UI Components of the VizIR Framework
Markus Raab (raab@ims.tuwien.ac.at)
2004-02-09

Abstract
This document describes the user interface components of the VizIR Framework. It gives a general view of the architecture used and shows the creation of user interfaces in the VizIR framework.

Table of Contents
Architecture of the UI-Components........................................................................................................ 2
How to create a Querying3D Panel............................................................................................................ 4
Adding media objects ................................................................................................................................ 7
Interact with media objects ..................................................................................................................... 9
How to implement a new Media Renderer ............................................................................................... 9
  Generate texture images of the media object ......................................................................................... 10
  Draw the representation of the media object ......................................................................................... 10
Context menu ............................................................................................................................................ 10
**Architecture of the UI-Components**

The central element of the VizIR user interfaces is a three-dimensional panel for the visualization of retrieval results. The technology used for this panel is GL4Java. GL4Java maps the complete OpenGL 1.2 API and the complete GLU 1.2 API to Java and implements all window handle functions (native and java) by using the Java Native Interface (JNI) and the JDirect-Interface of MS-JVM. For more information on GL4Java (including installation and setup) see [www.jausoft.com](http://www.jausoft.com).

---

**Figure 1- Architecture of the UI components of the VizIR framework**

Querying3DPanel represents the visual component of the 3D-panel. The OpenGL context is created in this class. Firstly, the scene is shown with default color settings and default perspective. Every Querying3DPanel has an instance of Querying3DPanelProperty. This class stores information on the UI panel look and feel (colors, perspective, etc.). Two handler classes are responsible for interaction with the 3D panel:

- **Querying3DPanelKeyHandler** is mainly used to move the camera. The key setup is described below.
- **Querying3DPanelMouseHandler** is used to interact with objects that are drawn on the 3D panel. This handler implements the Querying3DPanelMouseListener interface. The event class Querying3DPanelMouseEvent is used to exchange the information between the panel and the handler class.
See below for more information on handling (section “Interact with media objects”).

MediaRenderer objects are used to visualise media objects in the 3D environment. Multiple media renderer implementations may exist for each type of media. Each renderer has to extend the class MediaRenderer. In the current version of the VizIR framework, there are two implementations of media renderer: an ImageRenderer and a simple VideoRenderer. This simple video renderer is just an image renderer of the first frame of a video clip.

In the user interface, MediaRenderer objects can be combined to a MediaRendererGroup. The properties of MediaRenderer and MediaRendererGroup objects are encapsulated in MediaRendererProperty and MediaRendererGroupProperty classes.

Figure 2 - UML diagram of Querying3DPanel and default handler
How to create a Querying3D Panel

As mentioned above, we use GL4Java to create the 3D panel. Therefore, it is necessary to install GL4Java before the UI components can be used. Installation instructions and a convenient web-installer can be found at [www.jausoft.com](http://www.jausoft.com).

An instance of Querying3DPanel can be placed in a container in the same way as any other java component class. The following example shows how to create a simple 3D panel with dimension 200x200 pixel and default parameters. The 3D panel is then placed on a JPanel.

```java
Querying3DPanel q3dpanel = new Querying3DPanel(200, 200);
JPanel jp = new JPanel();
jp.setBounds(0, 0, 200, 200);
jp.add(q3dpanel);
```
This default Querying3DPanel already comes with a default key and mouse handler (see above). The key handler is for moving the camera and to change the perspective. The following table shows a summary of defined keys and assigned actions:

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursor keys</td>
<td>move camera in x/z direction</td>
</tr>
<tr>
<td>shift + cursor keys</td>
<td>move camera in x/z direction fast</td>
</tr>
<tr>
<td>page up/down</td>
<td>rotation of the camera</td>
</tr>
<tr>
<td>shift + page up/down</td>
<td>move camera in y direction</td>
</tr>
<tr>
<td>ctrl + &quot;-&quot; key</td>
<td>zoom out</td>
</tr>
<tr>
<td>ctrl + &quot;+&quot; key</td>
<td>zoom in</td>
</tr>
<tr>
<td>ctrl + &quot;g&quot;</td>
<td>enable/disable grid</td>
</tr>
<tr>
<td>ctrl + &quot;0&quot;</td>
<td>load default view</td>
</tr>
</tbody>
</table>

The keys are defined in the class Querying3DPanelKeyHandler.

Every Query3DPanel object has one assigned instance of Querying3DPanelProperty. The properties of a 3D panel can directly be changed by adjusting resources of the property class. The following example shows how the background and grid color of a 3D panel can be manipulated:

```java
q3dpanel.getProperty().setBackColor(new Color(0.8f,0.8f,0.8f));
q3dpanel.getProperty().setGridColor(new Color(0,0,0));
```
Additionally, it is also possible to create a new `Querying3DPanelProperty` class and assign it to the 3D panel:

```java
q3dpanel.setProperty(Querying3DPanelPropertyObject);
```

The following table gives an overview of the parameters used in the 3D panel property class:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>perspective projection</strong></td>
<td></td>
</tr>
<tr>
<td>fovy</td>
<td>Specifies the field of view angle (in degrees) in the y direction</td>
</tr>
<tr>
<td>aspect</td>
<td>Specifies the aspect ratio that determines the field of view in the x direction. The aspect ratio is the ratio of x (width) to y (height).</td>
</tr>
<tr>
<td>zNear</td>
<td>Specifies the distance from the viewer to the near clipping plane (always positive).</td>
</tr>
<tr>
<td>zFar</td>
<td>Specifies the distance from the viewer to the far clipping plane (always positive).</td>
</tr>
<tr>
<td><strong>window size</strong></td>
<td></td>
</tr>
<tr>
<td>windWidth</td>
<td>Width of the window in pixel.</td>
</tr>
<tr>
<td>windHeight</td>
<td>Height of the window in pixel.</td>
</tr>
<tr>
<td><strong>camera (eye) position, angel of view and restrictions</strong></td>
<td></td>
</tr>
<tr>
<td>aov, aovDefault</td>
<td>Angel between z-axis and camera, default 20°</td>
</tr>
<tr>
<td>cameraXpos, cameraXposDefault</td>
<td>Camera position in X direction, default=0</td>
</tr>
<tr>
<td>cameraYpos, cameraYposDefault</td>
<td>Camera position in Y direction, default=1</td>
</tr>
<tr>
<td>cameraZpos, cameraZposDefault</td>
<td>Camera position in Z direction, default=2</td>
</tr>
<tr>
<td>aovMin, aovMax</td>
<td>Minimum and maximum angel of view (0°, 90°)</td>
</tr>
<tr>
<td>cameraXposMin, cameraXposMax</td>
<td>Minimum and maximum X position (-100, 100)</td>
</tr>
<tr>
<td>cameraYposMin, cameraYposMax</td>
<td>Minimum and maximum Y position (0, 20)</td>
</tr>
<tr>
<td>cameraZposMin, cameraZposMax</td>
<td>Minimum and maximum Z position (-100, 100)</td>
</tr>
<tr>
<td><strong>camera movement steps</strong></td>
<td></td>
</tr>
<tr>
<td>stepX</td>
<td>Camera movement step in X direction</td>
</tr>
<tr>
<td>stepY</td>
<td>Camera movement step in Y direction</td>
</tr>
<tr>
<td>stepZ</td>
<td>Camera movement step in Z direction</td>
</tr>
<tr>
<td>stepaov</td>
<td>Step for changing the angle of view</td>
</tr>
<tr>
<td>cameraMovementSpeedFactor</td>
<td>Multiplication factor for fast movements.</td>
</tr>
<tr>
<td><strong>scale factors and restrictions</strong></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5 - 3D panel with grey background and grid color
scaleFactor, scaleFactorDefault
scaleFactorMin, scaleFactorMax

colors
backColor
gridColor
mediaObjectColor
mediaGroupColor
selectionColor

other options
gridEnabled, gridEnabledDefault
maxElementsPerGroup
allowMouseHandling
allowKeyHandling

Adding media objects

Each media object depicted in the 3D Panel has to be a MediaRenderer. In the current version of the framework two simple implementations of MediaRenderer exist (ImageRenderer and VideoRenderer). As mentioned above, MediaRenderer objects can be collected in a MediaRendererGroup. The main purpose of grouping MediaRenderer objects is the assignment of relevance values (e.g. "positive results") during the retrieval process. There are three relevance values for groups (positive, neutral and negative). Additionally, it is possible to assign group labels and colors to groups.

The configuration of MediaRenderer and MediaRendererGroup objects that are shown within the 3D panel is called the scene. The scene is stored in two vectors:

```java
protected Vector mediaObjects=new Vector();
protected Vector mediaGroups=new Vector();
```

The scene can be structured with the methods addMediaRenderer, removeMediaRenderer, addMediaRendererGroup and removeMediaRendererGroup of class Querying3DPanel. The position of the media- and group objects is stored in assigned property classes. The access to media objects is performed through MediaContent classes (see documentation of MediaContent for more information). Each instance of a MediaRenderer has one assigned MediaContent object.

The following example shows how two grouped images, a single image and a video are added to a scene.

```java
//create MediaContent
URL url=null;
try {
    url = new URL("file://f:/image1.gif");
} catch (Exception e) {}
MediaContent mcl=null;
try {
    mcl=new MediaContent(url);
```
try {
    url = new URL("file://f:/image2.gif");
} catch (Exception e) {}

MediaContent mc2=null;
try {
    mc2=new MediaContent(url);
} catch (Exception e) {}

try {
    url = new URL("file://f:/image3.gif");
} catch (Exception e) {}
    MediaContent mc3=null;
try {
    mc3=new MediaContent(url);
} catch (Exception e) {}

try {
    url = new URL("file://f:/blindhai.avi");
} catch (Exception e) {}
    MediaContent mc4=null;
try {
    mc4=new MediaContent(url);
} catch (Exception e) {}

//create Image- and VideoRenderer
ImageRenderer mr1=
    new ImageRenderer(q3dpanel.gl,q3dpanel.glu, mc1);
mr1.setPosition(1f,1f);
ImageRenderer mr2=
    new ImageRenderer(q3dpanel.gl,q3dpanel.glu, mc2);
mr2.setPosition(2f,4f);
ImageRenderer mr3=
    new ImageRenderer(q3dpanel.gl,q3dpanel.glu, mc3);
VideoRenderer mr4=
    new VideoRenderer(q3dpanel.gl,q3dpanel.glu, mc4);
mr4.setPosition(2f,1f);

//create MediaRendererGroup and add MediaRenderer to the group
MediaRendererGroup mrg1=
    new MediaRendererGroup(q3dpanel.gl, q3dpanel.getGLF(),
                          -2f,-2f);
mrg1.setColor(new Color(0f,0f,0f));
mrg1.addMediaRenderer(mr1);
mrg1.addMediaRenderer(mr2);

//insert MediaRender and MediaRenderGroup into the scene
q3dpanel.addMediaRenderer(mr3);
q3dpanel.addMediaRenderer(mr4);
q3dpanel.addMediaRendererGroup(mrg1);
Interaction with media objects

As mentioned above, interaction with media objects is performed through mouse operations. For this purpose, the Java listener interfaces MouseListener and MouseMotionListener are implemented by the class Querying3DPanel. By the implementation of these listeners, basic mouse events can be handled. Furthermore, to be able to interact with media objects in a 3D panel, it is necessary to map the screen coordinates to the actual position of the media objects. This mapping is implemented in the class Querying3DPanel. If a mouse event is resolved as being relevant for one of the media objects or groups in the scene, a Querying3DPanelMouseEvent is fired. This event contains information about the object that fired the event. Additionally, other involved objects and groups are stored in two vectors. This is necessary if more than one object/group is involved in the event action (e.g. dragging a selection).

In Querying3DPanelMouseListener (the listener interface for the Querying3DPanelMouseEvent) we distinguish between the following four events: mouseClicked, mouseEntered, mouseExited und mouseDragged.

One Querying3DPanelMouseHandler is registered by default. This default handler is responsible for moving and selection of media objects and groups. It is also possible to register and remove other mouse handler with the methods addQuerying3DPanelMouseListener and removeQuerying3DPanelMouseListener of Querying3DPanel. This may become relevant if, for example, it is desired to show information on the media object behind the mouse cursor in a separate panel.

How to implement additional Media Renderer classes

As mentioned above, it is possible to extend the framework by additional media renderer classes. A new renderer has to extend the class MediaRenderer. MediaRenderer contains basic and media-independent methods (like moving and selecting objects). Implementation-specific functionality has to be implemented in the new class. Every new
renderer class has to fulfil two major tasks: (1) generation and management of textures of media objects and (2) drawing of textures on the screen.

**Generation of texture images**
Before a representation of a media object can be visualised, it is necessary to generate texture images that represent the object. These textures are mapped onto polygons and other primitives. The OpenGL standard requires that the dimension of texture images must be a power of two. For more information about how to create textures in OpenGL (and GL4Java) see the OpenGL documentation or have a look at the implementation of the ImageRenderer class.

**Drawing the representation of the media object**
Every new media render has to implement the drawing method defined in class MediaRenderer:

```java
public void draw(float aov, int mode, int name);
```

Parameters are explained in the API documentation.

As mentioned above, Querying3DPanel stores the scene information in two vectors (mediaObjects and mediaGroups). In method display() of Querying3DPanel, the draw methods of all active media renderer objects in the scene are called. See the implementation of the ImageRenderer class for an example.

**Context menu**
In the current version of the VizIR framework there is just a very simple context menu to enable/disable the grid on the surface of the 3D panel. In future versions we will try to provide a convenient opportunity to configure user-defined context menus.