VIRPI: A High-Level Toolkit for Interactive Scientific Visualization in Virtual Reality

Desmond GERMANS[1], Hans J.W. SPOELDER[1], Luc RENAMBOT[1] and Henri E. BAL[1,2]

[1] Division of Physics and Astronomy
[2] Division of Mathematics and Computer Science
Faculty of Sciences, Vrije Universiteit
De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands

Abstract. Research areas that require interactive visualization of simulation data tend to dismiss virtual reality due to the lack of accessible tools for application specialists. This paper presents an integral toolkit for interactive visualization in virtual reality environments. The toolkit defines a framework to build applications that allow the user to interact with arbitrary simulation software and describe virtual measurement tools for the visualized data. The approach is illustrated with a case study in medical imaging.

1 Introduction

Current trends in virtual reality show that interaction and collaboration are key research topics[14]. However, researchers working with scientific simulation and visualization seldom use these new techniques in practice[17]. This is partially due to the lack of available software.

Visualization is commonly done using a low-level graphics library and specific tailor-made libraries that supply collaborative and interactive aspects. These libraries are not suited for application experts in fields of research that are not related to virtual reality. Even though it is possible to combine existing tools and libraries to present applications in interactive scientific visualization, non-VR experts would benefit greatly from an overall high-level approach, aimed at their application domain.

This paper describes a high-level toolkit that implements interactive paradigms for visualization, analysis and measuring with virtual instruments[15]. The toolkit is aimed at non-VR experts who wish to explore simulation data using virtual reality environments, towards the idea of "Virtual Laboratories"[8]. In the rest of the paper, we will discuss related work in section 2. Section 3 describes two key issues in the acceptance of VR outside the VR community: direct interaction with a simulation program and quantifying observations in VR. With requirements from these issues, the functionality of the toolkit is described in section 4. Then, a detailed case study shows the use of the toolkit in section 5. The paper finishes with conclusions and pointers to future work in section 6. The contributions of this paper are:

- It presents a high-level interactive visualization toolkit aimed at non-VR experts.
- The toolkit integrates access to measuring paradigms and the interaction with a running simulation program.
- It demonstrates the toolkit using a real-world case study from dentistry research.

2 Related Work

tool	Scene Graph	Data Vis.	Interaction	Collab.	Multi-Platf.	VR Hardware
OpenGL	no	no	no	no	SGI+Lin.+Win.	no
Direct3D	D3DRM	no	DirectX	DirectX	Win. only	no
IRIS Performer	supported	with VTK	low-level	no	SGI+Linux	with CAVElib.
Inventor	supported	no	low-level	no	SGI+Lin.+Win.	no
VTK	no	supported	no	no	SGI+Lin.+Win.	no
CAVELibrary	no	no	low-level	low-level	SGI+Linux	supported
VR Juggler	no	no	low-level	low-level	SGI+Linux	supported
CAVERNsoft	no	no	no	supported	SGI+Lin.+Win.	with CAVElib.

Table 1. The various available libraries and their features with respect to interactive data visualization in VR.

Approaches in scientific visualization are mostly based on either application builders (like IBM's DX, or AVS[2]), application programming aids (like the Visualization Toolkit[12]) or proprietary programming. Because application builders are generally not available for immersive virtual environments, one is left with programming graphics and accessing tracking devices. A low-level starting point to address visualization issues is the widespread OpenGL graphics standard. When a scene graph (hierarchical ordering of 3D objects) is required, one can use IRIS Performer[11], a high-performance visual simulation library. IRIS Performer provides a wide variety of functionality, and is very general. However, it can only be used on IRIX machines and with limited functionality also on Linux machines. To visualize relationships and constructions, to derive data, add color, etc. the Visualization Toolkit (VTK) can be used, which runs on top of OpenGL. VTK is a complete toolkit, providing implementations of many algorithms with which the programmer can do extensive data visualization. Based on a dataflow model, where the programmer can connect objects into a data-flow graph, VTK supports data derivations, isosurface extraction, various coloring schemes, etc. However, VTK alone is not aimed at interactive and collaborative virtual reality.

Interaction in virtual environments is commonly handled through the CAVElibrary, a toolkit to access CAVE hardware (trackers and displays). An alternative to this is VR Juggler[4], an open source library that addresses similar issues. Both libraries present a small layer around the tracker hardware and ways to channel graphics on the various screens in the virtual reality setup. The user is left to program visual interaction and navigation aids for analysis and measurement.

On a higher level, Open Inventor[18] provides interactive facilities, and offers an extensive library of scene-graph related tools. Furthermore, Open Inventor is an interactive runtime system and defines the inventor file format, which is used as a basis for VRML. However, Open Inventor was not tailored for VR environments.

Collaboration is supported through numerous libraries that provide network functionality[1, 3, 5, 7, 13]. Among them, CAVERNsoft[6] presents a network infrastructure

and is specialized in telepresence features, like support for several protocols, shared data management, high throughput network, avatar realization and audio/video streaming. GNU/Maverik[10] and Avocado[16] present flexible communication frameworks for larger-scale collaboration in virtual environments. Like CAVERNsoft, several issues to realize telepresence are addressed. All of these libraries approach telepresence issues at a relatively low-level. At this level, describing communication for the task is difficult for non-VR experts.

Table 1 shows features that are supported by the various libraries. Note that the table also shows a proprietary Windows-based library, Direct3D. With an increasing trend to use off-the-shelf PC clusters (for tiled displays or CAVE-like setups), the possibility to run the software in a Windows-environment becomes an issue. In Windows, most of the necessary tasks are covered by DirectX, a proprietary entertainment-related high-performance peripheral library. Direct3D is the 3D graphics part of DirectX and is to a large extent functionally comparable with OpenGL. Retained Mode Direct3D (D3DRM in the table) is a scene graph library on top of this, providing the basic needs for scene graph management. Interaction and collaboration for Windows-based machines can be handled by DirectInput and DirectPlay. Again, this only runs on Windows-based machines, and uses limited proprietary protocols and formats.

All of the above issues are described from a technical specialist's point of view. They require knowledge on the structure of computer graphics, the issues of network communication and human-computer interaction aspects. As seen from table 1, data visualization and VR hardware support can be combined with IRIS Performer. For this purpose, special versions of VTK, the CAVElibrary and VR Juggler are available. Also, IRIS Performer can cooperate with CAVERNsoft. A combination of these libraries would thus present the tools needed to write interactive visualization applications for VR environments. However, a user who is not to some extent expert in these issues will dismiss the available libraries and toolkits.

3 Motivation

Ideally, a non-VR expert would rely on the functionality of one toolkit. This toolkit provides communication, measurement paradigms and interactive VR. With very little programming effort, the programmer would create an interactive VR application that suits the needs of the apparent field of research.

To develop such a toolkit, two main steps can be identified. First, use the available toolkits to arrive at a platform-independent base level. This level should hide the platform differences between a range of different setups, and it should provide basic visualization-oriented features, like a scene graph, simple shapes, text, graph data loading, accessing trackers, etc. In our approach, we call this step *Aura*. For the second step, we need to assess and build high-level tools to provide a measuring environment on top of the base level. This takes on the shape of an interactive framework with paradigms to manipulate and measure graphical representations of data. In this paper, we describe a toolkit, *VIRPI*, that facilitates this second step.

Two aspects in the acceptance of VR outside the VR community are important:

First, *the user wishes to interact seamlessly with a remotely running simulation program.* The user wishes to rapidly understand phenomena the simulation program is mimicking. Therefore, it is highly unwanted to alter the actual source code of the simulation program by inserting probing and analyzing instructions. In some cases this is not even possible, as the source code is not available, or altering the source would introduce erroneous behavior. For VIRPI, this problem is addressed by coupling VIRPI to CAVEStudy[9]. CAVEStudy wraps an unaltered remotely running stateless simulation program, and presents a simple programming interface to the user in the VR environment. This way, the user can connect buttons and sliders in the virtual laboratory to parameters of the simulation, and read out the data from the simulation.

As a second aspect in the acceptance of VR, *the user wishes to quantify his observations.* To enable a user to do so introduces several requirements for VIRPI. Primarily, the user needs to be able to specify the type of measurement in a rapid and easy way. Also, VIRPI needs to support ways to examine the dataset visually and specify subspaces on which the measurement procedure has effect.

To summarize, the following requirements define Aura and VIRPI:

- Aura hides platform differences, provides scene graph support, simple shapes, text, graph loading and access to VR hardware.
- VIRPI communicates to remote simulation programs by incorporating CAVEStudy.
- VIRPI presents an easy interface to define interactive behavior for measurements, object examination and the selection of subspaces.

4 Functionality

Using the requirements from the previous section, we will now describe functional aspects of the toolkit. Figure 1 shows an overview of our approach. The left side shows the VR environment, and the various layers that are running there. The right side shows the simulation environment, and the connection that is made with CAVEStudy. First, we will describe the Aura and VIRPI-layers in the VR environment. Then we describe how CAVEStudy combines both environments and facilitates the communication. Finally, we show how measurement paradigms can be described in this setup.

4.1 Low-level layer: Aura

Aura is the first layer as described in the previous section. It presents a lightweight interface to underlying low-level libraries. Without noticeable decrease of efficiency, it adds features that the underlying software lacks, or it simply passes calls in a transparent way. The core of the graphical part of Aura consists, like IRIS Performer, of nodes in a scene graph. These nodes can be geometry nodes, cameras, lights, etc. Next to this, Aura defines a variety of simple shapes (cube, ball, cylinder, arrow, pyramid, etc.), can load graph data (3D model files from modelers and other polygon descriptions), fonts and textures/images (.JPG, .RGB, .PNG, etc.).

Aura can encapsulate the CAVElibrary or VR Juggler, providing a simple interface to input devices and rendering contexts. On traditional workstations, a simulator (much like the CAVElibrary simulator) fills in the missing hardware.

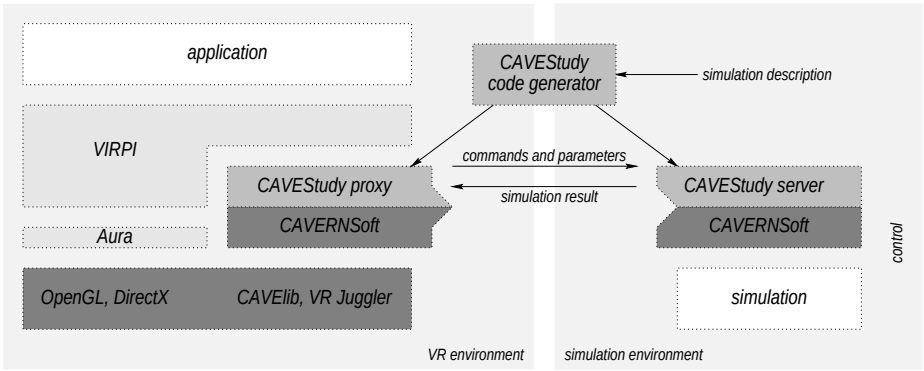


Fig. 1. The software layers for scientific visualization in VR.

In the current implementation, Aura comes as a set of libraries for various setups on different platforms. Aura is functionally the same for each setup. When the given setup is selected, simply creating the AuraVR environment object initializes the necessary hardware and tools.

To reduce complexity, Aura presents an interface where the connected hardware acts as a single environment. The interface does not present issues like multi-pipe output, multiprocessor systems and shared memory to the programmer. Instead, the Aura implementation deals with it when applicable.

4.2 High-level layer: VIRPI

Beyond the level of Aura, there are no more hardware-related issues, and platform-independence should be guaranteed. The high-level VIRPI toolkit rests on top of this level, and is identical for all platforms. VIRPI is roughly based on concepts and ideas from 2D GUIs like X or GTK. Where 2D GUIs use a hierarchical window-tree which can be used for event structuring, VIRPI uses a similar concept for hierarchical interface *views*.

A view is an atomic unit, which has a representation and can handle events. It is a tree node, so it can have a parent and one or more children. A node that contains children is called a *group*. The children of a group are called *sub-views*. Like with a scene graph, rotation, scaling and translation can be specified for each view with respect to its parent.

Like 2D GUIs, events are passed in two ways. First, events can be passed across the tree structure, originating at the root of the tree. These events are typically external events, like clicks of buttons on the pointer, movement of the pointer or other trackers, keystrokes, joystick changes, etc. Second, events can be passed directly from one view to the other, typically, to send messages between different control views.

When an event is received by a view, it is either passed down to its children, or processed. The user can overload event handlers for various types of events, adding functionality to the program. Figure 2 shows how the user can overload handling of the tracker motion event to have an object follow the user pointer (Wand, Stylus, mouse).

```
1  cMyApplication::OnTrackerMove(num,v,q)
2  {
3      if(num == TN_POINTER)
4          object->Translate(v);
5      else
6          cvApplication::OnTrackerMove(num,v,q);
7  }
```

Fig. 2. Code example showing how to overload a tracker move event.

The tracker motion event is defined as a tracker ID number, a vector indicating the position of the tracker and a quaternion indicating the orientation of the tracker. As soon as the tracker is moved, the `OnTrackerMove` method is invoked. When the tracker is the user pointer, the object is translated to the location of the tracker. When the tracker is something else, the inherited `OnTrackerMove` is called.

Completing the basic framework is an application base class. This is essentially a view, serving as root to the rest of the program. Writing a VIRPI program means subclassing the application class, and adding interactive features.

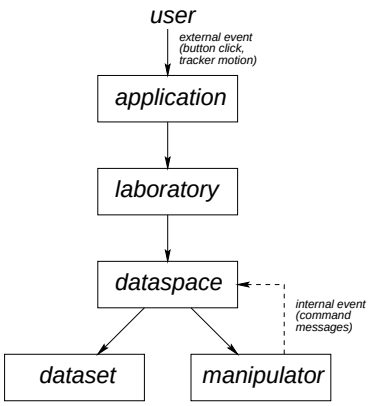


Fig. 3. An example event tree for a simple virtual experiment.

Figure 3 shows a view tree for a fictive application created with VIRPI. The application initializes a laboratory, which displays familiar laboratory surroundings and spatial references (for instance, a virtual room with a table). The laboratory creates a data space, which serves as a frame of reference in which data and measuring tools can be added. A dataset is added to the data space. Manipulating the data space will directly manipulate the dataset as well. To do this, a *manipulator* is added to the data space. The dashed line in figure 3 indicates command messages that are being sent by the manipulator to transform the data space.

Simple Controls Next to the basic structure, VIRPI provides several simple controls to interact with the user. Functional parts of 2D GUIs, like menus, sliders and (ra-

dio)buttons have a VR counterpart in VIRPI. These controls can be created, added and removed to the application at will. As the interface manifests itself in an immersive 3D environment, one can envision more new controls that are not directly related to their desktop paradigm. This, however, is beyond the scope of the paper.

```
1  // in the constructor:
2
3      :
4      button = new cvButton("arrow.jpg");
5      AddView(button);
6      :
7
8  // the message handler:
9  cMyApplication::OnMessage(sender,message,...)
10 {
11     cvApplication::OnMessage(sender,message,...);
12     if((message == MSG_PRESS) && (sender == button))
13     {
14         :
15     }
16 }
```

Fig. 4. Code example showing how to add a button control and handle its messages.

All of these controls send events to the parent groups, indicating changes or command messages. These messages can then be processed by overloading the appropriate methods. Figure 4 shows how a button control is added to a VIRPI application. In the application's constructor, the button is created and added to the application. Furthermore, the application's message handler is overloaded. First the inherited message handler is called to process different messages, then if the message is a press-message and originates from the button, the button action is processed.

Data Manipulation An important aspect of visually examining an object is the ability to zoom in on it, move it and rotate it. These tasks are generally done with manipulators. A manipulator is a construct that interprets events from the user to make an object move, rotate or scale in an intuitive way. Internally, it is a group containing one or more handle views with which the user can interact. A handle view is a (mostly small) object, serving as interaction point for the manipulator. By dragging on a handle view, the user can scale, translate or rotate the object to which the manipulator is attached. Depending on the scheme of handle views and the functionality they provide, a manipulator can have many functions. Several standard manipulators are provided in a similar fashion as for Open Inventor.

4.3 Steering of a Simulation

As noted, one of the requirements of VIRPI is that it allows the user to interact with a running simulation. To minimize programming for the control of the simulation and the data management, the user has to describe the simulation with an XML description

file. This file is processed by CAVEStudy to generate two objects, a *proxy* and a *server* (see Figure 1). The simulation is wrapped into a server object to control its execution. The server's interface provides methods to start, stop, pause and resume the simulation. The data generated by the simulation are automatically propagated to the proxy object. This object can be seen as a local copy of the remote simulation. Through the network, it reflects the input values and the commands to the server. Furthermore, it manages the incoming data from the simulation, and presents it to VIRPI.

CAVEStudy generates C++ code for the server and the proxy object, using the CAVERNsoft[5] network layer, as shown in figure 1. CAVERNsoft uses a "subscribe and publish" paradigm. A site can define keys to publish its own data, and a remote site that subscribes to these keys will automatically receive the data through call-back functions. This mechanism can be used for small data (tracker data), but also for large datasets (data-mining) using different policies.

The CAVEStudy code generator produces objects for the server and the proxy. Each of these objects contains a set of keys with their associated call-back functions to transmit input and output values. The marshaling code for all the types is generated to be able to use the system in a heterogeneous environment. A set of keys is also created for the control of the simulation (*initialize*, *start*, *stop*, *pause*, *resume*, *shutdown* methods). It is therefore possible to manipulate proxy and server entities as C++ objects, without dealing with network issues. For the server, the program is an endless loop, waiting for remote method invocations. The proxy object is embedded into VIRPI.

By using CAVERNsoft, it is possible to access one simulation with multiple VR setups. This way, a basic collaboration setup can be realized among multiple sites. Each site can, depending on their VR setup and the individual wishes of the users, display different representations of the data.

4.4 Measurement

Theoretically, measuring can be defined as a quantification of an observation in a given space. We see a virtual environment which displays the simulation results as space in which we can observe, and on which we can perform measurements. Some examples of measurements that can be done in VR are:

- In a gas simulation, dynamically count the number of atoms inside a given volume, and related to this give information about pressure and temperature.
- For a molecular dynamics simulation, measure the local electromagnetic field at a point, derived from potential data or the charge distribution from the simulation.
- From an MRI-scan of the human upper body, measure the volume and surface of a lung, or parts of a lung.

There are two key issues in making the toolkit useful for writing measurement applications as described. First, the user needs to be able to describe geometrical subsets of the data. He needs to select atoms, describe a volume, a surface, etc. Second, the user must specify the measurement itself, the calculations that need to be done on the data subset.

Basic subsets, like point sets, spherical volumes, box volumes or rectangular surfaces can be described in VIRPI by adding a selection view to the data (or rather, a data

space as described in section 4.2) in an identical fashion as adding a manipulator. A combination of more subsets can be defined by means of constructive solid geometry operations.

```
1  cMyTool::Update(data,selection)
2  {
3      count = 0;
4      for(all atoms)
5          if(selection contains(atom[i]))
6              count++;
7      SendMessage(display,MSG_UPDATE,count);
8  }
```

Fig. 5. Code example showing how to implement a measuring tool.

To describe a measurement procedure, the user can overload the `Update` method of the basic VIRPI measurement tool. This method gets called every time the dataset changes or the subset selection changes. In the `Update` method, the user has access to the elements of the dataset and the geometrical extents of the selected subset. Figure 5 shows the implementation of an example measuring tool that counts atoms in a given box volume. The `Update` method of the basic measuring tool class is overloaded, and using the selected volume, the atoms are counted. The result of this is transmitted to a display object.

5 Case Study

This section presents a case study implemented using Aura and VIRPI. The case study is representative for a range of applications in medical imaging. Furthermore, the application allows to compare results from virtual measurements with results from real-world measurements. Using this case study we describe how to setup Aura and VIRPI to examine data and quantify observations from it.

In order to clean the root canal of a patient's molar effectively, it is essential to know its exact length. Traditionally, measuring this length is done by sticking various sized files into the molar, guided by one or more X-ray photographs in different orientations. Figure 6 shows an alternative to this. By using a local CT scanning technique, the patient's molar is scanned, and a volume of 16-bit grey-scale voxels is calculated from the resulting images. An external program can produce isosurfaces for different threshold values from the volume, using VTK's Marching Cubes implementation. The isosurface is visualized by the VIRPI application, and the user measures the length of the root canal. This application was considered because it shows how the non-VR expert dental researcher is able to use VIRPI. Also, it lets the user measure on a visualized reconstruction in VR.

Figure 7 shows the setup in a VR environment for the application and the corresponding view tree. The root application group creates a data space and adds the tooth data to it. To measure the length of the inherently curved root canal, the application adds a yardstick to the data space. This yardstick is an application-specific view containing

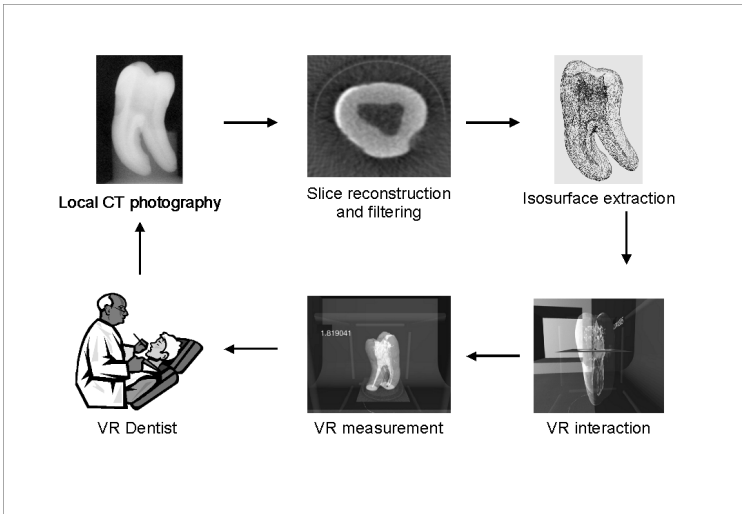
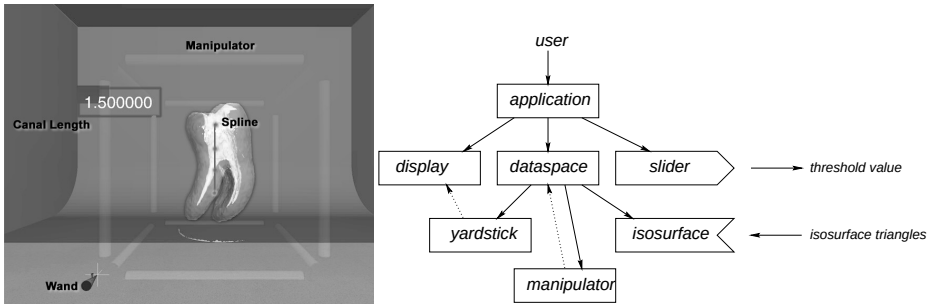


Fig. 6. The process steps from tooth to VR application.

a controlled Catmull-Rom spline representation and four interactive control points. The control points are VIRPI data subset selectors, and messages from the control points are processed by updating the spline coefficients of the yardstick.



(a) setup (b) view tree

Fig. 7. Setup and corresponding view tree for the dental application.

Interaction with the simulation is implemented by adding a slider to the application. The slider's messages are passed to the CAVEStudy proxy object, which indicates changes in the threshold value to the marching cubes program.

To display the length, a display is added to the application. The display receives messages from the yardstick's `Update` method every time it changes shape (indicated

by the dotted line in figure 7(b)). Finally, a manipulator is added to the data space (indicated by the cylinder cube in figure 7(a)), so the user can rotate the tooth and the yardstick, in order to view the experiment from a better angle. Dragging the edge cylinders of the cube makes the tooth rotate around the parallel axes.

6 Conclusions

The paper shows that a high-level toolkit, available on various setups, allows non-VR expert users to consider VR for their work. It also shows, by means of a case study, that it is possible to use VR as an experimentation environment where real-world measuring paradigms can be applied. Furthermore, it shows that it is feasible to implement such a toolkit for various platforms. Issues that are addressed regarding the differences in underlying operating systems and libraries present no unovercomable drawbacks. However, a lot of testing still needs to be done, as the project grows.

Towards a more complete environment which is considered useful by non-VR experts, some aspects still require attention. Currently, visualization of data in VIRPI is application-specific and limited to simple user-defined representations. An interface with data-flow visualization packages, like VTK or IBM's Data Explorer, should enable more genereral descriptions of the representations. Also, one could envision an interactive description of the representation, where the user can directly adjust the visualization data-flow to specific needs.

Next to this, although basic collaborative capabilities using CAVEStudy are mentioned, more work on this is desirable. We plan to research and implement more tight collaboration schemes in Aura/VIRPI that enable higher level telepresence aspects for measurement environments. Scientists can greatly benefit from the availability of a collaborative examining environment, in which they can discuss findings with peers around the world.

References

1. Christer Carlsson and Olof Hagsand. DIVE — A Platform for Multi-User Virtual Environments. *Computers and Graphics*, 17(6):663–669, November–December 1993.
2. I. Curington and M. Coutant. AVS: A flexible interactive distributed environment for scientific visualization applications. *Proceedings of 2nd Eurographics Workshop on Scientific Visualization*, 1991.
3. Chris Greenhalgh and Steven Benford. MASSIVE: A collaborative virtual environment for teleconferencing. *ACM Transactions on Computer-Human Interaction*, 2(3):239–261, 1995.
4. Christopher Just, Allen Bierbaum, Albert Baker, , and Carolina Cruz-Neira. VR Juggler: A Framework for Virtual Reality Development. In *2nd Immersive Projection Technology Workshop (IPT98)*, 1998.
5. J. Leigh, A.E. Johnson, T.A. DeFanti, and M. Brown. A Review of Tele-Immersive Applications in the CAVE Research Network. In *IEEE Virtual Reality'99*, pages 180–187, 1999.
6. Jason Leigh. Issues in the Design of a Flexible Distributed Architecture for Supporting Persistence and Interoperability in Collaborative Virtual Environments. In ACM, editor, *SC'97: High Performance Networking and Computing, San Jose (CA) USA.*, 1997.

7. M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz. NPSNET: A Network Software Architecture for Large-Scale Virtual Environment. *Presence*, 3(4):265–287, 1994.
8. B. Plale, G. Eisenhauer, K. Schwan, J. Heiner, V. Martin, and J. Vetter. From Interactive Applications to Distributed Laboratories. *IEEE Concurrency*, pages 78–90, April-June 1998.
9. Luc Renambot, Henri E. Bal, Desmond Germans, and Hans J.W. Spoelder. Cavestudy: an infrastructure for computational steering in virtual reality environments. In *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing*, pages 57–61, Pittsburgh, PA, August 2000. IEEE Computer Society Press.
10. R.Hubbold, J.Cook, M.Keates, S.Gibson, T.Howard, A.Murta, A.West, and S.Pettifer. GNU/MAVERIK: A micro-kernel for large-scale virtual environments. In *VRST'99, ACM Symposium on Virtual Reality Software and Technology*, 1999.
11. John Rohlf and James Helman. IRIS performer: A high performance multiprocessing toolkit for real–Time 3D graphics. In Andrew Glassner, editor, *SIGGRAPH '94*, Computer Graphics Proceedings, Annual Conference Series, pages 381–395. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.
12. W. J. Schroeder, K. M. Martin, and W. E. Lorensen. The design and implementation of an object-oriented toolkit for 3D graphics and visualization. In R. Yagel and G. M. Nielson, editors, *Proceedings. Visualization '96. San Francisco, CA, USA. 27 October–1 November 1996*, pages 516–??, 93–100, 472, New York, NY 10036, USA, 1996. ACM Press.
13. G. Singh, L. Serra, W. Png, and Hern Ng. BrickNet: A Software Toolkit for Network-Based Virtual Worlds. *Presence*, 3(1):19–34, 1994.
14. Sandeep Singhal and Michael Zyda. *Networked Virtual Environments: Design and Implementation*. Addison-Wesley, 1999.
15. Hans J.W. Spoelder. Virtual Instrumentation and Virtual Environments. *IEEE Instrumentation and Measurement Magazine*, 3(3):14–19, 1998.
16. Henrik Tramberend. Avocado : A Distributed Virtual Reality Framework. In *IEEE Virtual Reality'99*, pages 14–21, 1999.
17. Andries van Dam, Andrew S. Forsberg, David H. Laidlaw, Joseph J. LaViola, Jr., and Rosemary M. Simpson. Immersive VR for scientific visualization: A progress report. *IEEE Computer Graphics and Applications*, 20(6):26–52, November/December 2000.
18. J. Wernecke and I. Mentor. *Programming Object-Oriented 3D Graphics with OpenInventor*. Addison-Wesley, 1994.