WP 4.5 Project Deliverable
VR implementation with limited functionality and speed

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

Deliverable Number	D4.5
Title of Deliverable	VR implementation with limited functionality and speed
Nature of the Deliverable	Source code
Contributing WPs	WP 4
Contractual Date of Delivery	2003-05-31
Actual Date of Delivery	2003-10-31
URL	www.virtualfires.org
Authors	Kai-Mikael Jää-Aro (KTH)
Contact Details	Institute for Structural Analysis / SiTu Research Univ. Prof. Dipl.-Ing. Dr. techn. Gernot Beer Lessingstrasse 25/II 8010 Graz / Austria Tel.: +43 316 8736180 Fax: +43 316 8736185 Email: gernot.beer@ifb.tu-graz.ac.at

Abstract	This report describes the prototype of the VR-implementation including visualization methods and user interface functions.
Keywords	source code, user interface, visualisation

Overview

This report describes the current functionality of the VR system components developed by KTH and FIGD for the VIRTUALFIRES simulator. Relevant source code is attached on CD-ROM.

Components

The components in question are the following:

- The Graphical User Interface. This is the part of the system with which the user interacts and which initiates all other actions in the system.
- The User Interface Controller (UIC). This handles the communication between all other components.
- The geometry handler (geomHandler). This handles the placement and interaction with all graphical objects that are not strict data visualisation objects, i e the tunnel model, vehicles, fire-fighting equipment etc.
- The visualisation function (HVR). This does the visualisation of the scientific data—smoke, fire, airflow, etc.

The base platform for visualisation is COVISE, a modular visualisation system. The rendering module is called COVER. It uses OpenGL Performer as its rendering API. COVER can be extended with *plugins*, user-developed code that can be dynamically linked into a running session. The plugins can access the COVER scene graph to add and modify objects, and read the input devices. Plugins can send messages to each other.

GUI

The GUI is implemented as a Java application and designed to run on both PDAs and desktop computers.

The GUI has been rewritten to a large extent since the first version. The functions of the user interface have been reworked in order to be more intuitive. The definition and editing of missions have been redesigned, as has the way the selection of visualisation methods is done. The added functionality allows the user to:

-Select an existing mission definition in the database.
-Visualise a computation that has been stored in the database.
-Adjust the speed of visualisation of that computation, forwards and backwards.
-Restart the computation of a given mission.
-Start a new computation for a given mission.
-Stop an ongoing computation.

The communication protocol between the GUI and the UIC has been refined to better match the needs of the users as these have become more obvious during development. The communication protocol is included in an appendix.

UIC

The UIC is a COVER plugin. Its primary purpose is to receive commands from the GUI and forward them to the relevant receiving components, primarily the Data Manager Controller (see deliverable D3.5 for the details on the communication between UIC and DMC). The UIC has been continuously updated to match the communication requirements of all other components, so the

new functionality in UIC is the communication functions for the new functionality in GUI, DMC, geomHandler and HVR. The ambition has been to keep little knowledge in the UIC code itself, but rather let the objects contain sufficient knowledge that processing can be as data-driven as possible.

geomHandler

As one of the aims of the VIRTUALFIRES project has been realistic representation of not only fire, but the actual environment where the fire occurs, a number of 3D models of tunnels, vehicles, fire-fighting equipment and so on have to be inserted in the visualisation. geomHandler is a COVER plugin, receiving commands from the UIC. For each simulation scenario the set of comprising objects is retrieved from the database and the geomHandler inserts them into the scene graph. The geomHandler is a new addition since the previous prototype and has been tested in a number of versions to find the most efficient way of handling the graphical data.

HVR

The HVR is a COVER plugin that does the visualisation of the computed data. It does highly efficient parallel hardware-based rendering of:

- Particles, these can be displayed as streamlets or geometry primitives as well as textured billboards.
- Volumes, by using an appropriate colour mapping, fire and smoke can be rendered with a granularity based on the simulation data. Now a refinement of this method uses a random distortion field (“ noise ”) to add details into the scenery with a higher resolution level.

The code does not currently run on the KTH Onyx as it lacks the necessary pixel shader hardware. The communication protocol between HVR and the UIC is given in an appendix.

Appendix: Communication UIC-GUI

This is the communication protocol between the GUI and the UIC. The messages are sent as UDP messages in ASCII format.

Request	***Response***	***Response types/comments***
Request Scenarios	***Name0 ScenarioID0 Name1...***	***String Integer ...***
Request Missions "ScenarioID" "TunnelID"	Name0 TimelineID0 EndingTime0 Name1...	String Integer Integer ...
Request Objects "TimelineID" "time"	"Ok"	
Computation Start "MissionID"	"Ok"	
Computaion Continue "MissionID"	"Ok"	
Computation Stop "MissionID"	"Ok"	
Computations Running	Name0 ID0 Name1 ...	String Integer ...
Request Description * "*ID"	Description	String * can be: Scenario Mission Tunnel? Template Computation Object
Request TemplateObjects "ClassID"	"Name0" ObjectID0 "Name1" ...	String Integer ...
Request PlacedObjects "ClassID"	"Name0" ObjectID0 "Name1" ...	
Request TemplateObject Mission "TemplateID"	ObjectID timestamp Parameters (mission_edit tags) (1)	According to the parameter protocol. For template objects the name of the object is to be sent as the first parameter. Since it is a template object there is no objectID, so the UIC chooses one.
Request TemplateObject Scenario "TemplateID"	ObjectID Parameters (scenario_edit tags)	According to the parameter protocol. For template objects the name of the object is to be sent as the first

Request	*Response*	*Response types/comments*
		parameter. Since it is a template object there is no objectID, so the UIC chooses one.
Request Object Mission "ObjectID"	ObjectID timestamp Parameters (mission_edit tags)	According to the parameter protocol
Request Object Scenario "ObjectID"	ObjectID Parameters (scenario_edit tags)	According to the parameter protocol
Request Mission Placed	Name0 ID0 Name1...	Union of list of instantiated object of type firefighters and mobile firefighting equipment
Request Scenario Placed	Name0 ID0 Name1...	
Update Scenario "ScenarioID"	"Ok"	This command will be needed as users can still edit scenario parameters that don't affect the grid
Update "ObjectID" timestamp Mission Parameters	"Ok"	Only the values of the parameters are sent. In the **same** order that they were received. If the objectID indicates a template object being instantiated, then the first parameter will be the name.
Update "ObjectID" Scenario Parameters	"Ok"	Only the values of the parameters are sent. In the **same** order that they were received. If the objectID indicates a template object being instantiated, then the first parameter will be the name.
Dissociate Object "ObjectID" "MissionID"	"Ok"	
Delete Mission "MissionID"	"Ok"	
Delete Scenario "ScenarioID"	"Ok"	
Set Mission Endpoint "MissionID" "endpoint"	"Ok"	
Set Mission Resolution "MissionID" "resolution"	"Ok"	

Request	*Response*	*Response types/comments*
"resolution"		
Request Computation Parameters "MissionID"	Endpoint Resolution	Float Float
Playback Play	"Ok"	
Playback Pause	"Ok"	
Playback Stop	"Ok"	
Playback FastForward	"Ok"	
Playback FastRewind	"Ok"	
Request Tunnels	Name0 TunnelID0 Name1...	String Integer ...
Request Tunnel "TunnelID"	"Ok"	
Request Event List "MissionID"	"Name0" ID0 "Name1"...	Passed on to the DMC with an mID added first
Request Event List Past "MissionID"	"Name0" ID0 "Name1"...	Passed on to the DMC with an mID added first and the current time added last
Request Event List Future "MissionID"	"Name0" ID0 "Name1"...	Passed on to the DMC with an mID added first and the current time added last
Save Scenario "TunnelID" "Name" Description	"Ok"	
Request Visualization Data	*FieldID0 FieldName0 UnitName0 Type0...*	fieldName and unitName have to be in double quotes
Get_Available_Vizmethods VisType	Name0 Name1...	VizType is scalar, vector or all. The names returned are unique.
Activate_New_Probe VizMethodName FieldID	*ProbeID Name0 Range0 Type0 Value0 Name1...*	When a new probe is instantiated the UIC must request the parameters
Deactivate_Probe id	Ok/errormessage	
Get_Active_Probe_Parameters id	*Name0 Range0 Type0 Value0 Name1...*	(1)
Set_Active_Probe_Parameters id Value0 Value1 ...	Ok/errormessage	
Get_Active_Probe_Ids ReqType Name	*id0 id1...*	See UIC-VIZ protocol
Get_Active_Probe_Vizmethod_Name id	Name	
Get_Active_Probe_Visibility_State id	visible/invisible	

Request	*Response*	*Response types/comments*
_State id		
Show_Active_Probe id	Ok/errormessage	
Hide_Active_Probe id	Ok/errormessage	

Errors are always sent as Error "…" and can always be the response instead of the specified response to a request.
Observe space between "Error" and error string.

(1)The format of parameters is:
name minrange maxrange type value
"time x NIL float y"
"time NIL NIL choice yes no selected no"

Appendix: Communication UIC-HVR

Protocol for the Communication between UIC and Viz Cover Plugin. The corresponding commands and parameters will be exchanged as strings using the " sendMessage " function. The particular commands and parameters are separated by space. There are 2 things to distinguish: available visualization methods (= not yet instantiated) and active probes (= a box in the simulation space carrying a Vizmethod).

"mID Get_Plugin_Version"
"mID Plugin_Version Version"

Parameters:
mID The message ID
Version The current version of the plugin (includes supported protocol version info)

Function:
Get the version of the plugin.

"mID Get_Available_Commands"
"mID Available_Commands CommandList"

Parameters:
mID
CommandList

Function:
Get a list of available plugin commands.

"mID Get_Available_Vizmethods VisType"
"mID Available_Vizmethods Name0 Name1 …"

Parameters:
mID
Name#
VisType#
based Vizmethod or you want to receive all Vizmethods available.

Function:
Get an overview of the available Vizmethods.

"mID Activate_New_Probe VizMethodName FieldID"
"mID New_Probe_Activated id"

Parameters:
mID
VizMethodName Valid name for a Vizmethod received through *Get_Available_Vizmethods*
id
FieldID
Initialize

Function:
Activate a new probe carrying a certain Vizmethod. The default size of the probe will be

the same as the whole dataset has. Default parameters of the probe are used. The id is always in the form " Probe_[number] " with [number] being an integer. This tag will as well be stored in the corresponding performer scene graph node (to allow easy identification of nodes representing a probe).

"mID Deactivate_Probe id"
"mID Probe_Deactivated"

Parameters:
mID
id

Function:
Deactive a probe.

"mID Get_Active_Probe_Parameters id"
"mID Active_Probe_Parameters Name0 Range0 Type0 Value0 Name1 Range1 Type1 Value1 …"

Parameters:
mID
id
Name#
Range#
" NIL " means: no limit (e.g. " NIL-NIL " = no lower and no upper limit) or no one can be defined (e.g. for strings)
Type# " *integer* " *or* " *double* " *or* " *string* " *or* " *choice* "
Value#
For " integer " :
current integer value
For " double " :
current double value
For " string " :
current string value
For " choice " :
choicepossiblity1 choicepossiblity2 … selected choicepossibility

Function:
Get the current parameter values for a certain active probe.

"mID Set_Active_Probe_Parameters id Value0 Value1 …"
"mID Parameters_Set_For_Probe"

Parameters:
mID
id
Value#
Type and the range of the parameter (see *Get_Available_Parameters_For_Vizmethod*
Furthermore they must be specified in same correct order as *Get_Active_Probe_Parameter*

Function:
Get_Active_Probe_Parameters

"mID Get_Active_Probe_Ids ReqType Name"
"mID Active_Probe_Ids id0 id1 …"

Parameters:
mID
id#
ReqType
Name
In case of " by_vizmethod " this parameter has to be a name of a *Get_Available_Vizmethods*
will filter the reply Ids according to the given vizname
in case of " by_field_id " this parameter has to be a valid field ID
Initialize

Function:
Get a list of the current active probe ids (which use vizmethod Name).

"mID Show_Active_Probe id"
"mID Probe_Shown"

Parameters:
mID
id

Function:
Show a probe.

"mID Hide_Active_Probe id"
"mID Probe_Hidden"

Parameters:
mID
id

Function:
Hide a probe.

"mID Get_Active_Probe_Vizmethod_Name id"
"mID Probe_Vizmethod_Name Name"

Parameters:
mID
id
Name *id*
Get_Available_Vizmethods

Function:
Get the name of the Vizmethod for a probe.

“mID Get_Active_Probe_Position id”
“mID Probe_Position x0 y0 z0 x1 y1 z1”

Parameters:
mID
id

Function:
Get the location in space of a probe.

“mID Set_Active_Probe_Position id x0 y0 z0 x1 y1 z1”
“mID Probe_Placed”

Parameters:
mID
id
x0, y0, z0
simulation space
x1, y1, z1
simulation space

Function:
Place the probe in the VR / simulation space.

“mID Get_Available_Parameters_For_Vizmethod Name”
“mID Available_Parameters_For_Vizmethod Name0 Range0 Type0 Name1 Range1 Type1 …”

Parameters:
mID
Name#Name of Parameter # (String)
Type#“double” or “integer” meaning the type of the parameter
Range#“0.0-10.0” or “0-10” meaning the valid range of the parameter.
“ NIL ” means: no limit (e.g. “ NIL-NIL ” = no lower and no upper limit)

Function:
Get an overview of the available parameters for a certain Vizmethod.

“Time_Step_Ready FieldID0 FieldName0 UnitName0 Type0 FieldName1 UnitName1 Type1 ...”

Parameters:
FieldID#
FieldName#Abstract name of Field # (String) e.g. “temperature”
UnitName#Abstract name of Unit of Field # (String) e.g. “°C” or “K”
Type#Type of Field # which must be „vector“ or „scalar“

Function:
This reply will be sent every time the plugin has finished reading a timestep.

“mID Get_Active_Probe_Rotation id”
“mID Probe_Quaternion x y z alpha”

Parameters:
mID
id
x y z alpha

Function:
This reply will be sent every time the plugin has finished reading a timestep.

“mID Initialize“
“mID Initialized“

Parameters:
mID

Function:
Initialize the Visualization plugin with the names and types of the available fields – must be called once for every scenario at the beginning of the visualization process by the UIC.