framework a hit with distance 0 will be signalled,

- the framework, after having processed all other components this way, might call back the main component to tell that the hit was real,

10 - the main component will call the hit component,

- the hit component will send a message via its outslot.

Rendering interface

In the AMIRE framework, each component is called via its display methos to render itself. The glGUI widget components do not use this callback; all is done by the main component.

15 It was called 'invisible' above, as it does not occur as a widget on the user's screen. In reality, it makes all UI widgets visible by calling the rendering chain in its display callback. This rendering method of the main UI object recursively decends the widget tree and renders each widget, then returns.

In the glGUI architecture, the rendering of all widgets calls an interface, that must be

20 provided by the application. This interface has a rather simple functionality: It can render a rectangular box somewhere on the screen with either a solid color or a texture. The latter is used for text: each character is a texture in a small box of a character's size. This allows to have semi-transparent text, if so desired and supported by the image format used. It is, however, rather costly.

25 The renderer interface used by glGUI is an object itself, registered with the main UI object. In the current example programs, some UI widgets use methods of this object to have textures loaded. So they must be able to access it.

It folllows that each UI widget component creator has a second parameter, namely a reference to the renderer object.

30 The image loader in the original version of glGUI is self-contained and loads PNG files only. For the purpose of AMIRE, the framework's image loader will be used.

XML loader and stored UI trees

glGUI in its distributed version uses XERCES to save and load a widget tree in XML format. The AMIRE authoring system will have its own subsystem to keep track of the information needed to re-create the objects from scratch, save this list and use it to load a stored

5   application. So, the glGUI loader wil not be used an can be removed from the AMIRE version of glGUI.

Authoring considerations

The structure described above needs a special treatment of UI components by the authoring system:

10 -   A main UI object and a renderer object have to be created and a reference to both passed at each creation of a new UI component.

-   A widget in a container, i.e. a button in a panel, has to be informed of this relation by a corresponding method call.

glGUI state

15 It should be noted that the CVS develpment tree is rather different from the snapshot used in AMIRE.

Framework issues

In the current snapshot examples, the components are all created in advance before the framework is started. Because not openGL window is open, no all openGL functions work as

20 expected. Currently, this means that all UI components must postpone all calls to the renderer.

Currently, even the functional callback has not yet assured that the final openGL window is open.

So, at least temporarily, the occluder callback might be misused for a delayed setup, as the

25 GUI does not need the occluder function. Further investigation is needed.