WP 4 Project Deliverables D4.1 & D4.2

Specification of methods of displaying CFD data
Specification of immersive user interface

Project Number	IST-2000-29266
Project Title	Virtual Real Time Fire Emergency Simulator
Deliverable Type	Report
Deliverable Class	Restricted

Deliverable Number	D4.1 & D4.2
Title of Deliverable	Specification of methods of displaying CFD data [WP4.3] Specification of immersive user interface [WP4.4]
Nature of the Deliverable	Report
Contributing WPs	WP 2, WP 3
Contractual Date of Delivery	02-11-01
Actual Date of Delivery	02-12-13
URL	http://www.virtualfires.org/
Authors	Kai-Mikael Jää-Aro, Tomas Weber (KTH)
Contact Details	Center for Parallel Computers Department of Numerical Analysis and Computer Science Royal Institute of Technology SE-100 44 Stockholm Sweden

Abstract	We outline the user interface for the immersive visualisation front end and enumerate the visualisation methods we will use.
Keywords	User Interface, Scientific Visualization, VR

Contents

Background 3
Communication in the system 3
Necessary functionality 4
Suggested interface 7
Communication protocol 12
References 14

Background

Within the project we have two types of immersive environments—one using a head-mounted display (HMD) as display device, the second using a CAVE system [Schneider *et al*, 2001a]. These have fairly different properties and affordances for interaction. Some important differences are the following:

- HMDs are single-user environments, whereas CAVEs accommodate several viewers.
- HMDs shut out the wearer from the outside world, it is therefore not possible to simultaneously view the virtual environment and use e.g. a keyboard (unless the user is a proficient touch typist). On the other hand the user can easily be seated in a chair in front of a keyboard, something that typically is not possible in CAVEs.
- CAVEs normally have higher resolution images than HMDs. Still, we want to minimise the development work and utilise as many common components as possible for the user interfaces for the two conditions. If we can support simple desktop versions of the system with the same interface, so much the better.

Some basic assumptions underlie our reasoning:

- There are no really effective text input devices for immersive environments, some promising prototype systems notwithstanding, e.g. [Grosjean & Coquillart, 2002].
- It is difficult to make fine adjustments in the environment, as the available tracking systems do not have sufficient spatial resolution for millimetre-scale adjustments and in addition often are subject to a considerable amount of jitter.

Based on the reasoning above we suggest the development of a basically traditional 2D GUI with menus and buttons for the interaction with the system. This can the be used in three ways:

1) In a traditional desktop environment.
2) On the desktop for HMD users so that they can either themselves, or with a helper, operate the interface using keyboard and mouse.
3) CAVE users on the other hand can use a personal digital assistant (PDA) type of device, where the same menus and buttons can be displayed, and where pen-based text and number input can be used.

A likely suggestion for PDA is the Compaq iPaq, as it can run Linux and also has support for wireless communications and thus can be easily interfaced with the rest of the environment. Furthermore, a position tracker can easily be attached to the PDA, thus not requiring any additional devices for pointing and selection in the 3D space.

We are working on a prototype interface in Java and will use Tweek [Hartling *et al*, 2002] for the communication between the PDA and COVISE.

Communication in the system

The organisation of communications in the system will therefore be as follows:

In COVISE there will be a module, which coordinates the communication between the user interface, the database and the visualisation functions. This module will dispatch user interactions to the relevant recipients and relay necessary results (including error messages) back to the user. It will be necessary to define a

communication protocol for the messages between coordinator module and the database. For the first iteration we will choose to send requests as plain text, thus simplifying debugging. A future implementation might choose to use binary messages instead, but considering that the control messages are fairly short and infrequent, there seems to be little advantage in doing so. Data transfers, on the other hand, are large and frequent and have to be binary. A preliminary definition of the protocol is shown in ***Communication protocol*** below.

However, another issue to be considered is the trade-off between security and performance. Open socket connections are vulnerable to both eavesdropping and spoofing, thus for sensitive sessions one may wish to encrypt communications, whereas standard sessions can run faster without encryption. We will not incorporate encryption in this first version, but allow its inclusion in future versions.

Necessary functionality

Deliverable D3.1 [Lenz & Reichl, 2002] specifies the structure of the simulation database. Based on this we can begin to define the necessary functionality for the user interface. The user interface has to support both control of the simulation and control of the visualisation. This gives the following functions:

- Loading pre-defined *configurations and scenarios*. A *configuration* defines the fixed contents of a tunnel including the geometry, ventilation fans, signs, etc. A *scenario* defines the vehicles, flammable materials etc in the tunnel and the boundary conditions for the simulation.
- Defining a new configuration and scenario, e.g. by adding fans, changing their properties; placing objects (cars, trains, etc) in the tunnel, setting their properties (if different from defaults), setting vents, doors etc into open/closed positions, etc.
- It may of course be questioned whether the virtual environment is the most convenient environment for placing and editing large numbers of objects, as it may be more convenient to use an off-line process for this, but we felt that it would be good to test this, not least to try out methods for interaction in immersive environments.
- Describing a *mission*, i.e. inserting mobile fire fighting equipment into an ongoing simulation, giving the fire fighters objectives and letting them act over time.
- Starting, stopping and continuing a simulation.
- Detaching from and attaching to an ongoing computation. The visualisation will receive all its data from the database, to which the output of the computation is directed, but since the computation of a single timestep may vary from “real time” to a few minutes and upwards depending on the computer hardware used for computation and the size of the simulation, it may not be possible to follow a particular computation as it proceeds. Therefore a user will need the option to initiate a computation and then leave the visualisation station for a while and return at a later time to see the results. (Or indeed, to initiate the computation from an entirely different place and then going to the visualisation station.)

- Selecting alternate mission timelines and timesteps within them. The user can at arbitrary points in time decide to change an ongoing mission, or add a new mission. Thus each timeline is always associated with a single mission. A new timeline causes a branching off of computation, so the user has to be able to shift between alternate timelines and compare them.
- In fact, we should look at methods to visualise the *differences* between two different timesteps, e.g. too see the effectiveness of an alternative extinguishing method, the results of opening a vent, etc.
- Choosing visualisation method. As set out in D3.2 [Lenz & Reichl, 2002b] there are four computed variables that are to be visualised: the three scalars *temperature*, *smoke density*, *smoke concentration* and the vector *air velocity*. In addition, geometrical objects such as cars, trains and the tunnel itself have to be visualised.

  FIGD has implemented a parallel visualisation kernel and used it to develop parallel versions of a number of visualisation algorithms:

  - Vector glyphs. A vector field is visualised as arrows. Currently prototype only.
  - Streamlines and timelines. A vector field is visualised by continuous lines from a set of starting points. Currently prototype only.
  - Line integral convolution. A vector field is visualised by distorting a texture along the vectors. Currently prototype only.
  - Isosurfaces. A scalar field is visualised by drawing the surface separating the values on either side of a threshold value. Two different algorithms for generating isosurfaces have been developed; one using an improved version of the Marching tetrahedra algorithm [Bloomenthal, 1994], the other using the shader functions in a graphics board.
  - Volume rendering. A scalar field is visualised by rendering it as a 3D texture, typically with semi-transparent voxels (volume elements). A method using shader functions is being implemented.
  - Geometry rendering. More efficient methods for rendering geometry are being implemented.

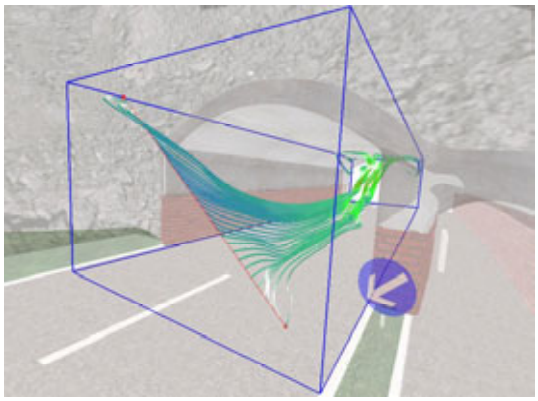

Figure 1 Streamlines representing the airflow in a tunnel. Image generated by FIGD.

In addition to these methods there are a number of other "standard" visualisation methods, such as colour-coded cutting planes, where a surface

samples scalar values and renders them in different colours according to the data values.

The "realistic" rendering of smoke will be most dependent on volume rendering, as this is the most natural way to render partially transparent objects.

For the "scientific" visualisation on the other hand, one often will use multiple visualisation methods. Each of these methods must therefore be possible to turn on and off independently. Furthermore, all of the methods have parameters that must be possible to adjust, such as isosurface thresholds, colour maps for volume rendering, cutting planes, etc.

- Navigating in the visualisation. The visualisation has to be studied from different vantage points and for systems as large as the projected Lyon-Turin tunnel [Schneider *et al*, 2001b, section 1.1.1] it will be necessary to be able to move between possibly quite distant points.
- Saving and restoring settings. When the user re-enters the system after having been disconnected, the user should be re-placed at the same position and with the same visualisation settings as at the time of disconnection.

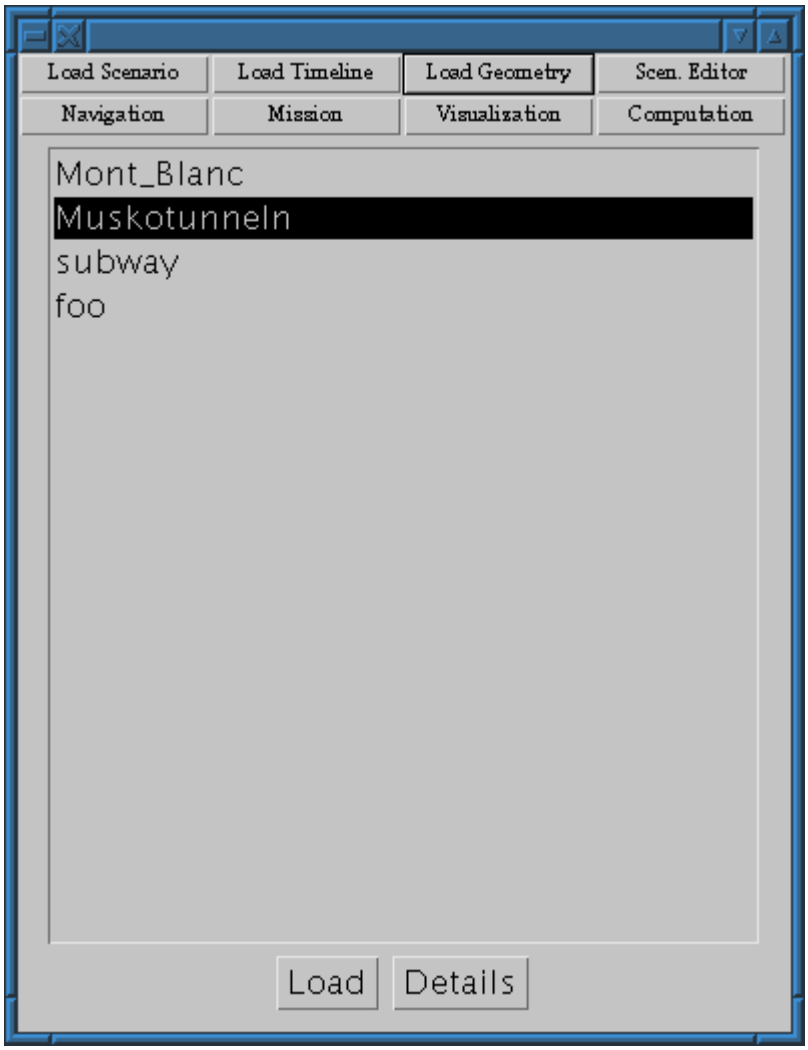

Figure 2 A list of tunnel geometries to choose from when building a scenario.

Suggested interface

As space is limited on a PDA screen, we propose a card- or tab-based interface, with important functions related to a given context gathered on a card, easily overlaid by another as interaction proceeds. These are the most important cards:

Scenario
Creating a scenario entails first choosing a tunnel geometry (see figure 2) and then placing various objects, such as cars, trains, ticket machines etc in the tunnel. The user is presented with a list of available objects (see figure 3), and if an object is selected it will be displayed in the virtual environment for the user to place. The placement is done by direct manipulation, but constraints in the environment (walls, floor, etc) see to that the object ends up on the appropriate surface inside

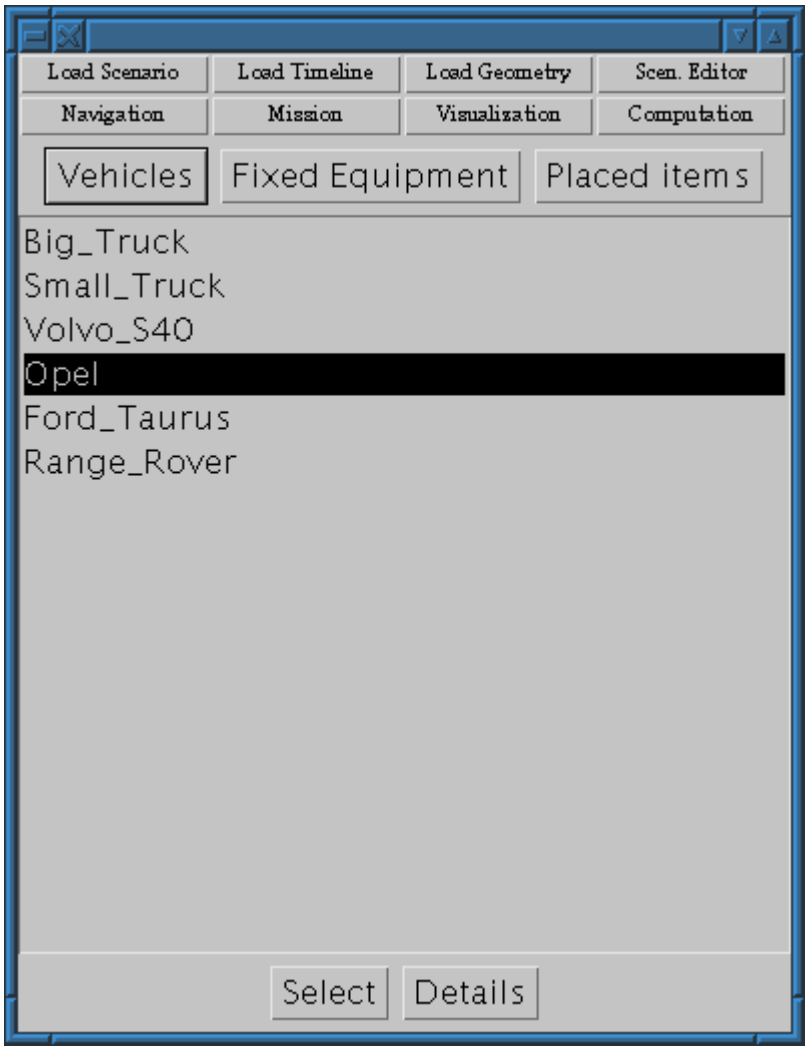

Figure 3 The user can select from a list of objects that can be placed in the tunnel.

the tunnel. The objects have to contain a field indicating whether they attach to the floor (e.g. cars), a wall (e g street signs) or the ceiling (e.g. fans). At any time from here on the object can be selected, either in a list of all items or directly in the 3D environment, and a form is brought up in the GUI so that any parameters associated with the object can be adjusted.

Items can also be copied and pasted in the environment for faster construction. Objects and their descriptions will be downloaded at once and then stored in the interface until the scenario is saved, when all objects are sent to the database at once. Further, we want to be able to set boundary conditions at all relevant points, such as tunnel entrances, air inlets/outlets, etc. The boundary conditions are set with the same type of form-filling interface as the objects.

This card also contains a "save scenario" command. This must allow, or even force, the user to document what the scenario models contains.

Similarly, the user can select from a list of existing scenarios (see figure 4). Some of these scenarios will have finished computations, some of them will have still ongoing computations, which can be attached to.

Figure 4 The user can select from a list of already defined scenarios.

Load Timeline

If the loaded configuration has more than one precalculated timeline stored in the database, this command will present the user with a graphical representation of the available timelines associated with this scenario.

The user can select a point on the timeline and transfer the visualisation to proceed from that point. This also means that the simulation proceeds from that point if

needed. It may be desirable to indicate if a visualisation is "live" or working off stored data, so that the user is aware of the expected visualisation speed.

Define mission

As this option is selected the user is presented with a list of available fire-fighting equipment and the playback of the current timeline is paused (though note that the *computation* of it will proceed). Selecting a piece of equipment will let the user to specify parameters for this piece of equipment with the usual form interface. The user can now place fixed and mobile fire-fighting equipment and define their actions and triggers, dependent on time or certain simulation variable values. These actions will then be played out over time by the simulation engine.

Adding a mission generates a new branch on the timeline, so the user will be prompted to add his/her comments to describe this timeline as the changes are submitted to the simulation engine.

The software will not enforce any limits on the number of fire fighters or their equipment but will leave this to a human training leader to oversee.

A computation is always limited to a given number of timesteps, set by the user in order to avoid runaway processes. (The number of timesteps is felt to be a more natural measure than the time for computation, even if a maximum time may be enforced by the computer system.)

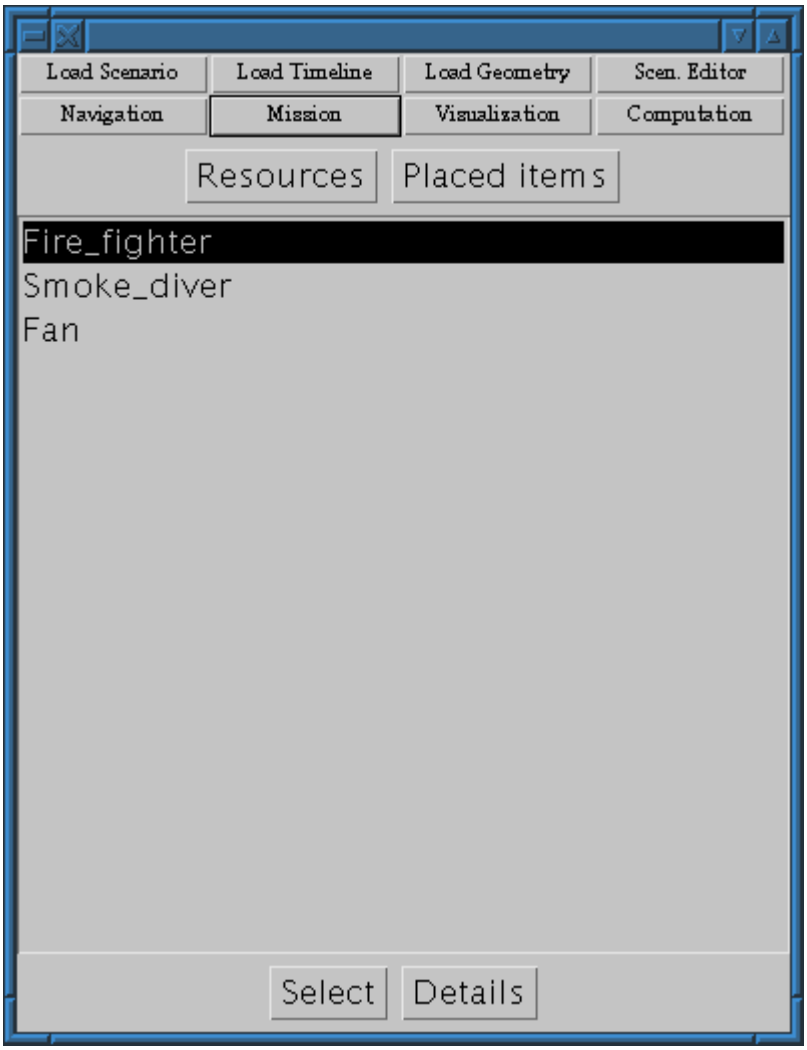

Figure 5 The user can select from a list of fire-fighting equipment to construct a mission.

Computation management
On detaching the user can give a name to the computation, to identify the correct one to attach to.
When attaching, the user gets the list of ongoing computations and can choose to attach to any of them (see figure 6).

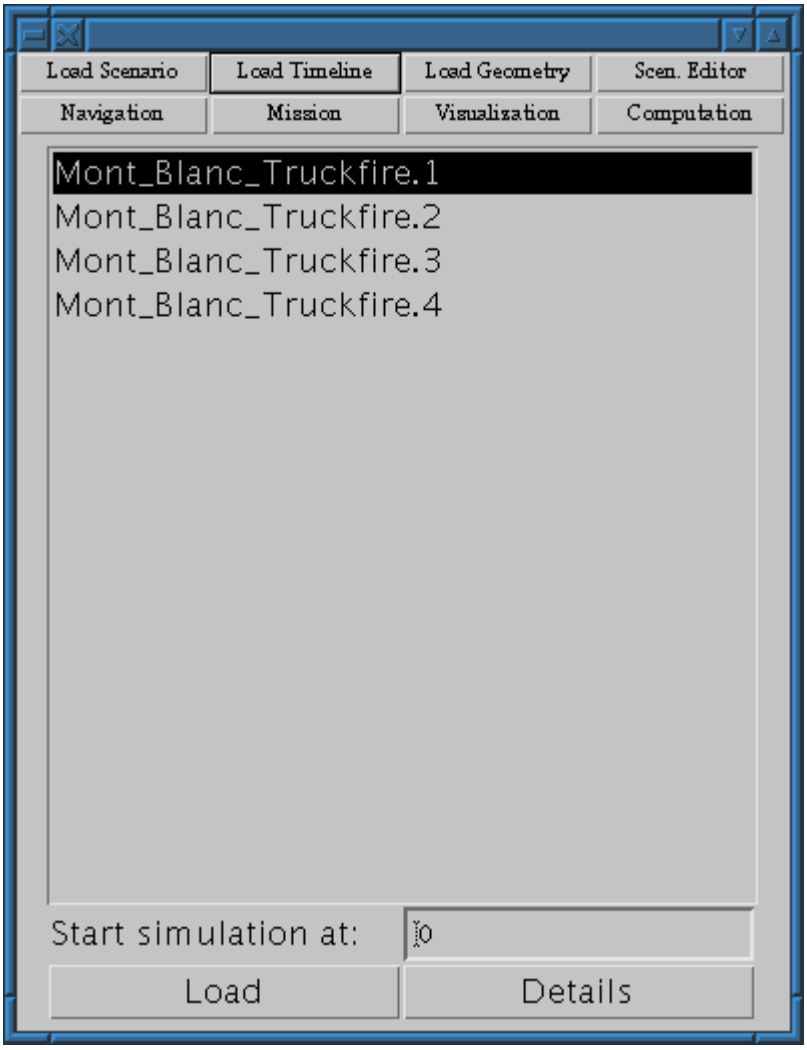

Figure 6 A list of timelines (missions) within the current scenario that the user can elect to visualise.

The database should incorporate a permissions system, so that a user only can access computations that are "open" to that user. Preferably this should include separate read and write permissions, so that a given user may see a visualisation of a given computation, but not be allowed to modify any parameters or initiate new computations.

Visualisation
The identity of each variable to be visualised is received from the database and displayed in a list. Depending on their type (scalar/vector) a list of choices of visualisation method is generated for each variable, where each choice can be checked. To set and modify parameters for a given visualisation, one can select the visualisation method and get a new form with the values for that visualisation—seed points and track length for streamlines; number of surfaces and threshold values for isosurfaces, etc.

It might turn out that one will need to be able to have several visualisation forms visible at the same time, in order to be able to adjust parameters in a synchronised manner. If this is so, it may be more efficient to get a page that actually contains all parameters for all currently active visualisations.

We use a video player-type metaphor for moving through the time of the visualisation, we may play forward, reverse, and change the speed of playback (i.e. the number of timesteps skipped for each frame displayed).

Navigation

As mentioned above, tunnel systems may be large and complex, and we may need a battery of navigation methods. COVISE supports "flying" and "walking" motion as well as "viewpoints"—points in the environment that can be set and returned to. In addition to this we will use a variation of "worlds in miniature"-style navigation [Stoakley *et al*, 1995], where we present the user with an overhead projection of the tunnel. Clicking on a point in the projection will move the user to that location.

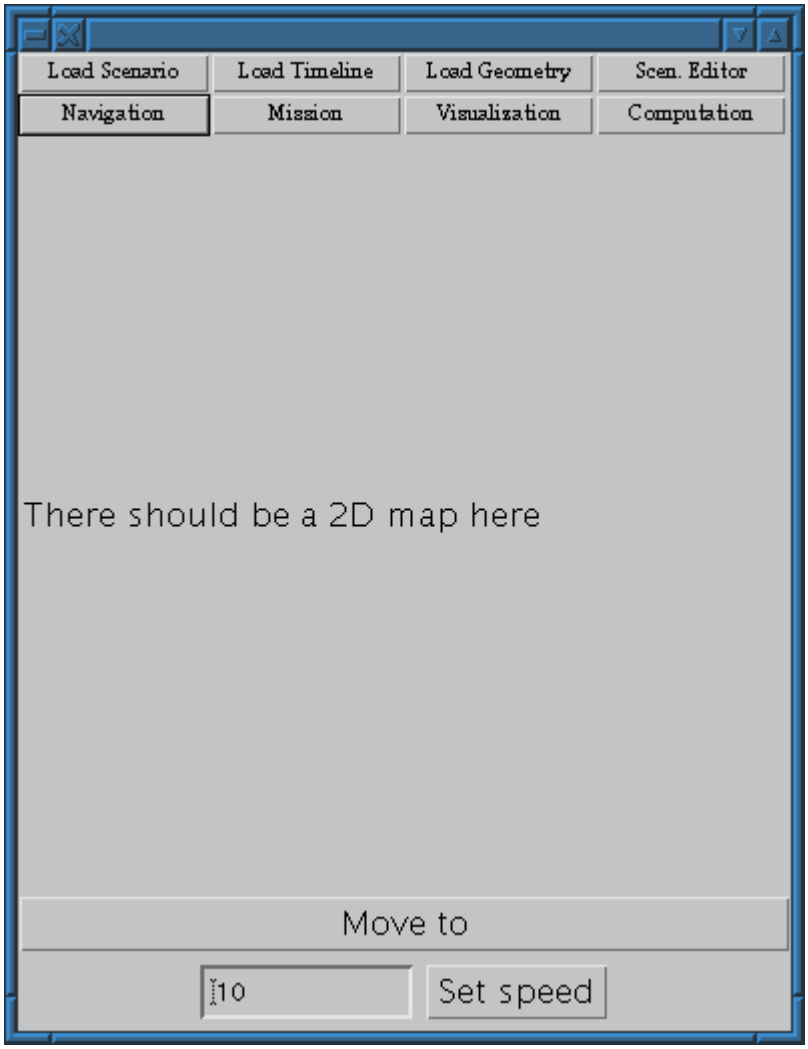

Figure 7 In order to rapidly move across a large area, the user can click on points in a 2D projection of the tunnel and be instantly transported to the indicated spot. By dragging with the stylus, a view direction can be indicated as well.

Communication protocol

The user interface sends requests to the coordination module in COVISE, which then relays these to either the database or the visualisation functions. These in turn will respond with either data to be displayed or OK/error messages.

Message	*Recipient*	*Function*	*Response*
`request tunnels`	Database	Get (possibly empty) list of tunnel geometries.	`tunnels <id`$_1$ … `id`$_n$`>`
`request modobjects`	Database	Get (possibly empty) list of modifiable objects (cars, fans, fire fighters, etc).	`modobjects <id`$_1$ … `id`$_n$`>`
`request object` *`id`*	Database	Get description for specific object.	`object` *`id`* `XML` *`info`* *`object notfound`*
`request blob` *`id`*	Database	Get specific binary data (typically geometry).	`blob` *`id data`* *`blob notfound`*
`save scenario` *`id XML_info`*	Database	Send all information pertinent to this scenario as XML data. The scenario is stored in the database under the name *name*.	`save OK` `save Error` *`nnn`* `Message`
`place object` *`id blob restrictions`*	Visualiser	The geometry *blob* is to be placed in the environment, subject to placement restrictions *restrictions*.	`position object id <x y z> <`ρ φ θ`>`
`selected` *`id`*	UI	Object *id* has been selected by the user.	
`selectpt <`*`x y z`*`>`	UI	Point <*x y z*> has been selected by the user.	
`request scenarios`	Database	Get (possibly empty) list of scenarios.	`scenarios <ID`$_1$ … `ID`$_n$`>`
`request timelines` *`scenario`*	Database	Get XML description of timelines for scenario *scenario*.	`timelines` *`XML_data`*
`start` *`scenario timeline limit`*	Simulation engine	Start computation of timeline *timeline* in scenario *scenario*. Run for max *limit* timesteps.	`start OK` `start Error` *`nnn`* `Message`
`stop scenario timeline`	Simulation engine	Stop computation of timeline *timeline* in scenario *scenario*.	`stop OK`
`request computations` *`scenario`*	Database	Get the running computations for scenario *scenario*.	`computations <id`$_1$ … `id`$_n$`>`
`save timeline` *`scenario`*	Database	Save the definition of the timeline *timeline*.	`save OK`

Message	*Recipient*	*Function*	*Response*
`parent_timeline timeline XML_data`		*parent_timeline* may be NULL.	`save Error nnn Message`
`request timestep timeline time`	Database	Get the simulation data for timestep *time* of timeline *timeline*.	`data`
`request timestep timelineID time` <x_m, x_M, y_m, y_M, z_m, z_M>	Database	Get the simulation data for timestep *time* of timeline *timelineID* for the given subset of space.	`data`
`save profile XML_data`	Database	Save the user's current settings, including position in world, visualisation parameters, current scenario, timestep, etc.	`save OK` `save Error nnn Message`
`request profile`	Database	Load the user's settings.	`profile XML_data`
`request scenario variables`	Database	Get the variables as `XML_data` involved in this simulation and their datatypes.	`variables XML_data`
`request method type`	Visualiser	Get the possible visualisation methods for this data type and the parameters for the methods.	`method` <$name_1$ `XML_data`$_1$ … $name_n$ `XML_data`$_n$>
`set method variable method settings`	Visualiser	Use visualisation method *method* for variable *variable* with parameter settings *settings*.	`set OK` `set Error nnn Message`
`moveto` <x y z> <ρ φ θ>	Visualiser	Move to point in environment.	`moveto OK`

Notes:

- The syntax for object identifiers is the usual: an alphabetic character followed by alphanumerics and underscore, case is significant. All identifiers have to be globally unique.
- Poses are given as translation and rotation.
- Error messages both give a numeric value for the interpretation of software, as well as a text string which can be presented to a user and thus is configurable with respect to language, user category, etc.

References

[Bloomenthal, 1994] “An Implicit Surface Polygonizer”, Jules Bloomenthal, in *Graphics Gems IV*, 1994.

[Grosjean & Coquillart, 2002] “Quikwriting on the Responsive Workbench”, Jerome Grosjean & Sabine Coquillart, in *SIGGRAPH 2002 Conference Abstracts and Applications.*

[Hartling *et al*, 2002] “Tweek: Merging 2D and 3D Interaction in Immersive Environments”, Patrick Hartling, Allen Bierbaum & Carolina Cruz-Neira, in *Proceedings of 6th World Multiconference on Systemics, Cybernetics, and Informatics*, 2002.

[Lenz & Reichl, 2002a] *WP 3 Project Deliverable D3.1—Specification of Geometrical Database*, Gunther Lenz & Thomas Reichl, 2002.

[Lenz & Reichl, 2002b] *WP 3 Project Deliverable D3.2—Specification of CFD Database*, Gunther Lenz & Thomas Reichl, 2002.

[Schneider *et al*, 2001a] *WP 2.1+2.3 Project Deliverable—Report on available developer tools, Report on available system capabilities (hardware)*, Sascha Schneider, Michael Schmidt, Thomas Reichl, Kai-Mikael Jää-Aro, Björn Sjögreen, Gert Svensson, José Luis Ajuria, Christian Redl & Wilhelm Brandstätter, 2001.

[Schneider *et al*, 2001b] *WP 2.4 Project Deliverable—Specification of planned system capabilities (software)*, Sascha Schneider, Michael Schmidt, Volker Luckas, Gunther Lenz, Thomas Reichl, Wilhelm Brandstätter, Christian Redl, Gert Svensson, Kai-Mikael Jää-Aro, Alain Bochon, René-Michel Faure, Klaus Schäfer, Rainer Koch, José Luis Ajuria & Christian Redl, 2001.

[Stoakley *et al*, 1995] “Virtual Reality on a WIM: Interactive Worlds in Miniature”, Richard Stoakley, Matthew Conway & Randy Pausch, in *Proceedings of CHI '95*, pp 265 – 272 , 1995.