ICAT 2005
December 5-8, Christchurch, New Zealand

A Generic Virtual Reality Software System's Architecture and Application

Frank Steinicke, Timo Ropinski, Klaus Hinrichs
Institut für Informatik, WWU Münster, Einsteinstraße 62, 48149 Münster, Germany
{*fsteini,ropinski,khh*}*@math.uni-muenster.de*

Abstract

Virtual reality (VR) systems utilize additional input and output channels in order to make interaction in virtual environments (VEs) more intuitive and to increase the user's immersion into the virtual world. When developing VR applications, developers should be able to focus on modeling advanced interaction and system behavior instead of rendering issues. Many systems and tools for developing virtual reality applications have been proposed to achieve this goal. However, no de facto standard is available. In this paper we present *Virtual Reality VRS* (*VR*^{2}*S*), a generic VR software system, which is an extension of the high-level rendering system VRS. The system provides flexibility in terms of the rendering system and the user interface toolkit. Thus, with using VR^{2}S rendering can be performed with several low-level rendering APIs such as OpenGL, RenderMan or ray-tracing systems, and the interface can be implemented by arbitrary user interface toolkits to support both desktop- and VR-based interaction. The proposed system meets the demands of VR developers as well as users and has demonstrated its potential in different planning and exploration applications.

Keywords: Virtual Reality, Software Architecture, VR Interaction Techniques, VR Applications

1. Introduction

According to Burdea and Coiffet ([5]) virtual reality (VR) is a high-end human-computer interface that involves real-time simulations and interactions through multiple sensorial channels. These systems serve one main purpose, namely the enhancement of human-computer interaction (HCI). But virtual reality is not limited to advanced user interfaces; VR applications help to solve real problems since immersive virtual environments (VEs) improve perception and provide better insights into complex datasets. Examples are complex design and planning tasks, and the exploration of large volumetric datasets.

In these VR systems the main focus is on interaction, which combines adequate representations of the virtual environment with the manipulations available in the VE. Due to the availability of special hardware, in particular intuitive and natural VR input devices, applications in this context are highly-interactive and enable miscellaneous interaction concepts. Thus, another important aspect of VR systems is real-time performance. Response time and update rate of the system need to be sufficient to avoid latencies.

Within dynamic and responsive computer generated VEs, multimodal interactions are typically realized by using special VR hardware. These devices such as data gloves, wands and speech recognition systems etc. enable users to navigate in and to interact with the virtual world more immersively and intuitively since multiple senses are addressed during the interaction process. In addition, immersive display technologies, e.g., stereoscopic projection systems and head mounted displays, in combination with various multisensory output paradigms like auditory and haptic feedback improves the immersion into the virtual world.

When designing VR applications developers should have to focus less on rendering and implementing devices' interfaces but rather on interaction and simulation in order to make VR worlds to become more interactive and responsive. For that purpose, *VR software systems* and *toolkits* facilitate an abstract view of the system and the input and output devices by providing higher-level development libraries to reduce the effort needed for creating and rendering VEs. Since low-level APIs such as OpenGL and DirectX address basic rendering issues, VR research can now focus on more advanced topics, e.g., to further extend multisensory interfaces and to advance basic interaction techniques by providing intuitive metaphors to enhance interaction in VR applications. In this paper we present such a VR software system.

The paper is structured as follows. The both next sections discuss related work and point out the contributions of our approach. In Section 2 we introduce *Virtual Reality VRS* (*VR*^{2}*S*), a VR-based extension of a generic graphics software system, which is currently under development in our VR laboratory. In Section 3 we discuss examples of interaction concepts we have integrated in VR^{2}S. Section 4 presents some example applications based on VR^{2}S and points out its benefits and potential. In the last section we conclude the paper and give an overview about future work.

1.1. Related Work

Many VR software systems and VR toolkits have been proposed which support the development of virtual reality applications. For example, users of VPLs Body Electric ([1]), which is a real-time simulation system for VR, specify relations between virtual objects and input or output devices in a dataflow diagram editor. This dataflow approach can also be found in a variety of similar systems such as SGIs Open Inventor and VRML. However, a pure dataflow approach does not support program modularity ([3]).

Alice ([11]) is a rapid prototyping system for creating interactive computer graphics applications without requiring much technical background. Since Alice has not been designed to allow complete control over geometry at polygon and vertex level for handling large data amounts, the library is unsuitable for scientific and complex data visualization.

CAVElib ([6]) which is based on both OpenGL as well as

IRIS Performer provides low-level APIs for creating applications for VR systems, in particular projection-based systems. Bierbaum and Just ([2]) mentioned that the CAVE library was not intended as a long-term solution for VR development, and thus its API is often difficult to extend in backwards-compatible ways. In addition, the library is inadequate for non projection-based VR systems, e.g., augmented reality (AR) systems.

AVANGO ([19]) extends IRIS Performer scenegraph objects to allow multisensory VR application development by providing fast and flexible VR support. Since AVANGO is based on IRIS Performer, it is limited to SGI platforms and not usable in other standard system environments. The modular system Lightning ([4]) is an object-oriented system for creating and developing VR applications. Similar to AVANGO Lightning is also restricted to SGI platforms.

VRJuggler ([3]) and DIVERSE ([9]) are software systems for building large high-end VR applications. Both systems have been designed to overcome the drawbacks of some of the previous systems. VRJuggler establishes a single common, modular architecture for different devices. To maximize performance the only layer the application interfaces to directly is the graphics API. Due to the complexity of VRJuggler the usage of its API is not effortless, whereas DIVERSE lacks platform independence since it processes on Linux and IRIS environments only.

Other approaches such as the Studierstube project ([13]) and MRToolkit ([14]) have originally been developed for augmented reality, but extend to other VR systems. However, these approaches provide insufficient support for projection-based VR environments.

A more detailed description of further VR software systems and toolkits can be found in [2]. Although, many systems are available for creating and developing virtual reality applications, due to compatibility and customization issues universities and research institutions tend to use their own visualization libraries and VR-based extensions to explore and interact with their specific datasets. Thus, there is no standardization of VR system architectures, yet.

1.2. Contributions

In this section the contributions of VR^{2}S as an alternative approach of a generic VR software system are pointed out. Some of the following features of VR^{2}S are standard in most VR toolkits and therefore not explained in detail. We briefly discuss our VR software system under different viewpoints, including ease of VR application development and usability of the library as well as technical properties, such as extensibility, modularity etc.

Demands on VR software systems include especially hardware-oriented issues such as support for multisensory output, various input paradigms, or device independence. Also multi-user support for collaborative interaction should be possible within the VR system. VR^{2}S meets these requirements and is used in different VR system configurations. In addition, it is possible to run VR applications in desktop environments by simulating VR devices with standard desktop input paradigms. Thus, expensive system components can be conserved by using desktop environments whenever possible.

Since VR^{2}S is based on the generic 3D graphics system VRS, the *Virtual Rendering System* ([8]), which has been designed to enable rapid development of interactive graphics applications, developers can focus on interaction and system behavior. VR^{2}S provides an adequate abstraction for VEs as well as interplay of highly independent modules by a multi-layered application programming interface. The platform independent and modular implementation of VR^{2}S eases both extensions as well as usability and aids portability. Since VRS supports standard formats exchange of content is ensured.

In contrast to most other graphics systems, VRS provides a rendering system independent API. Thus, a VR^{2}S-based application can switch between different rendering systems at runtime; there is no need for redesign of the application source code.

In addition a generic user interface concept supports the development of applications, which are not constrained to a specific user interface. Hence, VR applications can be ported to many existing standard user interface.

The proposed generic VR software system VR^{2}S meets the demands of VR developers and users and has demonstrated its potential in different planning and exploration applications.

2. Virtual Reality Extension of the Virtual Rendering System

In this section we describe the architecture and concepts as well as some components of VR^{2}S. Since VR^{2}S is based on the generic graphics software system VRS, the Virtual Rendering System will be introduced first.

2.1. Virtual Rendering System

VRS ([8]) is an object-oriented, scenegraph-based 3D computer graphics system written in C++, which provides numerous building blocks for composing 3D scenes, animated 3D objects, and 3D interaction. VRS concentrates on OpenGL for real-time rendering and supports advanced rendering techniques such as multipass algorithms for global illumination. In addition, VRS wraps and supports various other 3D rendering libraries, e.g., the RenderMan, the global illumination system Radiance and also raytracing renderers such as POVRay. Since VRS completely encapsulates low-level 3D rendering systems through a uniform interface, applications can switch between different rendering systems at runtime without any need for reimplementing the application. Low-level code specific to a rendering system can be integrated into VRS source code; but this low-level code causes loss of flexibility of the rendering system. In interior design prototyping, for instance, a modeled scene can be rendered for interactive exploration and sound propagation using OpenGL and an arbitrary spatial sound library. Rendering the scene with Radiance can simulate interior light dispersion, while an additional photorealistic impression of this room can be achieved using the POVRay interface. Applications based on other graphics systems need a reimplementation to switch to another rendering system.

VRS is available for most common platforms such as Windows, Linux, Unix, and Mac OS. All major user interface systems such as Qt, X11, wxWidgets, Tcl/Tk , and GTK+ are incorporated.

VRS is designed to support rapid development of interactive 3D applications. Furthermore, it also serves as a framework and testbed for application-specific and experimental rendering, and design of innovative interaction techniques.

For run-time debugging, developers can use the Tcl/Tk package *interactive VRS* (*iVRS*) ([10]), which is intended

as a rapid prototyping graphics environment. iVRS provides access to almost all features of VRS through the scripting language without the need of compiling the source code of the application.

2.1.1. Scene and Behavior Graphs in VRS

A VRS application requires at least one canvas, which is typically integrated into the applications user interface. The scene displayed in a canvas is defined by one or more scenegraphs, while interaction and animation is specified by behavior graphs. Thus, the four essential components a VRS developer has to deal with are:

- a **canvas** representing a drawing area for VEs,
- hierarchical **scenegraphs** describing geometry and appearance of 3D virtual objects,
- **behavior graphs** describing event-dependent and time-dependent behavior of the virtual scene, and
- **graphics** resp. **non-graphics objects** representing shapes, graphics and non-graphics attributes. These objects are evaluated when included in scenegraphs. Animations and interactive manipulations can be performed by specifying them in corresponding behavior graphs.

Modeling VEs with two types of graphs, scenegraphs and behavior graphs, is unique to VRS. Because behavior is independent from geometry and most behavior cannot be assigned to a single subgraph, both aspects of an application should be treated independently. Canvas objects delegate events and requests, which are sent from the VRS run-time management, to the associated graphs, e.g., redraw events to scenegraphs, and mouse or keyboard events to behavior graphs.

2.2. Virtual Reality Extension of VRS

VRS provides flexible and rapid prototyping of 3D applications. The distinction between geometry or appearance of VEs and their behavior, eases the development of interactive and immersive applications.

In this section we describe the extension of the VRS graphics system to obtain an alternative approach for a generic VR software system, which exploits the benefits of the Virtual Rendering System and contributes new concepts for HCI.

2.2.1. Software Architecture Overview

The main objective of VR^{2}S is to provide VR software developers with a suite of APIs that abstract, and hence simplify all interface aspects of applications including the user interface and typical VR system tasks. The integration of VR software components into VRS is based on the same paradigms as the entire design of VRS. VR^{2}S applications are essentially independent of the VR system, and hence applications run on different system architectures in both VR systems and desktop environments. The modular and flexible design of VR^{2}S permits individual as well as combined usage of VR components. Thus, existing desktop application can easily be extended by VR specific components, e.g., viewpoint-dependent stereoscopic rendering techniques (see Section 2.2.4) require just a few additional lines of code:

```
1 // initialization of the tracking system
2 TrackingSystem* ts;
3 ts = new TrackingSystem(camera);
4
5 // set ID of the tracked glasses
6 ts->setHeadID(0);
7
8 // append tracking system to the canvas
9 canvas->append(ts);
```

In line 2 and 3 a tracking system object required to determine the user's head position is created dynamically. The constructor's only parameter is the camera attribute, which is modified when viewpoint dependent rendering is active. Other optional parameters include, for instance, the measures and orientation of the projection-screen or additional transformations applied to the tracking data. In line 6 the application developer assigns the (default) ID for head tracking, i.e., pose of rigid body number 0 is used for the position and orientation of the virtual camera. Finally, in line 9 the tracking system object is connected to the canvas and all tracking events are evaluated by this canvas.

As illustrated in Figure 1, VR^{2}S extends VRS by a VR-based user interfaces to support multimodal HCI in applications. This VR-based user interface (see Section 2.2.3) provides advanced concepts for the development of interaction metaphors and techniques. All metaphors integrated into VR^{2}S can be used with both VR hardware devices and desktop-based input paradigms such as two-dimensional GUIs.

Since there are many libraries for multimodal input and output channels, the usage of multisensory interaction is performable by extensible interfaces to the corresponding libraries. Some of these extern libraries are not constrained to support multisensory input or output channels and include further advanced interaction concepts, such as physical simulation systems, e.g., *Open Dynamics Engine* (*ODE*), or the AR software system *ARToolKit*.

In the graphics layer, some of the VR^{2}S extensions refer to the VRS core of rendering techniques, e.g., off-axis stereoscopic rendering, other components relate more to the VR-based user interface at the application layer. Rendering of the scene is performed in the rendering layer by usage of adapters to the corresponding rendering system.

In the following subsections we will explain some of these basic concepts and interfaces in more detail. All interaction issues are handled via the `Interaction` class, which evaluates all kind of input events and eventually activates appropriate system behavior.

2.2.2. Interaction and Event Handling

An interaction between a user and the VR system consists of three logical steps:

1. It **starts**,
2. it **executes** for a certain time interval, and
3. it either **ends** regularly or is **canceled**.

Accordingly, the specific interaction tasks can be implemented in four methods `start`, `execute`, `end`, and `cancel` by subclassing these methods. Conditions that have to be satisfied determine the state of the interaction.

For example, considering 6 degrees of freedom (DoF) manipulations, the interaction is started, when the user performs a predefined action indicating the selection of a virtual object. During the 6 DoF manipulations the interaction executes until the user releases the object or an error

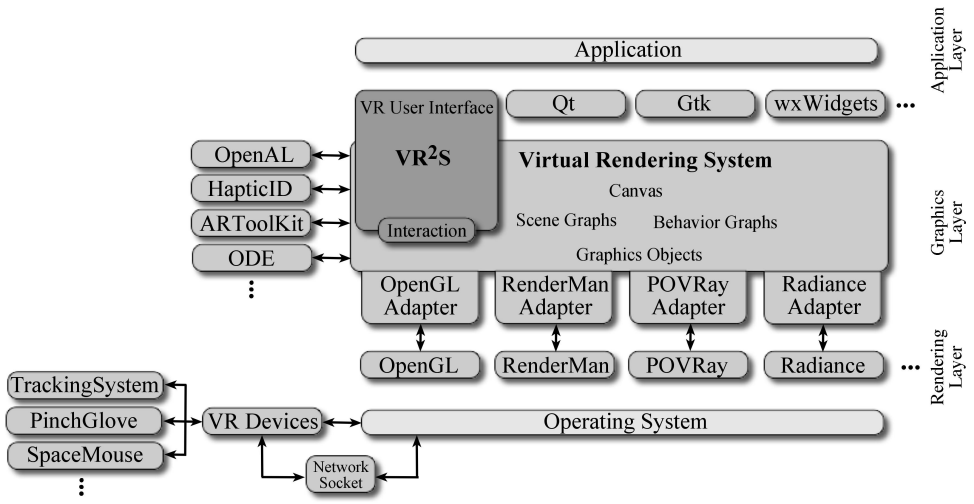

Figure 1: The system architecture of VRS ([8]) and its VR-based extension VR^{2}S consisting of application, graphics and rendering layers.

occurs. An interaction can receive and propagate canvas events, which belong to the interaction nodes appended to one of the behavior graphs of the corresponding canvas; other events are ignored.

According to the conditions the `Interaction` class triggers the `start`, `execute`, `end`, and `cancel` methods. Examples of subclasses of the `Interaction` class are control classes for input devices, which receive and propagate input device events, e.g., glove or space mouse events.

2.2.3. VR User Interface

VR system environments support multisensory inputs that enable a natural and intuitive interface for HCI. To use such devices a uniform class `SpatialInputDevice` provides an interface for controlling virtual input devices. A virtual input device is the representation of a physical device in the virtual world. It is controlled via sampled data received from the hardware and abstracts from the particular used input device. Special VR hardware devices are either connected to the same computer VR^{2}S runs on or they can be communicate via network sockets with VR^{2}S. Thus, to maintain performance extensive and also platform dependent processes such as tracking or gesture recognition can be distributed to different systems.

When an arbitrary VR input device, e.g., a wand or a data glove, is used, corresponding `SpatialInputDeviceEvents` are generated and processed by the interaction handling mechanism.

The events for a virtual input device are divided into four groups:

1. **Motion events** are generated and processed when an associated input device is tracked and moved through the VR system environment.

2. **Selection events** occur when the user performs a predefined action associated with the selection process, e.g., pressing a special button or posing a special gesture.

3. After the user has indicated a selection process, **move events** are generated permitting the 6 DoF manipulations of virtual objects.

4. Analogous to the selection subprocess, **release events** are generated by the user performing predefined release actions, e.g., releasing a button.

The resulting events cause corresponding system behavior, i.e., if a motion or move event occurs, the virtual input device resp. selected objects are positioned in the scene according to the pose of the real input device. A selection event causes grabbing, a release event unhanding of virtual objects. This interaction is implemented by a callback mechanism, i.e., events 1 to 4 execute user-definable callbacks in which all manipulations are handled.

The following source code describes how an arbitrary scenegraph element can be exploited as a virtual input device.

```
1  SceneThing* thing = new SceneThing(view);
2  thing->append(new Sphere());
3
4  // append spatial input device to the canvas
5  canvas->append(new SpatialInputDevice(thing));
```

In line 2 a virtual sphere is appended to a scene thing, which is contained in the scenegraph. In line 5 this sphere is used as virtual input device and can be controlled by arbitrary physical input devices, such as tracked data gloves.

Since the `SpatialInputDevice` class incorporates further information, several interaction metaphors can be implemented by inheriting these callbacks (see Section 3).

2.2.4. Multimodal User Input

VR input devices not already supported by VR^{2}S can easily be integrated into VR^{2}S to enable control via the described VR user interface. Besides desktop devices such as mouse or keyboard VR^{2}S supports data gloves, space mice and arbitrary tracked input devices. In addition, any physical object can functionalize as VR input device by applying appropriate tracking technologies. Thus, user-defined input devices such as wands, gloves or hands are trackable for natural and intuitive 6 DoF interactions.

The usage of tracking systems provides another powerful extension of VR systems. Tracking of the users' head position and head orientation enables the system to calculate a corresponding camera projection and orientation. Thus, users can walk around virtual objects and watch them from different positions. The evaluation of tracking data generated by an arbitrary tracking system is implemented by a general interface class `TrackingEvent`. The recorded tracking data generates events, which includes information about the uniquely tracked object, such as the transformation of this object in relation to its initial position and orientation, the moment of the tracking, and the canvas to which the tracking system is associated with. In the case of head tracking a trackable configuration is attached to the user's head, e.g., a rigid body attached to stereo glasses.

Since, VR hardware devices are expensive technologies, all devices can be simulated using VR^{2}S, which allows the implementation and testing of VR interaction concepts outside a VR system environment. For example, head tracking events can be emulated using the arrow keys on the keyboard.

2.2.5. Multisensory System Output

Even though visual perception is dominant in VEs, other sensors, e.g., auditory, haptic etc. provide the user additional immersive experience. Hence, like the multimodal user input channel, the system feedback may use multisensory concepts.

Similar to the other event classes, multisensory feedback can be initiated via corresponding events propagated, for example, by the `HapticEvent` and the `SoundEvent` classes, to enable haptic and auditory feedback. These events are generated by interface classes to the corresponding libraries such as the spatial sound library OpenAL or the haptic library (HapticID) developed at our VR laboratory. This library can be used to create and trigger user-definable vibration signals for hardware devices equipped with a vibration unit, e.g., wands or gloves. Interfaces to other multisensory libraries can be integrated easily with our approach.

3. Interaction Metaphors and Advanced Exploration

In this section we briefly present an overview of interaction metaphors, which have been developed and implemented with VR^{2}S.

3.1. Interaction Metaphors

The enhancement of human-computer interaction in VR requires the availability of intuitive and natural interaction metaphors to accomplish fundamental interaction tasks easily. Thus, we have developed several multimodal interaction metaphors, which underline the feasibility of the described interaction concepts.

3.1.1. Improved Virtual Pointer Metaphor

In order to advance the selection and manipulation process when using pointer techniques, we have introduced the improved virtual pointer (IVP) metaphor ([16]), which avoids most of the disadvantages of current interaction metaphors. The IVP metaphor enables a user to select a desired object without requiring an exact hit. A straight ray is used to indicate the direction of the virtual pointer, while an additionally visualized bendable ray points to the closest selectable object. The selection process is accelerated since a user can roughly point at desired objects, and the curve automatically gives a visual feedback about the object's exact position. Besides the advanced interaction with distant and small objects, the usage of a bendable ray also enables the selection of partially or completely occluded objects.

To further support the user during the interaction we add multimodal concepts described in [17] to this selection process. When the curve bends to another object the user perceives a smooth vibration signal, whereas the signal strength increases with decreasing distance between user and object. In addition, a gentle sound disperses from the position of that object and results in a better spatial cognition of the location of the desired object. Thus, also non-visible objects can be located approximately.

After selecting the active object, 6 DoF manipulations can be accomplished by different mapping approaches between movements of the input device and the manipulated object as described in [17].

The usage of the IVP metaphor is illustrated in Figure 2 and Figure 4. In Figure 2 a user interacts with a virtual building via a 3D widget ([7]) attached to it. The small handles enabling a constrained rotation and translation of this object are not accessible with common VR interaction techniques, whereas the IVP metaphor supports the 3D widget interaction. In Figure 4 the IVP metaphor concepts are used for menu-based interaction.

3.1.2. Dual-Purpose Interaction Metaphor

In [18] we introduced a dual-purpose metaphor which is an extension to the IVP metaphor concepts.

The objective of this approach is to simplify porting of desktop applications to VR systems. The main concept is based on enabling desktop interaction with VR hardware devices by mapping the tracked information to corresponding desktop events. In opposite to the VR simulation mode, VR hardware emulates standard desktop input devices. Thus, all desktop-based interaction aspects are applicable with the usage of VR devices. To overcome inaccuracy and latency of tracking results, we integrate the IVP metaphor concepts into this approach to advance two-dimensional GUI interactions. Switching between both modes, i.e., VR-based and desktop-based interaction, is ensured at runtime.

Figurer 4 illustrates the IVP metaphor in a medical application allowing an easy access to two-dimensional menu items such as checkboxes, buttons and sliders. The visual representation of the medical data can be explored and manipulated with VR-based interaction techniques.

A user study of the described concepts has pointed out the advanced usability and efficiency in comparison to other interaction metaphors. The participants have evaluated the described concepts as the most intuitive, easy to learn and easy to use approaches among the compared techniques ([17]).

3.2. Advanced Exploration Approaches

Traditionally, VR systems are used for advanced exploration tasks since stereoscopic viewing on large projection displays provides a better spatial perception of complex 3D datasets and also enables groups of users to explore and

experience VR together. In this section we will present two exploration metaphors which improve the spatial cognition in such a system setup.

3.2.1. Mixed Reality Environment

Seamlessly merging the real and the virtual world created within a computer is a challenging topic in current VR research. To optimize the user's immersion into a mixed reality (MR) environment, we have proposed the concepts of virtual reflections and virtual shadows ([15]), which use information about the real environment surrounding the user. The data is received via real-time cameras to generate corresponding reflections and shadows on virtual objects. For example, users can see their own reflections on reflective surfaces. Tracked input devices are used to determine the position and orientation of virtual light sources and they cast shadows on virtual objects according to their pose respectively.

To evaluate these concepts we have implemented an example application that supports the exploration of different car models. Virtual cars can be illuminated intuitively by positioning real lamps, to which passive markers are attached for optical tracking. The surrounding of the VR laboratory is captured with a USB camera mounted on top of the responsive workbench (see Figure 3 (a)).

Hence, the surface structures of cars look more realistic, and real and virtual worlds are merged. The virtual car seen from the users point of view is illustrated in Figure 3 (b), which shows clearly visible reflections and shadows on the engine hood.

3.2.2. Collaborative Exploration

Besides an enhanced visualization of complex data, large projection-based displays enable groups of users to explore and experience VR together. Such cooperations have been proven to be advantageous since in many application domains the bundling of experts knowledge increases the productivity. Since, generally these systems can project a perspectively-correct image for only one user, i.e., the user wearing tracked stereo glasses, we have developed an alternative approach for a collaborative virtual environment (CVE).

The main idea is based on sharing display capabilities after a registration process; the view space is divided into sub viewports in which every collaborator perceives a correct stereoscopic image. The registration process is handled via gestures and communication between the participants of a collaborative interaction. After a collaborator has announced for collaboration by performing a special gesture, this user is registered for the interaction process and appropriate viewport space is assigned to him. Alternatively, the interaction can be performed sequentially. In this case a successful registration process causes a switch of the active user, i.e., the user whose head and input devices are tracked for viewpoint dependent interaction.

Figure 2 shows two collaborators in a responsive workbench environment. The active collaborator manipulates a virtual building, while the second one is watching the interaction. By announcing via the registration process and a following acceptance of the currently active user, both users can switch their roles, or they can collaborate in front of a shared projection screen.

Figure 2: Two cooperative city planners interact with virtual buildings via a 3D widget on a horizontally mounted workbench arrangement.

4. Application Overview

In this section we will briefly describe two applications from different domains developed with VR^{2}S and point out the potential of the proposed VR software system. Both applications run in desktop and VR environments.

4.1. 3D Residential City Planner

Virtual 3D city models provide an important tool for analyzing and communicating spatial information associated with urban environments. They support many visualization domains, e.g., of city planning and scientific simulations. However, specific interactions performed by city planners, e.g., manipulation of virtual buildings and building areas etc., require further advancements of these basic interaction techniques. For this purpose, we develop a residential city planning software for urban planning tasks in cooperation with the department of city planning and building development as well as the cadastral department of the city of Münster (Germany). Using this software city planners are able to create and modify development plans by interacting with an automatically generated 3D representation of arbitrary districts or entire cities.

Desktop-based interaction within this software is performable via standard GUI interaction concepts. A menu supports standard tasks such as creation, selection, manipulation and removal of virtual buildings. Using the dual-purpose IVP metaphor (see Section 3.1.2) the port of this software to a VR system requires no adaptation. The user can easily switch between VR interaction such as manipulation of virtual building with the hands and the described GUI interaction. Hence, all interactions can be performed while access to the complete menu originally designed for the desktop application is ensured.

Figure 2 shows this application in a responsive workbench environment. The active user manipulates a virtual building by selecting handles attached to a 3D manipulation widget ([7]). This selection process is realized by using the IVP metaphor, which permits the access to the small handles of the widget.

(a) User in a responsive workbench environment with tangible user input elements.

(b) The resulting virtual reflections and virtual shadows from the users point of view.

Figure 3: Application of virtual reflections and virtual shadows to a car model in a mixed reality environment setup.

4.2. Molecular Cardiovascular Imaging

Visualization of medical volume datasets provides an essential diagnosis tool for medical applications. Today, medical volume rendering techniques ease health professionals work during medical diagnosis. Due to the rapid development of high precision medical imaging techniques, e.g., CT, MRI and PET, and their high spatial resolution, the amount and complexity of data makes performing diagnostic examinations a challenging task. Since in most medical applications the region of interest, e.g., tumors or arterial structures, is small in comparison to the overall dataset, mechanisms are required which support health professionals to focus on these regions.

Since VR technologies can be used to obtain immersive insights into such highly complex volume datasets, we have integrated different rendering techniques into VR^{2}S to highlight these regions of interest ([12]). The proposed algorithms enable to interactively define regions of interest within a volume dataset, to which a different visual appearance is applied. The region of interest is given by an arbitrary convex lens shape and a visual appearance associated with it. Data within the lens volume can be emphasized using arbitrary rendering techniques, e.g., edge-enhancement, isosurface rendering etc. In contrast to recent comparable techniques, the proposed algorithm is fully accelerated by todays commodity graphics hardware, such that the shape of the lens, its position as well as the visual appearance can be controlled interactively with VR hardware devices.

At present, these concepts are integrated in a software application for the exploration of medical volume data in desktop and VR environments. The software application is based on VR^{2}S to enable an easy development of interaction concepts. The engineering is focused on the design of intuitive and natural exploration metaphors to support an immersive perception of the inner structures of organisms. Another research issue is the development of further efficient interaction techniques permitting surgery specific tasks such as 3D segmentation.

Figure 4 shows the usage of a prototype of the described application in a back-projection passive stereo system. The IVP metaphor enables an intuitive interaction with 2D menu contents and volumetric PET datasets. The 2D menu can also be projected on personal interaction panels held in the non-dominant hand to enable tactile feedback during menu interaction. A spherical shaped emphasizing lens is applied to a PET scan of a mouse acquired using a Quad-Hidac small animal PET scanner.

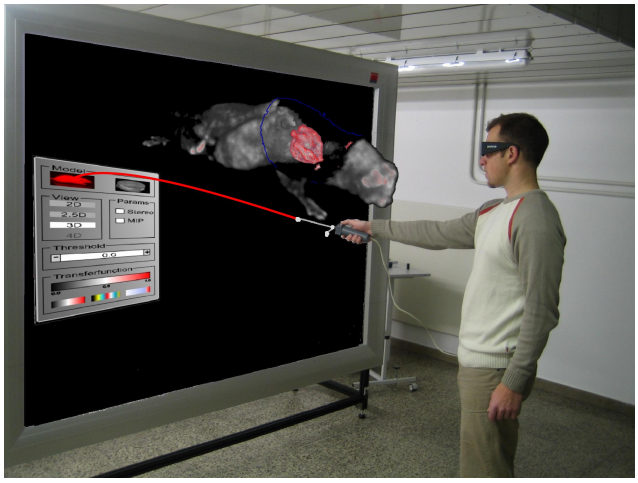

Figure 4: A user in a back-projection passive stereo system explores a volumetric PET dataset by applying a spherical shaped emphasizing lens and GUI interactions.

5. Conclusion and Future Directions

In this paper we have presented our approach of a generic VR software system. We have introduced the system architecture and described some basic components. The major benefits of VR^{2}S in comparison to other VR software sys-

tems and toolkits have been pointed out by discussing the integrated interaction concepts. Two example applications underline the potential and usability of the proposed extension for interactive domains focussed on interaction and exploration tasks.

In the future we will integrate additional input and output devices to further increase multimodality of VR applications. Additional interaction metaphors will be developed and evaluated in the context of the proposed VR software system. Further libraries, e.g., for physical simulation or speech recognition, will be integrated into the scenegraph-based structure to support the rapid development of constraint-based interaction.

6. Acknowledgments

We thank the students and graduands, who have implemented many parts of VR^{2}S and who evaluated the concepts in user studies. We appreciate the work of the VRS developers at the University of Münster and the Hasso-Plattner Institute in Potsdam, in particular Jürgen Döllner.

In addition, we would like to acknowledge the city planning and building development as well as the cadastral department of the city of Münster for their cooperation.

The development of the molecular cardiovascular imaging software is realized as part of the SFB 656 "*Molecular Cardiovascular Imaging*" which is funded by the german research foundation (DFG).

References

[1] Yoshitaka Adachi, Takahiro Kumano, and Kouichi Ogino. Intermediate representation for stiff virtual objects. In *Proceedings of IEEE VRAIS*, pages 195–203, 1995.

[2] Allen Bierbaum and Christopher Just. Software tools for virtual reality application development. In *Course Notes for SIGGRAPH 98 Course 14, Applied Virtual Reality*, 1998.

[3] Allen Bierbaum, Christopher Just, Patrick Hartling, Kevin Meinert, Albert Baker, and Carolina Cruz-Neira. VR Juggler: A virtual platform for virtual reality application development. In *IEEE Proceedings of VR2001*, pages 89–96, 2001.

[4] Roland Blach, Jürgen Landauer, Angela Rösch, and Andreas Simon. A flexible prototyping tool for 3D realtime user-interaction. In *Virtual Environments: Conference and Eurographics Workshop*, pages 195–203, 1999.

[5] Grigore C. Burdea and Philippe Coiffet. *Virtual Reality Technology.* Wiley-IEEE Press, 2003.

[6] Carolina Cruz-Neira. *Virtual Reality Based on Multiple Projection Screens: The CAVE and Its Applications to Computational Science and Engineering.* PhD thesis, University of Illinois at Chicago, 1995.

[7] Jürgen Döllner and Klaus Hinrichs. Interactive, animated 3D widgets. In *Computer Graphics International 1998*, pages 278–286, 1998.

[8] Jürgen Döllner and Klaus Hinrichs. A generic rendering system. In *IEEE Transactions on Visualization and Computer Graphics, 8(2)*, pages 99–118, 2002.

[9] John Kelso, Lance E. Arsenault, Steven G. Satterfield, and Ronald D. Kriz. DIVERSE: A framework for building extensible and reconfigurable device independent virtual environments. In *Proceedings of IEEE Virtual Reality 2002 Conference*, pages 183–190, 2002.

[10] Oliver Kersting and Jürgen Döllner. Interactively developing 3D graphics applications in Tcl. In *USENIX Annual Technical Conference*, pages 99–118, 2002.

[11] Randy Pausch, Tommy Burnette, A. C. Capehar, Matthew Conway, Dennis Cosgrove, Rob DeLine, Jim Durbin, Rich Gossweiler, Shuichi Koga, and Jeff White. A brief architectural overview of Alice, a rapid prototyping system for virtual reality. In *IEEE Computer Graphics and Applications*, pages 195–203, 1995.

[12] Timo Ropinski, Frank Steinicke, and Klaus Hinrichs. Tentative results in focus-based medical volume visualization. In *Conference Supplements of Smartgraphics 2005 (in print)*, 2005.

[13] Dietmar Schmalstieg, Anton L. Fuhrmann, Gerd Hesina, Zsolt Szalavari, Miguel Encarnação, Michael Gervautz, and Werner Purgathofer. The studierstube augmented reality project. In *PRESENCE - Teleoperators and Virtual Environments 11(1)*, pages 32–45, 2002.

[14] Chris Shaw, Mark Green, Jiandong Liang, and Yunqi Sun. Decoupled simulation in virtual reality with the MR toolkit. In *ACM Transactions on Information Systems 11(3)*, pages 287–317, 1993.

[15] Frank Steinicke, Klaus Hinrichs, and Timo Ropinski. Virtual reflections and virtual shadows in mixed reality environments. In *Short Paper Proceedings of the 10th International Conference on Human-Computer Interaction (INTERACT05)*, pages 1018–1021, 2005.

[16] Frank Steinicke, Timo Ropinski, and Klaus Hinrichs. Object selection in virtual environments with an improved virtual pointer metaphor. In *International Conference on Computer Vision and Graphics (ICCVG)*, pages 320–326, 2004.

[17] Frank Steinicke, Timo Ropinski, and Klaus Hinrichs. Multimodal interaction metaphors for manipulation of distant objects in immersive virtual environments. In *In Short Paper Proceedings of the 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG05)*, pages 45–48, 2005.

[18] Frank Steinicke, Timo Ropinski, and Klaus Hinrichs. VR and laser-based interaction in virtual environments using a dual-purpose interaction metaphor. In *IEEE VR 2005 Workshop Proceedings on New Directions in 3D User Interfaces*, pages 61–64, 2005.

[19] Henrik Tramberend. AVANGO: A distributed virtual reality framework. In *Proceedings of the IEEE Virtual Reality '99*, 1999.