To appear in Proc. of VRML98, Monterey, California, February 1998.

Looking Inside VRwave:
The Architecture and Interface of the VRwave VRML97 Browser

Keith Andrews Andreas Pesendorfer Michael Pichler Karl Heinz Wagenbrunn Josef Wolte *

IICM, Graz University of Technology, Austria

ABSTRACT

This paper presents an inside look into the VRwave VRML97 browser, discussing its internal architecture and some of the insights we have gained during its development.

VRwave is written largely in Java and is freely available in source code. A Java layer atop OpenGL provides 3D graphics output. In terms of look and feel, VRwave has a similar interface to the VRweb VRML 1.0 browser. VRwave also supports the Java External Authoring Interface (EAI), allowing it to be driven by an external program.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing Algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques.

Keywords: VRML, browser, VRwave, Java

1 INTRODUCTION

From 1992 to 1994 we developed a browser for interlinked 3D models for the Hyper-G Internet information system [9], called the Harmony 3D Scene Viewer [10, 2]. For lack of any other standard in 1992, we developed our own file format for 3D hypermedia models (called SDF for Scene Description Format), based on the ascii output from Wavefront's Advanced Visualizer software which we used at the time for modeling. The Harmony 3D Scene Viewer was first presented in a poster at the European Conference on Hypertext in Milan in December 1992 [1], and to our knowledge was the first work combining 3D models and hyperlinking across the Internet.

As discussions on VRML began to gain pace in late 1994, and an embryonic specification began to emerge, it seemed reasonable to adapt our browser to read VRML rather than SDF. Hence in April 1995 at the 3rd WWW conference in Darmstadt, we were able to announce a joint project with NCSA and the University of Minnesota to develop VRweb [3], a VRML browser based on the Harmony 3D Scene Viewer which we would make freely available in source code. The first version of VRweb was duly released on 5th July 1995 and an overview presented at VRML'95 [11].

In the summer of 1996 we began work on a VRML 2.0 browser. VRweb had been written in C++, but Java was gaining acceptance and promised numerous advantages over C++, not least of which was (some degree of) platform-independence. The new browser was written in Java, named VRwave to distinguish it from VRweb, and an initial version released on the 3rd February 1997. This paper takes a look inside VRwave, discussing the architecture of the code, the design of the user interface, and presenting some of the problems, discoveries, and insights we have encountered so far.

*kandrews@iicm.edu, apesen@iicm.edu, mpichler@iicm.edu, kwagen@iicm.edu, jwolte@iicm.edu
IICM, Schießstattgasse 4a, A-8010 Graz, Austria.

2 THE ARCHITECTURE OF VRWAVE

This section gives an overview over the internal class structure of VRwave and describes the main functionality and concepts of its primary classes.

VRwave source code is comprised of several modular Java packages. The core classes of VRwave, contained in package `iicm.vrml.vrwave`, handle the logic of the browser. Parsing a VRML97 input stream and building a scene graph is done by the the 'pw' library (package `iicm.vrml.pw`). Where needed, additional node data is stored in a `pwdat` subpackage. `iicm.widgets` contains a number of reusable GUI widgets.

Rendering is performed via a Java interface (package `iicm.ge3d`) to the GE3D native code (C) rendering library, which uses OpenGL as the backend graphics package. Several 3D data structures (3D vectors, matrices) and utility code for rays and picking are contained in the `iicm.utils3d` package.

Finally, VRwave implements the Java External Authoring Interface (EAI), which consists of the package `vrml.external` and the subpackages `field` and `exception`. It is planned to extend VRwave to support Java within Script nodes, implementing the `vrml`, `vrml.field` and `vrml.node` packages. Figure 1 shows the internal architecture of VRwave.

2.1 Representation of a VRML Scene

The entry point to the VRwave browser is class `iicm.vrml.vrwave.VRwave`. It can be run as a standalone Java application (providing a main method) or as a Java applet embedded in HTML (overriding the start and stop methods of class Applet). It evaluates parameters given on the command line or included in the APPLET tag to set options and specify the file name or URL of the VRML97 input data to load. Its main tasks are to create an instance of class `Scene`, which reads the initial scene data, and to create either the VRwave application window or the Java applet windows. Moreover, it includes methods needed to establish communication with the `vrml.external.Browser` class used in Java EAI applets. All further control and management of the VRML scene is handled by the Scene instance.

Class `Scene` is responsible for central control and management of user interactions and the drawing of the scene. It provides several methods to pass the VRML97 input data from different sources to the parser, and keeps the root level node of the scene graph and a naming dictionary created by the parser. User interface elements, like toolbar, status line, and menu bar, are constructed using Java AWT (Abstract Windowing Toolkit) classes or subclasses

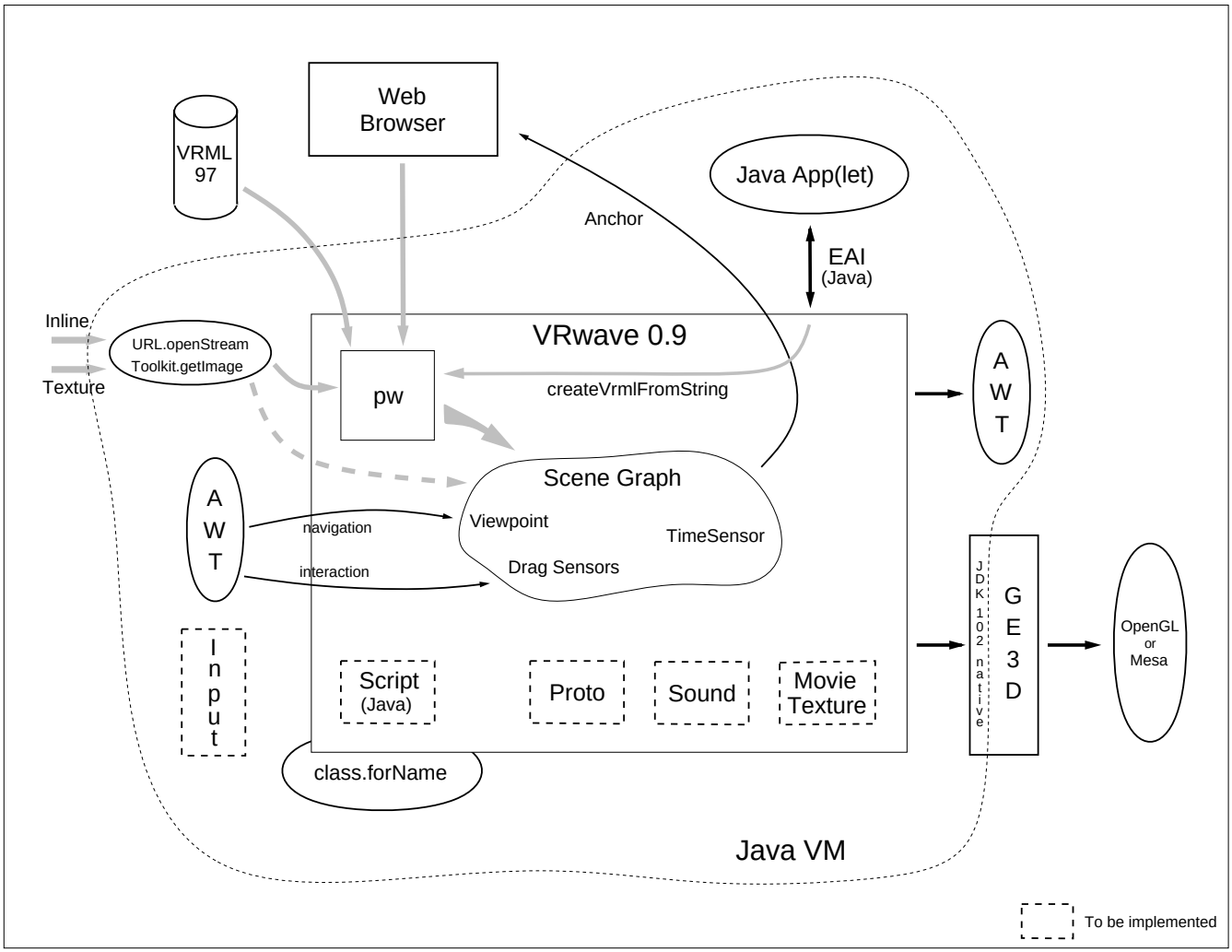

Figure 1: The internal architecture of VRwave.

of them. The most important of the derived components, class `iicm.ge3d.OGLCanvas` (described below), provides the drawing area, which uses native code to embed the OpenGL window into the Java GUI.

2.2 The 'pw' VRML97 Parser

The 'pw' (short for "parse world") VRML97 parser is implemented as package `iicm.vrml.pw`. The main class, `VRMLparser`, reads the VRML input data from a `java.io.InputStream` and creates the scene graph using nodes subclassed from class `iicm.vrml.pw.Node`. There are several node categories (grouping, geometry etc.) and there is a node class corresponding to each VRML97 node eith the appropriate fields. For uniform handling (simpler scene graph traversal), a root level `GroupNode` is created, containing the VRML file's top level nodes. A `java.util.Hashtable` is used to map names of DEFed nodes to their representation in the scene graph, and is consulted for USE statements and when accessed via the Java EAI.

Fields within nodes are represented by instances of class `iicm.vrml.pw.Field` or its subclasses respectively. The field/event categories (plain) Field, eventIn, eventOut, and exposedField, are all handled within one Field base class – in fact, an exposedField is just a combination of a Field, an eventIn, and an eventOut (this design decision was also influenced by the single inheritance model of Java).

Fields containing an eventOut (including exposedFields) keep a list of fields (eventIn, exposedField) they have a route to. The routes are created by method addRoute of class `Node`, which uses a helper function to complete field names with the appropriate prefix/suffix (e.g. 'translation' to 'translation_changed', 'rotation' to 'set_rotation'). The Java EAI also supports removing routes through the method deleteRoute. Once the fields invoked for a route are located, the routes are created/removed using a field's addReceiver and removeReceiver methods. The sending of events is described later.

Use of a scene graph is not necessarily imposed by the VRML97 design. Since a shape's attributes and geometry are tied together in VRML97, it is possible to choose non-hierarchical data structures inside the browser. The advantage of the scene graph model is that it already includes the transformation hierarchy needed for drawing, and that it provides the possibility for saving the current state of a dynamic scene. Class `Traverser` handles recursive traversal of a scene graph by visiting each node and calling an abstract method for

each node type, which has to be implemented by derived traverser classes.

For VRwave, care has been taken to keep 'pw' a reusable VRML97 parser library. A typical example is the preprocessing step necessary before drawing the scene for the first time. This is done by class `Builder`, which extends the Traverser class. The additional node information (e.g. the transformation matrix corresponding to a Transform node) is stored in VRwave specific classes (e.g. class TransformData) of subpackage `pwdat`, which are referenced via the userdata field of each Node. In dynamic scenes these data structures have to be updated after receiving events changing the field values they depend on, which again is described later.

2.3 Rendering

Rendering of the 3D scene in VRwave is done via the Java interface to the GE3D rendering library ("Graphics Engine for 3D"). GE3D contains higher-level functions for setting up a 3D camera, defining transformation hierarchies, materials, light sources, drawing polygonal data sets and quadric primitives and so on. OpenGL serves as the backend graphics library (or the Mesa library using OpenGL's API). Layers above the GE3D library do not have to care about differences between underlying graphic libraries. The current VRwave version uses the Java native interface of JDK 1.0.2. The implementation of the GE3D/OpenGL context and possible alternatives is described in more detail in the next subsection.

Every scene rendering, which causes a call of the `SceneCanvas`' paint method, is done by a scene graph traversal in class `Drawer`. Its methods specify the drawing rules for each node type based on methods of class `GE3D` and using information stored in the data structures created during building.

The `Scene` class also implements behaviour functions which occur on each frame. On each redraw, the timestamp of the current frame (needed for event handling) is obtained from the Java system time and stored. Any TimeSensors, registered during the preprocessing step, are then notified. Viewpoint management (including user navigation) is also handled by class Scene, which does not support bindable node behaviour at this time.

2.4 OpenGL and Java

OpenGL is a standard library for 3D graphics, which is easy to program, widely used, very suitable for hardware acceleration, and available on many platforms. OpenGL originally evolved from SGI's GL and is now available for Windows, various Unix machines, and the Macintosh. A graphics library called Mesa, which is freely available in source code, also uses the OpenGL API. It was natural to provide a Java binding to OpenGL, which was originally done by Leo Chan at the Computer Graphics Lab University of Waterloo [4], and later by various other people. However, there does not exist an "official" (standard) binding for OpenGL.

Providing Java wrappers for the various OpenGL calls is straightforward and easily done via the Java native code interface. The more difficult part is to provide the window context integration (which is not part of the core OpenGL library) for various platforms. There are basically two possibilities to achieve this: the native code can open a new toplevel window for rendering or integrate a drawing area into a window or canvas created via Java's AWT. The first approach is relatively simple, but lacks true integration of 3D output into a Java application, which may include a menu bar and "decorations" like buttons, status lines and so on.

For this, it is necessary to obtain the native window information from the Java Canvas (an AWT subwindow for drawing) and to establish an OpenGL context for it. One way to do this is to circumvent Java's internals and get this information in a window system dependent manner. (as we did for VRwave, described next). The more direct way is to use the interface to the Peer objects of the JDK. The latter method is more reliable and provides the possibility to use several OpenGL canvases within one Java application, but obviously depends not only on the platform, but also on the particular JDK implementation (the interface of which may also be version dependent).

As mentioned above, class `iicm.ge3d.OGLCanvas`, a subclass of the Java AWT Canvas, uses native methods to get an OpenGL context for rendering into the Canvas area. For this purpose it is necessary to determine the Canvas' native window ID. A string containing the title of the Java frame window is passed to the native code implementation (in file `gejcon.c`). The application window ID is found by examining the window titles of all children of the root window. Afterwards, the canvas window is determined by choosing the most deeply nested subwindow (the search is recursive). It also provides a method to activate and initialise the created OpenGL context, which must be called prior to issuing OpenGL commands.

In August 1997, the Java3D API specification [6] was released as part of the Java Media API framework. Java3D uses a high-level, scene graph-based programming model. Initial Java3D implementations are expected to be layered atop other, lower-level 3D graphics libraries, such as OpenGL, Direct3D, or QuickDraw3D. As Java3D implementations become available in 1998, the rendering part of VRwave can be replaced to make Java3D calls, allowing a 100% pure Java version of VRwave.

2.5 Event Processing

VRML events originate from Sensors, Scripts, or the Java EAI. Sensors may become active based on time (TimeSensor), user input (TouchSensor, dragging sensors) or scene states (VisibilitySensor, Collision). When an eventOut (managed by class `Field`, see 'pw') has to be sent, it calls its sendEvent method passing the current timestamp and itself as sender to its receiveEvent method, i.e. it reacts as if it just got the event and has to dispatch it further on to its own recipients. Loops are prevented by comparing the timestamp with the last time of receiving an event.

Each class wanting to be notified about a change of field value has to implement the GotEventCallback interface and register itself for the field with the method setEventCallback. At the time of receiving an event, all these callbacks are executed. This mechanism is provided for the Java EAI and is also used inside VRwave, for example to recompute a transformation matrix when a field is changed. Finally, receiveEvent is called for each receiver.

Class `SceneCanvas` extends the `OGLCanvas` of the GE3D package. It overloads the paint and event handling methods to draw the 3D scene and to handle user input (mouse and keyboard events). The paint method activates the OpenGL context and calls the draw method of the scene class, which draws the 3D scene via the Drawer traversal. Afterwards, any icons forming part of the navigation interface (for example heads-up navigation mode) are drawn atop the 3D scene. Finally, it must schedule another redraw if there are any TimeSensors active (and behaviour is enabled).

Mouse and keyboard events in VRwave are separated into two categories: *navigation* and *interaction*. Navigation events control the user's movement through the world (provided by the browser), whereas interaction events encompass activation and manipulation of Anchors and dragging sensors (e.g. CylinderSensor) defined in the VRML scene.

2.6 Communication

Inline scene and texture image requests are handled by an instance of class `iicm.vrml.vrwave.URLServer`. It implements the `java.lang.Runnable` interface. Therefore, its in-

stances can be executed by threads and load the scenes and images without blocking. It manages a queue of nodes which request the data. Depending on the type of the next element in the queue (Inline or ImageTexture) the appropriate method is called to load the data. Inline scene data is directed into the parser as a `java.io.InputStream` as received from `java.net.URL.openStream()`. The scene graph created by the parser is traversed by the `Builder` class and its root level node is stored as `InlineData`. Texture image data is loaded with `java.awt.Toolkit.getImage()`. The loading process is controlled by a helper class, which implements the `java.awt.image.ImageConsumer` interface and receives pixel data in a Java float array. This array is passed to a GE3D method, which encapsulates the texture inside an OpenGL/Mesa display list (or texture object) and returns a texture identifier number for drawing.

Another job of the `Scene` class is to send the URL information of any activated anchors to an external web browser and cause it to display the requested document. If VRwave is running as a standalone application this is done by a Netscape remote call, otherwise it can be done directly via the AppletContext.

3 THE VRWAVE USER INTERFACE

In terms of the look and feel of its user interface, VRwave is a direct descendant of VRweb. The VRwave window is divided vertically into four areas: menu bar, tool bar, display area, and status bar. The menu bar provides access to the full functionality of VRwave, the tool bar and additional accelerator keys provide quick access to commonly used functions.

As in VRweb, five rendering modes are supported: wire frame, hidden line, flat shading, smooth shading, and texturing. The rendering mode specified in the VRML file can be overridden. It is also possible to specify a separate rendering mode for use during interactive navigation. For instance, on a display without hardware graphics acceleration, it might be advisable to navigate in wire frame mode and see a textured version only when stationary.

VRwave provides four navigational modes. *Flip* mode is used to examine an object, whilst the viewer remains stationary. The mouse buttons have assignments for translation, rotation, and zooming. Figure 2 shows a model of a cavalry pistol from the famous collection in the Landeszeughaus (Armoury) in Graz being examined in Flip mode.

The *Walk* metaphor is used to stroll through a 3D environment. Natural walking motion (forwards and backwards, possibly veering slightly to left or right) is assigned to the left mouse button. Complementary controls for vertical motion and/or side-stepping, and for turning the head are assigned to the middle and right mouse buttons respectively.

Fly To mode implements point-of-interest (POI) style navigation [7]. Here, the user first selects a point of interest somewhere in the model and is then able to perform controlled, logarithmic motion towards (and away from) the POI, approaching by the same fractional distance in each time step. Optionally, a rotational component can be activated (with the Shift key), which results in a final approach path to the POI head-on along the surface normal. This mode is very useful for examining details of a scene and complements other navigation metaphors like Walk or Flip, but is not sufficient as a navigation technique per se.

Heads Up is perhaps the easiest navigation mode for beginners, since its controls are clearly visible. Icons overlaid across the centre of the viewing window (like a pilot's heads-up display) symbolise the individual navigation tools: eyes to look around, a walking person for walking, crossed arrows for vertical and sideways motion. Figure 3 shows the Stefaniensaal of Graz Convention Center [5] being explored in Heads Up mode.

Other functions include opening local files, saving files, setting preferences, changing colours, and so forth. The current frame rate can be displayed in the status area.

4 THE JAVA EXTERNAL AUTHORING INTERFACE (EAI)

The Java External Authoring Interface (EAI) [8] allows an external program (applet or application) to drive the VRML browser. The principal functionality provided by the Java EAI is to make a Browser object available to an external Java applet, which gives access to the entire Browser Script Interface defined for scripts inside Script nodes, and to nodes named by DEF in the scene.

Currently, there are two commonly used ways to get an instance of this Browser class: getting it from the static method `Browser.getBrowser()` or using Netscape's LiveConnect which depends on a proprietary netscape package and requires the Browser class to be a subclass of Plugin, which is also proprietary to the Netscape browser. The Browser class of VRwave's Java EAI implementation supports both options.

As most of VRwave's functionality can be accessed via the Scene class, the Browser instance obtained contains a private reference to VRwave's actual Scene instance. Before any of its methods can be called, it must be ensured that this Scene instance has been created by VRwave already. This is done implicitly if the Browser instance has been obtained using its getBrowser() method: the method returns null as long as the Scene instance doesn't exist, a loop in the external applet is run until it returns the Browser instance. If the Browser instance was obtained using LiveConnect, a method which blocks until the Scene instance has been created is called at the beginning of each method accessing this Scene instance.

After parsing the root scene, the Scene instance obtains the Hashtable mapping DEF names to Nodes inside the scene graph from the parser and stores it as a member variable. The `Browser.getNode()` method uses this Hashtable to get a reference to the required node. This reference is passed to the created instance of vrml.external.Node, which stores a reference to Scene in a private member variable for event handling purposes. This Node instance is returned by method getNode().

Instances of `vrml.external.Node` provide methods to access the node's EventIn and EventOut fields by name. They use the iicm.vrml.pw.Node classes' `getEvent()` method, which returns a reference to the required field represented by an instance of `iicm.vrml.pw.Field`. Depending on the field type a new instance of the corresponding class of package `vrml.external.field` is created and returned. They are subclasses of `vrml.external.field.EventIn` or `vrml.external.field.EventOut` and contain methods to send events to the field or get the current value of the field respectively. They also maintain a reference to VRwave's Scene class instance.

Class `vrml.external.field.EventIn` defines a method sendEvent, which calls the sendEvent() method of its associated EventIn field (which is a subclass of `iicm.vrml.pw.Field`) after ensuring that the event does not have the same timestamp as the last event sent to this field. Events with same timestamp to one field are ignored to avoid event loops.

Instances of subclasses of `vrml.external.field.EventOut` should be able to register classes implementing the `vrml.external.field.EventOutObserver` interface, whose callback() methods are then called whenever an event is generated for the associated EventOut field. The advise() method takes the class, which is to be registered, and a user-defined

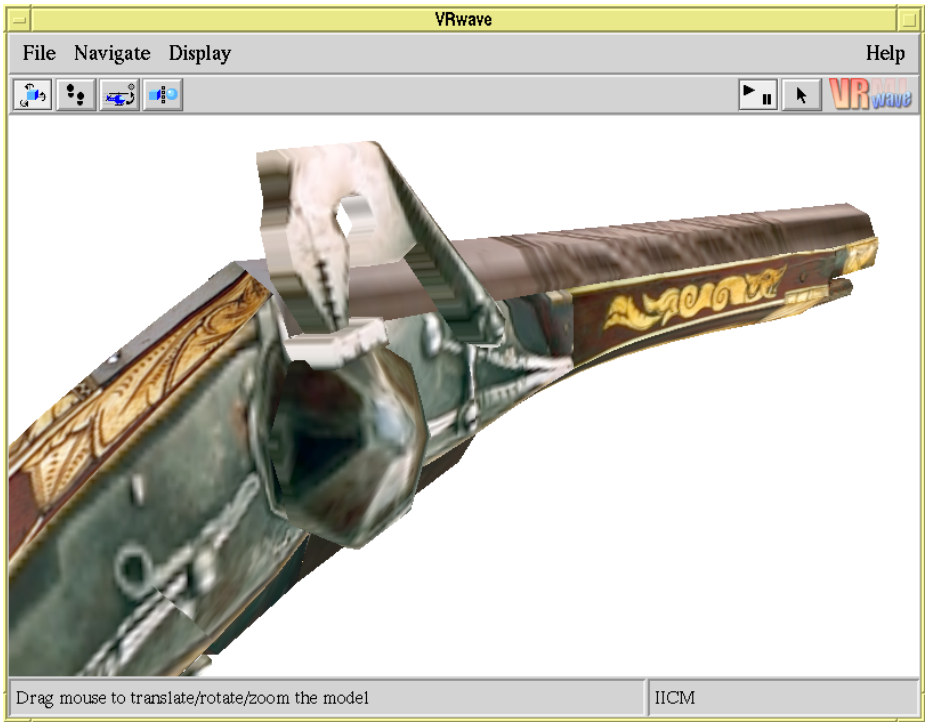

Figure 2: VRwave in Flip mode displaying a textured model of a cavalry pistol from the world-renowned Zeughaus (armoury) in Graz.

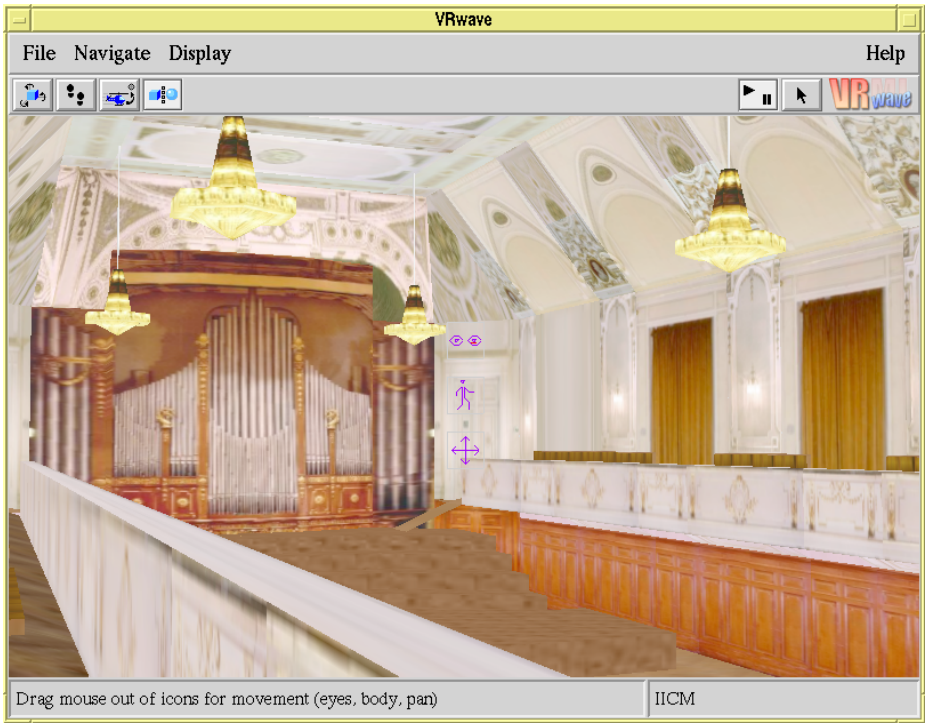

Figure 3: VRwave in Heads-Up navigation mode showing the Stefaniensaal of Graz Convention Center. Note the navigational icons overlaid across the centre of the viewing area.

object as parameters and adds them to lists. The EventOut class itself implements the iicm.vrml.pw.GotEventCallback interface, described above. On the first call of advise(), it registers itself to its EventOut field's list of callbacks. It's gotEventCB() method is now called whenever an event is sent by the EventOut field. Inside this method, it calls the callback() methods of all EventOutObserver classes, passing along the user defined object of the advise() call, which makes it possible to use the same callback function to handle events from multiple sources.

The `Browser.createVrmlFromString()` method creates a new instance of iicm.vrml.pw.VRMLparser to parse the String parameter containing the VRML data. Since the 'pw' parser processes input of class java.io.InputStream, an instance of java.io.StringBufferInputStream has to be created from the input String. The scene graph is built and stored in a Node array, which is duly returned by createVrmlFromString().

The `Browser.addRoute()` and `Browser.deleteRoute()` methods use the addRoute() or deleteRoute() methods of the nodes (instances of class iicm.vrml.pw.Node) containing the affected fields. The remaining methods of Browser use functions provided directly by the Scene instance, for example getName(), getCurrentFrameRate(), and getWorldURL().

5 CURRENT AND FUTURE WORK

Work on VRwave is ongoing. PROTOs and EXTERNPROTOs are currently parsed but are then ignored. Work on supporting Java scripting in Script nodes is underway, ElevationGrid and Text node support is planned soon. Currently, VRwave is available for Unix platforms, but work on a Windows port is underway.

In order to accelerate picking and collision detection, bounding boxes will be maintained internally. Viewpoint management and the two viewpoint-dependent sensors (VisibilitySensor and ProximitySensor) also still need to be implemented. Farther in the future, support for AudioClip and MovieTexture will depend on the availability of underlying audio and video libraries.

Particular priority is being given to the encapsulation of 3D output for various platforms. VRwave 0.9 uses the JDK 1.0.2 native code interface to access the GE3D and thence OpenGL libraries for 3D output. Unfortunately, there is no longer a single, standard native code interface for Java, but three different ones: JNI (Sun JDK 1.1), JRI (Netscape), and RNI (Microsoft). This necessitates branching code (`#ifdef` and `#define`) dependent on the local Java Virtual Machine being used (appletviewer, Netscape, or Internet Explorer respectively).

As long as native code is used for 3D output, users have to install a local library (GE3D for OpenGL access) in addition to the VRwave Java code. This is somewhat troublesome and prohibits the use of an entirely applet version of VRwave. Hopefully, at some point in the future implementations of Java3D will be available across multiple platforms, and GE3D can be eliminated entirely, resulting in a 100% pure Java version of VRwave.

6 CONCLUDING REMARKS

We have presented an inside look at the VRwave browser for VRML97, discussing its internal architecture and some of the insights we have gained.

VRwave is available both in binary and in source code. The source code is copyrighted, but is freely available for non-commercial use (see the licence notice with the distribution for full details). Further information about VRwave is available from the VRwave Home Page at:

`http://www.iicm.edu/vrwave`

The VRwave distribution itself is available by anonymous ftp from:

`ftp://ftp.iicm.edu/VRwave`

and numerous mirror sites worldwide. At time of writing, the latest release was VRwave 0.9 of 30th September 1997.

The VRML97 parser 'pw' is also available separately under the GNU LGPL (Library General Public License), allowing it to be freely incorporated into other programs.

7 ACKNOWLEDGEMENTS

We would like to thank our colleagues at the IICM and the VRweb and VRwave user communities for their support and helpful suggestions during over the past 5 years. The name VRwave is a trademark of IICM.

REFERENCES

[1] Keith Andrews. Using 3D scenes as hypermedia nodes. Poster at the ACM European Conference on Hypertext (ECHT'92), Milan, Italy, December 1992.

[2] Keith Andrews. Constructing cyberspace: Virtual reality and hypermedia. Presented at Virtual Reality Vienna '93. `ftp://ftp.iicm.edu/pub/papers/vrv93.ps.gz`, December 1993.

[3] Keith Andrews. VRweb project announcement (press release), April 1995. `http://www.iicm.edu/vrweb-press-announce01.html`.

[4] Leo Chan. An unofficial port of opengl to java. Computer Graphics Lab, University of Waterloo. `http://www.meta.cgl.uwaterloo.ca/SourceCodeAndDemos/OpenGL4java.html`.

[5] Cyber Congress. Graz Convention Center. `http://www.gcongress.com/vrml/3dmodel.htm`.

[6] Java 3D API Specification. `http://java.sun.com/products/java-media/3D/`.

[7] Jock D. Mackinlay, Stuart K. Card, and George G. Robertson. Rapid controlled movement through a virtual 3D workspace. In *Proc. SIGGRAPH'90*, pages 171–176, Dallas, Texas, August 1990. ACM.

[8] Chris Marrin. External authoring interface reference, January 1997. `http://vrml.sgi.com/moving-worlds/spec/ExternalInterface.html`.

[9] Hermann Maurer, editor. *HyperWave: The Next Generation Web Solution*. Addison-Wesley, May 1996. `http://www.iicm.edu/hgbook`.

[10] Michael Pichler. Interactive browsing of 3D scenes in hypermedia: The Hyper-G 3D viewer. Master's thesis, Graz University of Technology, Austria, October 1993. `ftp://ftp.iicm.edu/pub/papers/pichler1.ps.gz`.

[11] Michael Pichler, Gerbert Orasche, Keith Andrews, Ed Grossman, and Mark McCahill. VRweb: A multi-system VRML viewer. In *Proc. First Annual Symposium on the Virtual Reality Modeling Language (VRML '95)*, pages 77–85, San Diego, California, December 1995. `http://www.iicm.edu/vrml95/vrweb.html`.