University of Alberta

A Reconfigurable VR Tool for Spatial Navigation

by

Daniel Torres

Technical Report TR03-11
April, 2003

DEPARTMENT OF COMPUTING SCIENCE
University of Alberta
Edmonton, Alberta, Canada

A Reconfigurable VR Tool for Spatial Navigation

Daniel Torres Guizar
Department of Computing Science
University of Alberta
dtorres@cs.ualberta.ca

April, 2003

Abstract

Biological systems for spatial navigation provide fine-tuned mechanisms for orientation and complex navigational behavior. Experimentation in human beings is aimed to discover perceptual cues utilized to perform navigation in Virtual Reality environments. Such environments are required to have special characteristics to allow the measurement of brain activity and behavior in real time, and to provide flexibility for creating scenarios with perceptual stimuli of high quality. This paper describes a reconfigurable VR tool and a markup language created for this purpose. The capabilities and features of the architecture are detailed and the possibility of extending the system to other experimental domains is discussed.

Key words: Virtual Reality environments, Spatial Navigation, Human Response, Electroencephalographic analysis.

1 Introduction

The problem of navigation is not trivial. Many researchers in the area of Robotics and Computer Vision have devoted their time to the creation of better, faster and more reliable navigation systems for robots. A short glance at the literature will convince the reader of the complexity of this problem. Yet we, as humans, have no problem finding our way through the 3D world we live in. For us, navigation is almost a subconscious task. The brain manages to orchestrate our movements in a completely transparent way.

Many questions arise from this inherently natural ability. How do we find our way home when going back from work? How do we learn a new route when going to some place for the first time? How do we recognize a particular place in the city among other similar ones? How do we manage

to find our office in the apparently endless corridors of a business facility? Still, we know our perception can be tricked and even we get lost sometimes. Theseus himself would have been hopeless inside the Minotaur labyrinth had not Ariadne presented him with a thread to mark the way out. More concise questions can be formulated: What makes it easy for humans to navigate a 3D environment? What makes it difficult? What are the perceptual *cues* used by the human brain to perform this task?

A *maze* is an excellent testbed to address these questions. A virtual maze can be designed to explicitly test certain navigation abilities of the human user inside a controlled environment. Features like wall color and texture, topological structure and pattern, noise and navigation aids can be strategically placed and varied to analyze the behavioral response. At the same time, different recordings like the user's navigation pattern, time to find the exit and committed errors, (such as reaching a dead-end corridor on a previously walked section) can provide information about the processes that take place in our natural navigation system.

For this purpose a reconfigurable Virtual Reality tool named MANDALA, and its markup authoring language, was created. This paper describes the architecture, features and overall characteristics of this system. MANDALA was also designed with a second goal in mind: in many research projects related to psychology, biology, sociology and other areas not directly related to computing science, there is the need for conducting experiments in VR environments. This often presents a problem since the researcher has two options; either to adapt an existing VR system to their specific needs or build an *ad hoc* solution. In both cases significant effort must be spent performing tasks not related to the research. MANDALA comes here as a general-purpose VR tool with a simple authoring language that requires no additional knowledge of 3D programming, allowing the user to create very complex environments that integrate seamlessly with other modules specific to the research itself.

The paper is divided as follows: A general overview of the spatial navigation experiment is described in order to illustrate the requirements of the system. The architecture of MANDALA is then shown and a more detailed description of its elements and features will be provided. Some illustrative examples are shown along with their code. The general dynamics of a typical navigation experiment from the point of view of the tool is also described. Finally the current state of the MANDALA project and future work are presented.

2 The Spatial Navigation Experiment

A very quick overview of the Spatial Navigation experiment is described here. As this is not the place to discuss the mentioned project, only information relevant to the design of the virtual domain will be discussed.

The intention of the Spatial Navigation experiment is to discover how people find their way through a virtual environment, what are the perceptual cues they rely on, and what makes it easy or difficult to find the goal. To answer this questions a group of volunteers are asked to navigate a Virtual Reality maze that presents carefully designed *stimuli*, while simple behavioral measures like the time they take to complete each maze and the number of errors are recorded. Electroencephalographic activity is measured as the subject walks through the maze . Theta wave activation [4] indicate places where navigation has become difficult.

The whole simulation is automatic and different mazes are created, presented and evaluated in real time. At the end of the experiment a history of the simulation must remain for analysis. Given this scenario, the maze domain must be able to do the following:

1. Allow the construction of virtual environments in a simple way, so that there is not need of working with complicated data for the setup of the maze. At the same time the system must let the designer construct libraries of maze segments so that the maze can be built by joining together previously defined sections.
2. Present the user with a simple, clean and realistic interface that behaves in a believable way, using a variety of input devices like the keyboard or joystick.
3. Offer a flexible authoring and scripting language so that mazes can be made interactive and different events may be triggered in the appropriate situations.
4. Provide means for automating the use of external hardware devices like the electroencephalographic reader when certain events inside the virtual world are triggered.
5. Keep a log of interesting events and track the navigation of the user for offline analysis
6. Allow real time communication with external agents so that other expert systems can react to the events of the maze, review the navigation of the user and evaluate his performance, design new mazes on-the-go and present them to the user without suspending execution of the simulation.
7. Work with conventional computer monitors as well as with specialized VR hardware like stereo displays and the CAVE system.

With this in mind the available options are greatly reduced. Languages like VRML did not provide all the required flexibility with the expected performance, and adapting some existing graphic engine would have implied additional time to understand and restructure (when possible) other architectures that are frequently unfinished and unsupported. Buying a supported product would have implied additional economic resources and would still have to be adapted to the requirements of the experiment. Given that this was not the only project in need of a similar system, and that the requirements were very specific, the idea of developing a proprietary engine was decided.

3 MANDALA: A Virtual Reality tool

The name of the engine will be explained a bit later in this paper. First it is necessary to present the two main objectives of the MANDALA project:

1. To create a flexible markup language that will:
   - Allow the definition of virtual worlds while encapsulating all complex 3D details so that people without prior experience in computing science or Virtual Reality would be able to design and put their own scenarios to work
   - Allow the definition and inclusion of libraries for simplifying the process of constructing the virtual environment.
   - Allow straightforward scripting for defining interactivity.
   - Allow communication routines so that attached devices can be controlled from within the maze without the need of implementing other modules.
   - Provide all required resources to build simple worlds, without restraining the possibility of building more complex scenes.
   - Although inspired by the requirements of the Spatial Navigation experiment, be completely domain-independent so that it can be seamlessly imported into different domains.
2. To provide an architecture (and its implementation) that:
   - Is compatible with the MANDALA Markup Language (MML) definition and implements all of its features
   - Provides all the advantages and essential characteristics of a modern graphic engine
   - Immediately works with conventional and advanced VR hardware like stereo displays
   - Allows concurrent operation and communication with other specialized agents

3.1 Overview of the MANDALA Architecture

The MANDALA general architecture is composed of five layers and two external managers. At the very bottom we have the *World Definition*, which contains one or more simple text files written in MML specifying the objects, structure and interactivity of the virtual world. These files may also contain links to various multimedia resources like sounds, textures and geometric models in different 3D formats[1]. Next we have an interface that reads, parses and validates the world definition file(s) and integrates them into the *MANDALA Objects*, a series of specialized data structures that actually load external multimedia elements, organize the virtual world resources and keep and optimize the information for quick access from the *Real time agents*. In this layer a collection of agents that handle the realtime logistics of the world is found. Aspects like collision detection, script execution, avatar movement, navigation logging and other dynamic tasks are performed. This layer also administrates information circulating to and from two important external managers, one dedicated to administrate *input and output devices*, and another that keeps several kinds of communication channels open with *remote agents*.

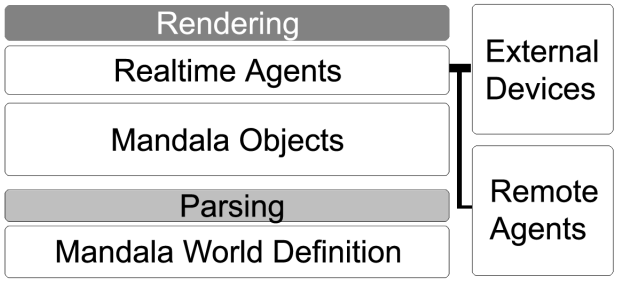

Figure 1: The components of the MANDALA Architecture

The *External Devices* manager reads input from the physical navigation controls utilized by the user to walk through the maze, this abstraction makes it possible to adapt the application to work with simple or advanced input devices without having to change other systems. Output features managed in this section include communication with hardware devices controlled by the Realtime Agents. The *Remote Agents* manager abstracts cooperation and communication with other research-specific agents that not necessarily reside on the same computer. Numerous external agents can be interacting with the MANDALA environment at the same time, sending and receiving messages to supervise the experiment in many forms. This abstraction provides the

[1]The format of external models is not restringed by MML, but the interpreters in the third layer

architecture with great flexibility for implementing domain-specific modules without having to modify the MANDALA architecture. Finally, the *Rendering* layer maintains and updates a graphical representation of the state of the virtual world. It is in this layer where some particular graphic library must be used to show the information contained in the MANDALA Objects as it is affected by the Realtime Agents, the External Devices and the Remote Agents.

Each layer is now reviewed in more detail.

3.2 The MANDALA Markup Language

It is in this section where the name of the system will make sense. First let us analyze the structure of a MANDALA file. As mentioned before, all information pertinent to the creation of a virtual world resides in simple text files, just like in html, and all multimedia elements like textures, sprite images, sounds and mesh files are included as external elements. The structure of every MANDALA file is shown in Figure 2.

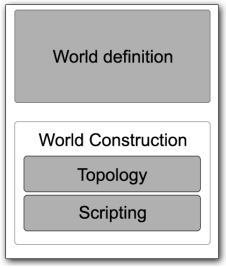

Figure 2: Structure of a MANDALA file

The two main parts of any MANDALA file are:

1. *The Definition Section*: Here all the building blocks required to assemble our world will be defined. All materials, meshes, multimedia elements, included libraries, basic and predefined structures are declared. Picture it as the box where all the pieces of a puzzle are waiting for you to take and put into place.
2. *The Construction Section*: In this section we take the building block defined in the Definition Section and actually do something with them. This is the place where the puzzle pieces are assembled to create the world itself. It contains two important elements:
    - *The world topology*: Think of it as a map that describes how pieces are to be arranged in order to construct the virtual world.

- *Scripts*: Very simple pieces of code that indicate actions to take when certain events happen somewhere in our virtual world.

Let us take a deeper look at the philosophy behind each of these sections.

3.2.1 Definition Section

The basic construction element for a MANDALA virtual world is a plain *unitary cube*. Imagine an invisible cube in space, an abstract box that occupies an area and waits for other things to be placed inside. One can place anything in these boxes and, as they are abstract elements, only what is put inside will actually *exist*. In the MANDALA language this is known as a *cell*.

Once a cell is defined many instances of it can be put together to form a bigger space. As the cell itself was only defined once, the elements it contains are also declared once. In other words, the cell as an *object* is defined and *instances* of it are connected on the construction section to assemble the virtual world.

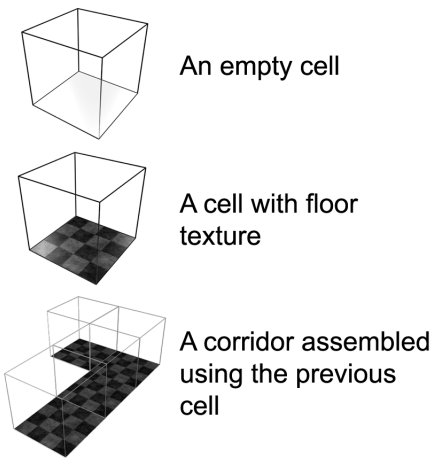

Figure 3: Using one cell to build a simple corridor

Many things can be put inside a cell. Two basic elements are a floor and a ceiling. Additionally one can put walls, furniture and objects designed in some 3D modeling program, sprites and billboards (2D images that always face the camera, giving the illusion of being 3D). In order to do that it is necessary to define how can textures be included in the file. The object used for this purpose is called a *panel*. A panel is a link to an external image

that will be used as ”wallpaper” for any surface in the world, be it a wall, a ceiling or a floor. Like the cells, once a panel is defined its instances can be used anywhere. In Figure 3 a simple cell is created by specifying a panel to use as the floor. Then, four instances of the same cell are concatenated to create a corridor. The definition of the panel and the cell would look like this in MML:

```
<!-- Defining the panel -->
<panel id='tiles' tex='someImage.bmp'/>

<!-- Defining the cell -->
<cell id='simpleCell' floor='tiles'> </cell>
```

The definition of a panel requires at least a unique id and the name of the external file. Additionally one can specify uv coordinates and RGBA values. As for the cell, an id is also required, and the floor and ceiling correspond to the id's of the desired panels. Note that the cell is actually *empty* as the floor and the ceiling are definition parameters, but not contained objects. *Walls*, in the other hand, are to be contained because they can be put anywhere inside the cell. Let us analyze a cell with a single wall as shown in Figure 4.

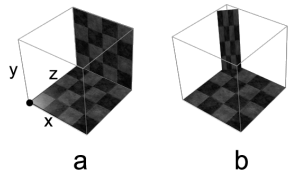

Figure 4: Two different placements of a wall. Axis and origin shown on left cell

Walls are defined by two 3D coordinates, the lower-left ($p1$) and the upper-right ($p2$) corners. The wall of left cell in Figure 4 would have the pair $p1(0, 0, 1)$ and $p2(1, 1, 1)$ (remember that we are working with unitary cells) while the wall on the right cell would be approximately $p1(0, 0, 0.75), p2(0.25, 1, 1)$. Following this method vertical walls can be positioned *anywhere* in the cell. The MML definition of the left cell will look like this:

```
<!-- cell with one wall -->
<cell id='oneWall' floor='tiles'>
   <wall panel='tiles' p1='0,0,1' p2='1,1,1'/>
</cell>
```

The wall entry needs a panel to decorate it and both 3D coordinates. A cell can hold as many walls as necessary. There are more optional parameters like visibility set to default values. During the simulation walls cause automatic collision response, so if a mesh object is put in the cell, collision

can be simplified by putting invisible walls around it.

3.2.2 Construction Section

Cells declared in the Definition section will be used to actually *construct* a maze. We have already seen in Figure 3 how a single cell can allow the creation of a whole corridor. It is now necessary to explain how cells are put together. Again, a simple method was considered. In order to create a maze we use the metaphor of the *watchman*. Imagine a watchman standing at the center of the first cell and deciding where to put the next one, he clearly has four options: *north, south, east* and *west*. Let us say that he places the next cell *west of the current cell* and he now walks to it. He is left with three options since the previous cell remains east. The next cell is placed and he moves again. The watchman can place cells in all available directions at each point and walk to the newly placed cells to put more until the world is finished. To illustrate this concept look at figure 5.

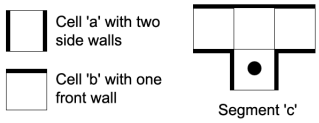

Figure 5: Two cells building a simple 'T' segment for the maze

Assuming we defined cells 'a' and 'b', the MML code to construct the 'T' maze segment is straightforward. Beginning in the dot-marked cell:

```
<!-- a simple T-like maze segment -->
<root cell='a'>
   <north cell='b'>
      <east cell='a'></east>
      <west cell='a'></west>
   </north>
</root>
```

It is important to note that the 'a' cells placed east and west are *automatically rotated* so that walls fall in correct place. Notice also that the nesting capabilities of the markup language allows simplicity when designing the maze. This metaphor also relates to known online text games, where at a given point one might choose to look around and objects standing at the four cardinal points are described. Now imagine that it is desired to reference our 'T' object as if it was a *single entity*, in fact, in MML this is called a *sector* and can be declared with a unique id at the definition section. It will look like this:

```
<!-- assuming cells 'a' and 'b' exist -->
<!-- this is our simple 'T' sector    -->
<sector id='simpleT'>
   <draw cell='a' direction='root'>
      <draw cell='b' direction='north'>
         <draw cell='a' direction='east'>
            <endpoint tag='east'/>
         </draw>
         <draw cell='a' direction='west'>
            <endpoint tag='west'/>
         </draw>
      </draw>
   </draw>
</sector>
```

Notice the endpoint tag. It tells MANDALA where to insert new cells or sectors when indicated to place them with respect to our 'T'. Using just simple T sectors it is possible to define bigger sectors and reach very high levels of complexity while maintaining simplicity of design. The complete MML file to generate the ***mandala-like*** [2] maze shown in Figure 7 is listed in Figure 6. Sectors provide a way for defining complete areas in the virtual world, one can create a sector containing a house and then put several houses to form a street with ease. It is possible to concatenate any number and combination of sectors and cells to create an adequate virtual world.In the navigation experiment, special building sets are contained in files to be included in the maze definition file. Including them and putting them together is completely trivial.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE maze SYSTEM "mandala.dtd">
<mandala>

   <!-- the definition section -->
   <definition>
      <!--- include some files with all the cell -->
      <!--- declarations                        -->
      <include file='myCells.xml'/>
   </definition>

   <!-- the construction section -->
   <construction>
      <root cell='a'>
         <!-- instead of direction tag, we use putsector -->
       <putsector sector='lotsOfT' direction='north'></putsector>
       <putsector sector='lotsOfT' direction='south'></putsector>
      </root>
   </construction>
</mandala>
```

Figure 6: The code to produce maze in figure 7

[2] The reason why *MANDALA* was chosen as the name for the system must be clearer now.

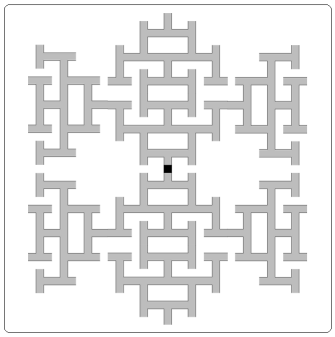

Figure 7: A more complex maze made by repeating patterns

3.2.3 Scripting

Loading a file like the one shown in Figure 6 will immediately put us inside the 3D maze (wonderfully decorated by our chosen textures) and let us navigate through it, but since this is a domain for investigation and the maze must be interactive, a scripting system was implemented. The chosen approach was, once again, very simple. It is easily shown with a couple of examples. Suppose we wanted to set some flag to TRUE if the user goes through a certain cell. Later we want to play a sound if the user successfully set the mentioned flag by the time he reached the exit cell. As every cell or sector put in the maze is an instance of the original, it is necessary to identify some special places where something is to happen. This is done by assigning a *label* when placing them in the maze:

```
...
<!-- assume this is the cell where some -->
<!-- flag goes on                       -->
<north cell='crime_scene' as='checkpoint'>
...
</north>
...
<!-- assume this is our exit            -->
<north cell='misterius_door' as='exit'>
...
```

Normally the *as* tag would not be needed. Now it is put to exactly reference these two places. We write two instructions after the topology creation of the maze:

```
<!-- set some flag to 1 when we enter the cell -->
<action id='action1' at='checkpoint' event='avatar_enter'>
```

```
   <execute function='setVariable' params='flag,1'/>
</action>

<!-- check the flag and play some sound -->
<action id='action2' at='exit' event='avatar_enter'>
   <condition>
      <require param='flag' value='1'/>
   </condition>
   <execute function='soundStart' params='finale'/>
</action>
```

The first action executes at the cell with label *checkpoint* when the avatar (that is, the user) leaves that place, creating a variable called *flag* with value equal to 1. The second action triggers when the user enters the 'exit' cell. If there exists the flag value and its value equals TRUE, then a sound with id='finale' will start playing (sounds are created in a similar way than panels, by specifying a file and some unique id in the definition section). The *condition* tag tests for several premises required for executing the list of actions. More complicated structures involving AND's and OR's can be also declared. Finally, *avatar_enter* and *avatar_leave* are events that tell when the actions are to be triggered. Table 1 shows some of the available events and functions of the MANDALA system.

Event	Description
m_start	the system is initialized
m_end	the system terminates
avatar_enter	the avatar enters a place
avatar_leave	the avatar leaves a a place
avatar_walk	the avatar moves inside a place
mouse_leftClick	left button of mouse pressed
mouse_rightClick	right button of mouse pressed
joystick_button(n)	nth joystick button pressed

Function	Parameter	Description
setStartPoint	cellId	sets start point
setEndPoint	none	sets end point
setVariable	name,value	creates a variable
incVariable	name,delta	modifies a variable
soundStart	id	starts playing a sound
soundStop	id	stops playing a sound
setFog	n,f,r,g,b	sets fog effects
setVelocity	cells/sec	changes avatar's velocity
exit	none	terminates the maze

Table 1: Some events and actions in MANDALA.

3.2.4 Creating more complex worlds

Until now It has been shown how maze-like worlds are created, but complex and thematically rich worlds can also be generated. Consider the space station pictured in Figure 8. The geometry was modeled using a commercial modeler and then included and placed as easily as a texture. The code looks like this:

```
...
<!-- include the model -->
<xmesh id='spaceStation' file='3dmodel.dat'/>
...
<!-- we just place it in some cell -->
<cell id='model'>
   <mesh ref='spaceStation' pos='0.5,0.0,0.5'/>
</cell>
```

A mesh placed inside the cell needs only one 3D position according to the cell's unitary size. Rotation and scale can also be specified. To create and navigate this scenario we define the cell that contains the model and an 'a' type cell like the one in Figure 5 with both walls are invisible. Now we have a pathway to walk through the bridge of the space station. Note that if we assign a mesh to the cell definition, an instance of such mesh will be put for each instance of the cell. This leaves the creation of virtual worlds limited only to the imagination of the designer.

Figure 8: A more interesting MANDALA virtual world

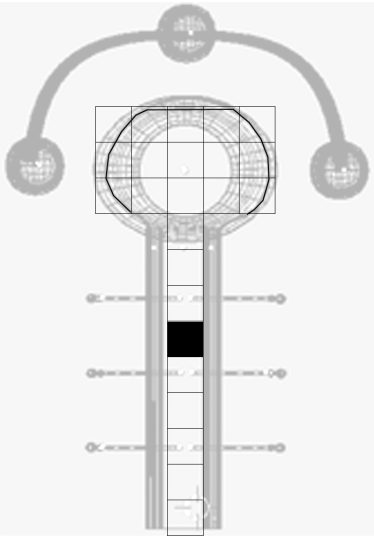

Figure 9: Top Layout of cells over the 3D model shown in Figure 8 to allow navigation over desired areas. The black cell is the one containing the model (which size is in fact not limited by the cell). The end cells contain a structure of invisible walls to prevent the user from walking away.

3.3 The MANDALA objects

When the MANDALA application starts, some virtual world file is read and parsed[3] into the *MANDALA Objects*, a group of data structures that hold the necessary information for conducting the simulation. This reads all the definitions and instructions in the MML file and loads into memory all external dependences like graphics and 3D models. It also generates geometry for supporting the environment and its cells. Many resources like textures and meshes are optimized by keeping just one instance in memory and using it whenever required. At the end of the process the realtime agents find in this layer all the required information to work. The transition from the MML file to the MANDALA Objects is shown in Figure 10.

Each type of data object is stored in a specialized structure that facilitates the work of agents in the upper layers. Collision geometry, for example, is kept on a dynamic plane shifting BSP[2].

[3]The parsing is implemented with Xerces-derived objects. Xerces is part of the *Apache XML Project*[1]

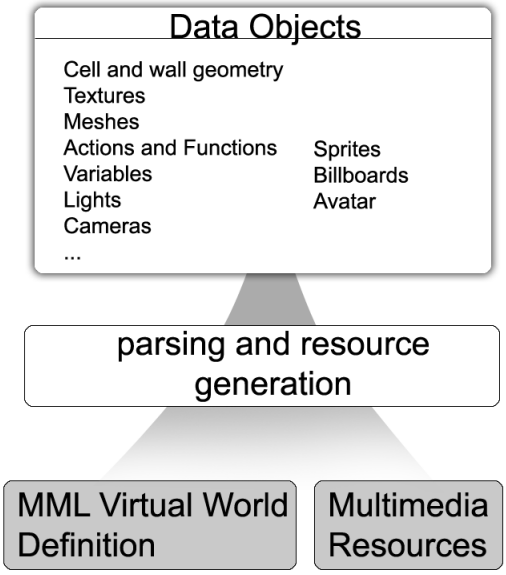

Figure 10: Process of data generation for the use of RealTime engine

3.4 RealTime Agents: Bringing the simulation to life

The fourth layer of the architecture holds a group of dedicated agents that work with the data maintained in the third layer for various dynamic purposes. This control center is where the real mechanics of the engine take place and where the environment is set in motion. Several agents sometimes work with common information but each one is completely independent of the others. Some examples of these agents are:

- *Avatar agent*: moves the user across the world, reads input from the external control devices (Like mouse or keyboard) and updates the position of the avatar. Controls factors like pace speed and camera settings.
- *Collision Detection agent*: reviews the position of moving objects at each time frame and calculates collisions. Sends update messages to agents controlling moving objects to rectify position when a collision is detected (avoiding objects to trespass walls, for example).
- *Navigation agent*: this entity is like an invisible character observing everything that happens in the simulation. It is mainly designed to take notes and produce certain reports with data collected from each experiment.
- *Actions agent*: constantly monitors events in the maze. Should a doc-

umented event trigger some action (specified in the MML file), this agent produces a message, reviews the action and takes pertinent measures.

- *Message agent*: performs callbacks for certain messages and distributes information across other modules.
- *Remote communications agent*: establishes and maintains connections with remote agents so that interaction between MANDALA and other domain-specific agents can take place.
- *Rendering agent*: provides an interface with the fifth layer of the architecture, orchestrates and optimizes the rendering process. This agent is, nevertheless, independent from the chosen graphic environment and library.

This architecture allows the complete replacement or addition of agents without altering other elements. Consequentially, it allows scalability and enhancement of functionality with no added pain.

3.5 Rendering the World

It was mentioned before that the rendering agent directs the process of showing the virtual environment to the user, but remained abstracted from any graphic library or environment. A convenient analogy is that of a construction worker driving some heavy machinery or tools (the agent is able to operate different kinds of construction tools without necessarily understanding their inner functionality). It is here where such tools reside and are utilized. The fifth layer of the architecture provides means for 3D graphical representation and is strongly based on some particular graphic library. One of the greatest advantages is that exporting the whole architecture to a different operating system or environment is just a matter of substituting this layer.

It is also here where different rendering methods reside. Currently simple-monitor rendering and stereo rendering are supported. In stereo rendering the screen is divided in two sections (usually left and right), each one showing the scene with an interocular difference of approximately 6 cm. Specialized hardware take these images and mix them to present the user with the illusion of a *real* 3D image.

4 A simple Navigation Test

This section describes a typical Spatial Navigation test using the MANDALA system and other research-specific software and hardware elements. The general structure of the experiment is shown in Figure 11. It is necessary to mention once again that specific details on this research are not mentioned

here, only those directly related to the functionality of MANDALA are considered.

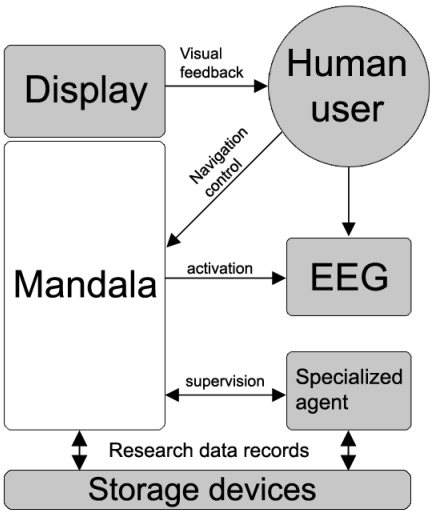

Figure 11: Structure of a typical setup for the Spatial Navigation experiment

The volunteer sits in front of a stereo display with an electroencephalographic (EEG) cap. When the experiment begins, all operations are automated. An agent specialized in this research opens a communication channel with MANDALA and orders the selection of an initial maze for the user. From now on this specialized external agent will be simply referred as the *agent* and unless otherwise noted all operations are performed by MANDALA. The required maze is loaded and the user can begin the navigation, his behavior been monitored and recorded at all times. When the user reaches the exit a notification is sent to the agent, which reads and appraises the navigation log. Based on certain rules and the user's performance a new maze can be selected from a collection or created in *realtime*, the old maze is terminated and the new one is loaded and presented to the user, who continues navigation. During this cycle certain portions of the maze activate or deactivate the EEG recorder and measurements from the user are kept for future analysis.

This experimental cycle (the user navigates the maze, MANDALA keeps a record, communicates with the agent and controls the EEG machine, the agent evaluates the user and creates/selects a new maze) repeats until certain end criteria are met, then the last maze is unloaded and MANDALA is instructed to close. The spatial navigation files for the current volunteer are

ready for analysis.

5 Using MANDALA in other Domains

It has been described how MANDALA is used to produce results for the Spatial Navigation research. The elements that are particular of MANDALA for this research are the set of instructions contained in the MML file, the design of the virtual world consisting on a series of carefully constructed mazes, and the functionality of the external agent.

It is with relatively small effort that MANDALA can be adapted to other research domains. The specific functionality of the domain can be encapsulated on the external agents that communicate with MANDALA and analyze information of interest or remotely control internal operations. In other cases, MANDALA can be immediately applied as a simple solution for virtual reality navigation (in applications where the user must walk across industrial or residential facilities, for example) taking advantage of the encapsulation of complex VR and 3D concepts made by MML, allowing designers not familiarized with such terms to creatively design and present interactive environments. Other applications include the integration of agents or modules of MANDALA in bigger or more complex implementations.

6 Future Work

A version of MANDALA that can be run on high-end CAVE Virtual Reality systems is being developed. In addition, a better and more abstracted implementation is being constructed, since the current version still overlaps some elements across layers. The complete MANDALA project is also to be used on research involving human-virtual characters interaction, so several pertinent adaptations will be done to support more realistic and sophisticated rendering, along with enhancements to the input agents so that information from magnetic sensors, cameras and other non-conventional input devices can be utilized.

References

[1] The Apache XML Project, *xml.apache.org*

[2] Stan M. *Dynamic Plane Shifting BSP Traversal* Graphics Interface proceedings. 2000

[3] David M. *Physics for Game Developers*. O'Reilly, 2002

[4] Kahana MJ, Sekuler R, and others. *Human Theta Oscillations Exhibit Task Dependence During Virtual Maze Navigation*. Nature 1999 399:781-784

[5] Howard, I.P., and Rogers, B.J. *Binocular Vision and Stereopsis*. New York: Oxford University Press, 1995.

[6] David H. Eberly, *3D Game Engine Design*. Morgan Kaufman, 1999 v