Knowl Inf Syst (2007)
DOI 10.1007/s10115-007-0066-6

Knowledge and Information Systems

REGULAR PAPER

Arjan J. F. Kok · Robert van Liere

A multimodal virtual reality interface for 3D interaction with VTK

Received: 3 October 2005 / Revised: 15 November 2006 / Accepted: 6 December 2006

Abstract The object-oriented visualization Toolkit (VTK) is widely used for scientific visualization. VTK is a visualization library that provides a large number of functions for presenting three-dimensional data. Interaction with the visualized data is controlled with two-dimensional input devices, such as mouse and keyboard. Support for real three-dimensional and multimodal input is non-existent. This paper describes VR-VTK: a multimodal interface to VTK on a virtual environment. Six degree of freedom input devices are used for spatial 3D interaction. They control the 3D widgets that are used to interact with the visualized data. Head tracking is used for camera control. Pedals are used for clutching. Speech input is used for application commands and system control. To address several problems specific for spatial 3D interaction, a number of additional features, such as more complex interaction methods and enhanced depth perception, are discussed. Furthermore, the need for multimodal input to support interaction with the visualization is shown. Two existing VTK applications are ported using VR-VTK to run in a desktop virtual reality system. Informal user experiences are presented.

Keywords Visualization · Virtual environments · 3D interaction

A. J. F. Kok (✉)
Department of Computer Science, Open Universiteit Nederland, The Netherlands
E-mail: arjan.kok@ou.nl

R. van Liere
Department of Information Systems, Center for Mathematics and Computer Science, Amsterdam, The Netherlands
E-mail: robertl@cwi.nl

1 Introduction

Throughout the past years, various authors have reported on the benefits of a virtual reality (VR) interface to scientific visualization [4, 24]. The key argument is that a well-designed VR interface can provide the sensation of presence and stimulates participation with objects in a scene. Such a VR interface needs support for 3D interaction with six degree of freedom (6 DOF) trackers and support for system control, for example, by speech. Clearly, this may lead to multimodal interfaces in which the user has more control of how the scientific data is presented and explored.

The object oriented visualization Toolkit (VTK) is rapidly becoming the standard for scientific visualization toolkits [18]. It is an open source class library that contains a large number of functions for the presentation of scientific data. VTK uses a pipeline mechanism for rendering. Data is passed through various pipeline objects (e.g. filters and mappers) to obtain geometry that can be displayed by the renderer.

For interaction with the data, VTK employs the concepts of picking and 3D widgets. Picking is used to select objects in the visualization, while widgets are used to interact with objects in specific ways. A widget has a visual representation within the 3D visualization and defines the behavior that is executed when the widget is manipulated [5]. Simple examples of 3D widgets are the point widget for probing object information, the box widget for positioning, rotating, and scaling of objects, the spline widget for defining a spline by editing control points, etc.

VTK widgets are currently controlled with 2D input devices, such as mouse and keyboard. An important question is whether a user interface to VTK can be improved when widgets are controlled with different modalities in a virtual environment. For example, what is the benefit of using 6 DOF input devices to control widgets? In what way can other input modalities, such as speech and gestures be used? In addition to controlling a widget with one specific input modality, it can be beneficial to combine modalities [12]: Two or more input modalities can be combined to define one action, different input modalities can be used concurrently to execute concurrent actions, and one input modality can be used to enable/disable one of the other modalities.

In this paper, we present a multimodal VR interface to VTK. Hands are used for direct 3D manipulation using widgets and pickers, feet for clutching, head for camera control, and speech for system control. In the next section, we position our solution against other approaches that have been taken to use VTK in a virtual environment. Section 3 shows how the VTK class hierarchy has been extended. In Sect. 4 we show how the support for direct manipulation has been added. It includes the basic picking and widget manipulation, multimodal interaction with widgets, some enhanced interaction features, and ways to allow more accurate 3D interaction by means of better depth perception. Section 5 shows how a VR interface can be extended with visualization and application control. Section 6 gives examples of real applications that have been implemented using our interface. It includes a description of the used hardware platform. Finally, the last section consists of a discussion of our design.

2 Previous work

We have chosen three points of view to position our solution against others. First, from an engineering point of view, in which we discuss how other researchers have approached the problem of using VTK in a virtual environment. Second, from a multi-modal interface point of view, in which we discuss how multi-modalities can be used to control widgets. Finally, from a 3D user interface approach, in which we discuss various performance issues in 3D interaction.

2.1 VTK in virtual reality

Several visualization systems researchers have investigated the integration of VTK with a virtual reality (VR) system.

One approach is the vtkActorToPF library [13]. In this approach, generation of visualization geometry is decoupled from the rendering of this geometry. VTK generates the geometry in the form of actors that consist of geometry and properties. A new class, vtkActorToPF, transforms these actors into pfGeodes that are included in a Performer scenegraph. This scenegraph is rendered independently of VTK. The advantage of this method is the explicit decoupling of rendering from geometry creation, by which the rendering is not interrupted because of complex geometry creation. Only whenever geometry is completely created by VTK, it is transformed to Performer. Disadvantages are the fact that the VTK cameras and lights are not considered in the Performer environment and the fact that there is no support for direct interaction with the VTK pipeline from the rendering. Objects can only be manipulated in the Performer application.

Another approach is based on the modification of the VTK renderer and render window [20, 27]. The renderer and render window are the objects that are responsible for rendering the objects to screen(s). By using derived classes of the renderer and render window that include virtual reality system dependent functions, rendering is done onto the VR system. All other functionality of VTK, including interaction using the mouse, remains available to the user. Advantages of this approach are the rather simple implementation and the fact that all VTK functionality is available for interaction. All objects in a pipeline can directly be modified as result of a user action.

We adapted the second approach. However, we have a different focus. To integrate VTK with VR there are two problems to solve:

- *Real-time update of the visualization*: This is needed to get the impression of motion in the visualization and, more importantly, to be able to do 3D interaction. Several solutions are available, such as decoupling rendering from processing, parallelizing the visualization and processing, proper data management, etc.
- *Efficient comfortable interaction*: We want to directly manipulate the visualization in 3D. That is much more difficult than interaction in 2D. Special care should be taken to investigate ways to improve the efficiency and comfort of direct 3D manipulation. Furthermore, it must be possible to combine the spatial 3D interaction with non-spatial interaction to allow for changing interaction modes, changing the visualization, controlling the application, etc.

Most efforts to integrate VTK with VR systems address the problem of real-time updates [4, 13, 27]. Our focus is on efficient and comfortable direct manipulation in 3D and on efficient visualization and application control.

2.2 Multimodal widgets

Smalltalk's Model-View-Controller abstraction can be used to model a VTK widget [6]. Every widget has a view (graphical representation on display), which provides the user with the feedback of the state of the widget. A box widget, for example, is represented as a wireframe of a box augmented with small handles for translation and scaling of the representation. The model of a widget encapsulates the widgets semantics, i.e. which operations are feasible. For example, the semantics of a box widget define that an object, controlled by this widget, is rotated when the user moves a plane of the widget, and that the object is scaled when a handle is dragged. The controllers are the devices used to manipulate the widget. A mouse can be used to drag the handles of the box widget representation.

In our VR-VTK system we turned the VTK widgets into multimodal widgets by extending the controllers, see Sect. 4.3. We have explored the usage of 6 DOF devices, pedals and speech as controllers. The user uses two hands to control the widget. The non-dominant hand is used to control the position and orientation of 3D objects in the virtual environment. The dominant hand is used to control the handles of the widget. Having both hands occupied interacting with the data prevents the user from using a keyboard or mouse at the same time. Therefore, other input modalities have been added. Foot pedals are used to simulate button actions, such as button press and release. The 3D widget mechanism also allows us to use a gesture recognition interface as controller to resize or move an object using the box widget.

To our knowledge there has been no other researcher that has explored the usage of multimodal VTK widgets. Multimodal interfaces, however, have been applied for visualization applications before, e.g. [11, 12, 21]. They all combine gestures with speech to navigate through the environment or to manipulate the virtual objects. As outlined above, our approach is different. In our multimodal interface, 3D interaction is driven by 3D widgets. These widgets are primarily controlled by direct 3D manipulation, but can also be controlled by other modalities.

2.3 Human factors

From the user interface community it is known that point location is a basic task in the design and implementation of 3D user interfaces. Point location, the task of positioning the cursor on a point in 3D, is the primitive operation used for tasks such as object selection and object manipulation. Many user studies have been performed to investigate the efficiency of point location subject to different hardware conditions (stereoscopic displays, head tracked displays) [1, 3]. In addition, user studies have been done to understand the effect that different depth cues have on the user performance of basic 3D tasks. For example, studies have reported

on the relative importance of auxiliary 3D depth cues (stereo, lighting conditions, perspective viewing and shadows) on interactive 3D tasks [7, 9, 25, 29].

We have taken the results of these studies into account in the design of VR-VTK. We have developed the notion of cursor regions in order to aid the user to perform the VTK pick in a three-dimensional environment. In addition, several cues have been introduced to enhance depth perception. These enhancements will be discussed in Sect. 4.

To our knowledge no user interface researcher has done formal user performance studies on the usage of 3D widgets.

3 VR-VTK architecture

Basic assumption in developing a virtual reality version of VTK is that the user must be able to port his existing VTK applications with minimal effort. Technical aspects of VR systems (support for input devices like trackers, pedals, keyboard, speech, support for multi-threading, . . .) should be hidden from the user.

The VR-VTK layer extends VTK with virtual reality support, see Fig. 1. This layer contains a number of new VR-VTK classes, most of them derived from the existing VTK classes (for modification of functionality), and some of them really new classes (to add new functionality). Note that only classes involved in interaction and rendering have to be added to this layer. The visualization pipeline classes (data objects, filters, mappers, etc.) can be used without any modification.

In "standard" VTK each render window (vtkRenderWindow) can be associated with an interactor (vtkRenderWindowInteractor). This interactor captures the windowing system specific events (e.g. mouse and keyboard events) in the render window, translates these system dependent events into system independent VTK events, and dispatches these VTK events. Interactor observers, such as interactor styles (vtkInteractorStyle) and widgets (vtk3DWidget), define the behavior associated with particular VTK events. They intercept these events and perform the defined action. For example, upon receipt of a mouse move event, the interactor style moves the camera of the renderer.

3.1 Render window and renderer

In VR-VTK, for rendering, subclasses of vtkRenderWindow and vtkRenderer have been developed. They control the rendering of the scene for a VR system.

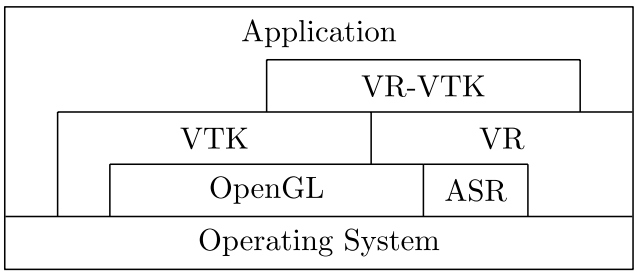

Fig. 1 VR-VTK architecture

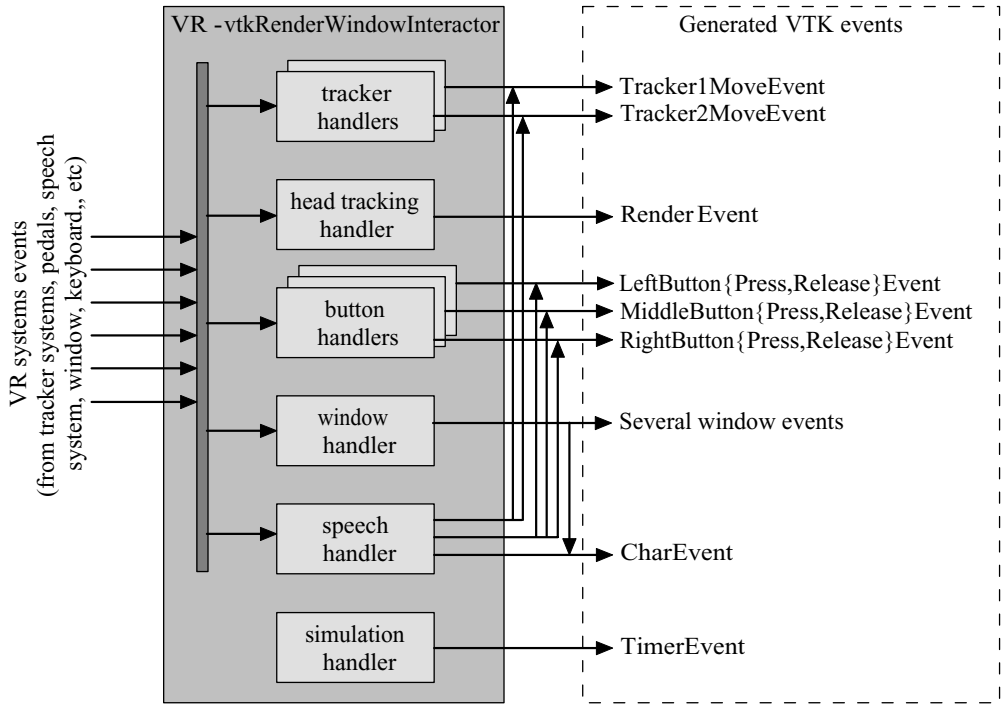

Fig. 2 The VR-VTK render window interactor. It captures VR system events and translates these events into VTK events. All event handlers run in parallel

The VR-vtkRenderWindow decouples rendering from interaction by creating a separate thread for rendering. Rendering by this thread is triggered by render events. These events are invoked by the head tracker. Also, the Render method of the render window, which is usually called when the scene is modified is overridden. Instead of directly rendering the scene into the render window, it now invokes a render event. A new event handling method of VR-vtkRenderWindow now does the actual rendering of the scene. Furthermore, the VR-vtkRenderer performs some extra operations to visualize the scene on the VR system properly. For example, it supports the rendering of shadows (see Sect. 4.5) and the rendering of the VR-VTK 2D widget interface (see Sect. 5.2).

3.2 Render window interactor

Central component in the VR-VTK library is the VR-vtkRenderWindowInteractor. Its main task is to capture and translate events from the VR system into events that can be interpreted by the VTK pipeline, that means into VTK events. At creation of the VR-vtkRenderWindowInteractor several event handlers are created (see also Fig. 2):

– *Two tracker handlers (one tracker for each hand)*: These handlers receive events from the tracking devices containing position and orientation (pose) of the trackers. Upon receipt of a tracker event, the pose of the tracker is stored in the interactor and a VTK event is invoked to process the event in the application. For that, two new VTK events are introduced: a Tracker1MoveEvent

that represents an event from the tracker in the dominant hand (this event replaces the MouseMoveEvent used in 2D VTK) and a Tracker2MoveEvent that represents an event from a tracker in the non-dominant hand.
- *A head tracking handler*: This handler receives head tracking events from the head tracking device. A head tracking event contains the head position of the user. This position is put into the active camera and an event to redraw the scene is issued.
- *Three button handlers*: These handlers receive events from button devices used in the VR system (e.g. mouse buttons or pedals). These are used as clutching devices. Upon receipt of a button event, the corresponding VTK button event is invoked.
- *A window handler*: This handler receives events from the window system (e.g. key press/release events, window events). Window events can be used to show/hide or resize the window. All these events are converted to their corresponding VTK events. Keyboard commands are converted to VTK character events and can be used to control the visualization and the application. This handler performs the tasks of the "standard" interactors.
- *A speech handler*: This handler receives speech events from an external automatic speech recognition (ASR) system. It converts these speech events into several VTK events: character events, button events or (the new) tracker events.
  The mapping to character events allows that all objects in "standard" VTK that react on keyboard events (such as styles and widgets) now automatically react on speech input. Therefore, speech commands are used to control the visualization and the application (see Sect. 5.1).
  Speech events mapped to button and tracker events are used to control the multimodal interaction using 3D widgets (see Sect. 4.3).
- *A simulation handler*: The simulation handler generates VTK timer events in a user-specified frequency. Users can register an observer with the VR-vtkRenderWindowInteractor that listens to these timer events. Upon receipt of a timer event the observer will execute the actions associated with the observer.
  The simulation handler can be used, for example, to integrate a simulation or an animation in the VTK pipeline.

Each of these handlers runs in its own thread. Therefore, parallel performance of interactions is feasible [22].

3.3 Interactor observers

The VR-vtkInteractorStyle defines the "look and feel" of the interaction and visualization environment. It is driven by VTK events (for example, those generated by the interactor, such as VTK tracker, button and character events). Its main tasks are controlling the picking process and the enhanced interaction methods (see Sect. 4), controlling features to enhance depth perception (see Sect. 4.5), and controlling the visualization (see Sect. 5).

The VR-VTK widgets are subclasses of the available VTK widgets. Just as the interactor style, these widgets observe VTK events. In stead of acting on mouse

move events, as the standard widgets do, the VR-VTK widgets act upon receipt of TrackerMoveEvents. They use the real 3D position and orientation information stored in the VR-vtkRenderWindowInteractor to perform their action (see also Sect. 4.3).

3.4 Platform independence

VR-VTK is generally applicable. It is not limited to specific hardware (platform, trackers, etc.). All VR-VTK classes are independent of specific hardware. Most of them are controlled by VTK events only. Even the VR-vtkRenderWindowInteractor does not depend on the used hardware. This class is implemented using a VR library, currently PVR [26]. The VR-library hides all device details from VR-VTK. The VR-vtkRenderWindowInteractor class receives events from this library (instead of from the devices directly), and translates these to VTK-events. It is not much work to modify the VR-vtkRenderWindowInteractor class to use other VR libraries, such as the MR toolkit [22] and VRPN [23]. These VR libraries allow us to use VR-VTK on other VR platforms, such as the CAVE. New input devices can be added easily, without the need of modifying VR-VTK, as long as the VR library supports these devices.

4 3D interaction

Interacting with a 3D visualization will typically include the manipulation of graphical objects. In general, direct manipulation can be decomposed in three tasks: start the interaction by selecting the object, perform the interaction by dragging the object and ending the interaction by releasing the object. VR-VTK provides three basic mechanisms that are used to support these tasks: 3D cursor regions, 3D picking (both controlled by the VR-vtkInteractionStyle) and 3D widgets.

4.1 3D cursor regions

VTK uses a 2D cursor as feedback to indicate the position of the 2D input device. The straightforward analog would be to use a 3D cursor for the position/orientation of the 3D spatial input device. However, to address the difficulties that users have with point location in 3D, VR-VTK uses the notion of a cursor region.

A cursor region is defined as a region in 3D space that is used to simplify selection. Objects that coincide with the region are objects that can be selected. Two regions are available: a cylinder and a sphere. Regions can be parameterized with two parameters:

- *An offset*: This offset defines the distance from the user's hand to the centre of the cursor region. An offset of zero results in direct manipulation, as the user has the regions directly in his hands. The offset can be enlarged, if the objects to be manipulated are not within arm's reach.

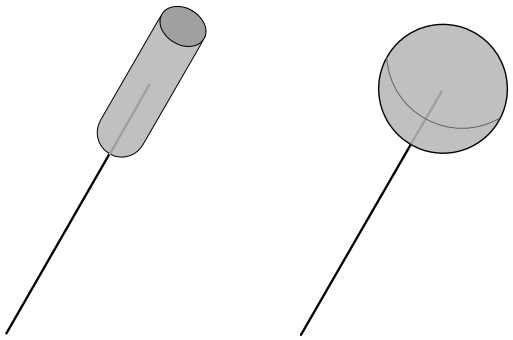

Fig. 3 Cursor region feedback: cylinder (*left*) and sphere (*right*). Parameters control the offset and size of the region

– *The size of the region*: The size of the region defines the accuracy required for manipulation. If very precise manipulation is required, the size of the region can be set to almost zero (approaching 3D point location), while a large size can be used when accuracy is less important or the objects in the scene are large.

Cursor region feedback is continuously given by drawing a ray for the offset and a cylinder or sphere at the appropriate position for the size of the region, see Fig. 3.

4.2 3D pick

The cursor region defines a picking volume: when picking, the object intersected by the cursor region that is closest to the center of this region is picked.

When picking in 2D it is always clear which object will be picked: the one that is visible in the pixel under the 2D cursor. In 3D it is not always clear which object is picked: the cursor region may intersect several objects, or the selected object is (partly) hidden by other objects. Therefore, a continuous feedback is given by highlighting the selected object, by placing a bounding box around the selected object.

The 3D pick is started by a button press event (in our environment we usually use foot pedals or speech). Upon press, the application can activate a VR-VTK picker object and call the VR-VTK pick function. While the button remains pressed, the selected object is dragged: its position is updated using the position of the cursor region. Upon button release, the interaction stops.

4.3 3D widgets

VTK provides a set of 3D widgets to allow many types of interaction and control of many shapes (points, lines, planes, volumes), e.g. vtkPointWidget, vtkLineWidget, vtkBoxWidget, etc. [19]. These widgets in VTK are controlled by 2D mouse movements and mouse button events. The 3D widget receives VTK events invoked by the interactor, and takes appropriate action.

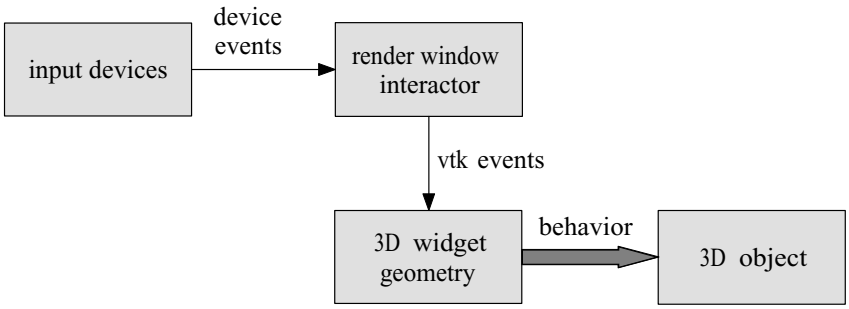

Fig. 4 Interaction with a 3D widget: Events from input devices are translated by the render window interactor into VTK events. These events control the shape, position and orientation of the geometry of the widget, which in turn control the behavior of the object associated by the widget

In VR-VTK all 3D interaction with the data visualization is done by manipulating 3D widgets. To interact with an object, a widget has to be linked to it. The type of widget determines the semantics of the action, i.e. which operations on the object are feasible. Each widget has some geometry; for example, spherical handles to define size and shape of the widget, and arrows to define orientation. Manipulation of the widget is done by interacting with this geometry.

Figure 4 shows the steps performed from manipulation of an input device to behavior of an object controlled by the widget. Operation of the widgets is controlled by events from the VR-vtkRenderWindowInteractor and the position and orientation of the cursor region. This information defines the modification of the geometry of the widget, which in turn defines the response of the object.

Interaction with the widgets can be done with different modalities (input devices) and different levels of control:

- *By 3D manipulation of the geometry of the widget:* The 3D tracker controls the cursor region that is used to select and move the handles of the widget in 3D. Pedals or speech are used for clutching.
- *By speech in combination with tracking to control the geometry of the widget:* The 3D tracker, represented by the cursor region, points to a position. A speech command that specifies a handle and an action (e.g. “put that handle there”) defines an action on the handle.
- *By speech to control the whole widget:* A speech command that specifies a widget and an action (e.g. “rotate that widget by ...”) defines an action on the widget.

Figure 5 shows an example of a 3D widget used to define a slicing plane. The handles at the corners of the plane can be used to enlarge/reduce the size of the plane, the arrow handle to modify the orientation of the plane while the plane itself can be used to move the plane.

The set of widgets can be extended to include more complex tasks. More complex widgets can be built from the basic widgets. VR-VTK contains, for example, a convex-hull widget (to define convex volumes) built from several point widgets.

For complex visualization and manipulation, the scene may contain several widgets. To avoid the scene to become cluttered with manipulable object representing these widgets, VTK allows for selective enabling/disabling widgets (by

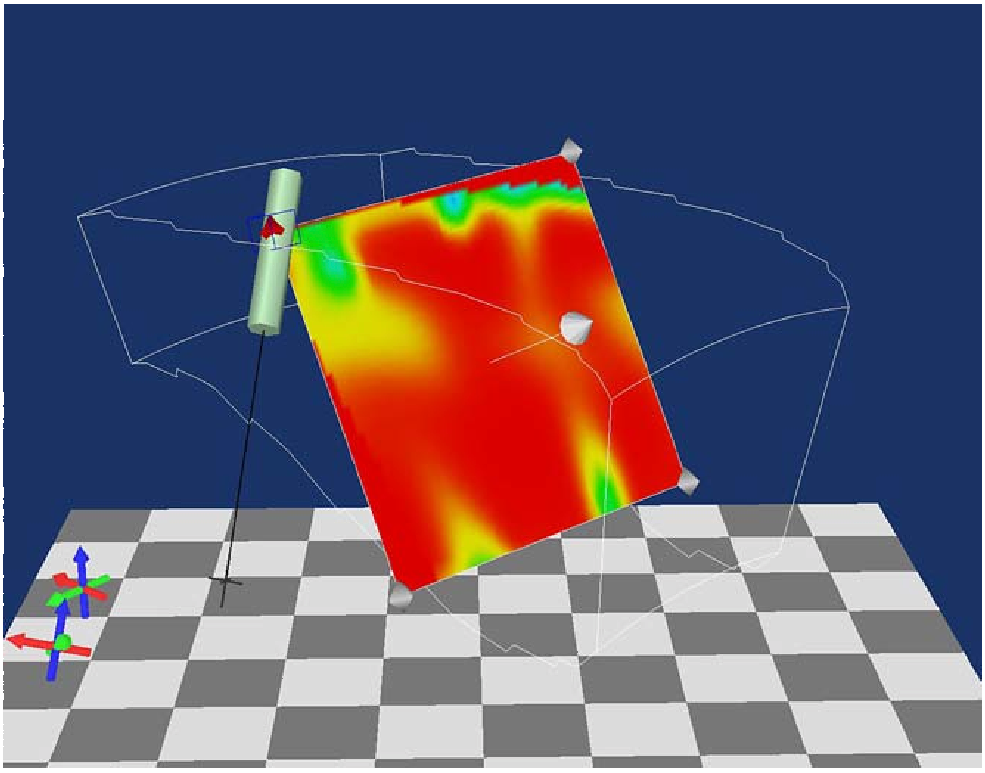

Fig. 5 A 3D widget to control a slicing plane. The cursor region selects a widget handle and can move this handle to change the size and position of the slicing plane

pressing a key). A disabled widget disappears from the environment. In VR-VTK enabling/disabling widgets can also be done by speech commands.

4.4 Enhanced 3D interaction

There are two ways to enhance 3D interaction for visualization:

- *World in hand*: In scientific visualization it is often useful to position and orientate the complete scene. The tracker in the non-dominant hand can be used to control the position and orientation of the complete visualization (world in hand). This makes it possible to inspect the world from all sides by rotating the non-dominant hand. The world in hand metaphor also improves interaction with the world. Rotating the world such that the objects to be manipulated are not hidden and within reach of the user, makes interaction more easy. The dominant hand can then be used to control the 3D cursor region and therefore performs the real operations on the visualization and data.
- *Two-handed input*: Two-handed input can be beneficial for interaction tasks. Tasks can be performed in parallel, but more importantly, tasks can sometimes be done more accurately than with one hand [8]. In two-handed input, both the dominant and the non-dominant hand control a cursor region. This kind of two-handed input is used for composite interaction tasks, e.g. object docking where each cursor region controls the position and orientation of an object, simultaneously positioning the two handles of a line widget, etc.

These enhanced interaction options can be enabled by speech commands.

4.5 Enhanced depth perception

One of the major problems with interaction in three dimensions is the lack of depth perception, even if stereoscopic rendering is used. It is often very hard to determine which objects are closer to the viewer than others. That makes it very hard to select an object: are we reaching far enough into the environment to select the object, or not?

There are several ways to add extra depth cues [29]:

- *Motion parallax*: Motion parallax is a powerful depth cue. Moving the head from side to side, objects closer to the viewer move relatively faster than objects that are further away. To achieve motion parallax, the head tracker is directly coupled with the active VTK camera. Moving the head changes the camera position.
- *The stage*: The stage consists of a checkerboard and two coordinate axes. The checkerboard serves as reference frame of the interaction space. Having the trackers in the area above the checkerboard will ensure that they are tracked properly. Furthermore, the checkerboard pattern on the floor enhances the depth perception in the virtual environment. The coordinate axes show the orientation of both trackers. They help the user orienting objects in the virtual environment. Figure 6 shows the stage.
- *Shadows*: A virtual light source is placed above the virtual world. Using this light source, shadows for the objects in the virtual world are projected onto the

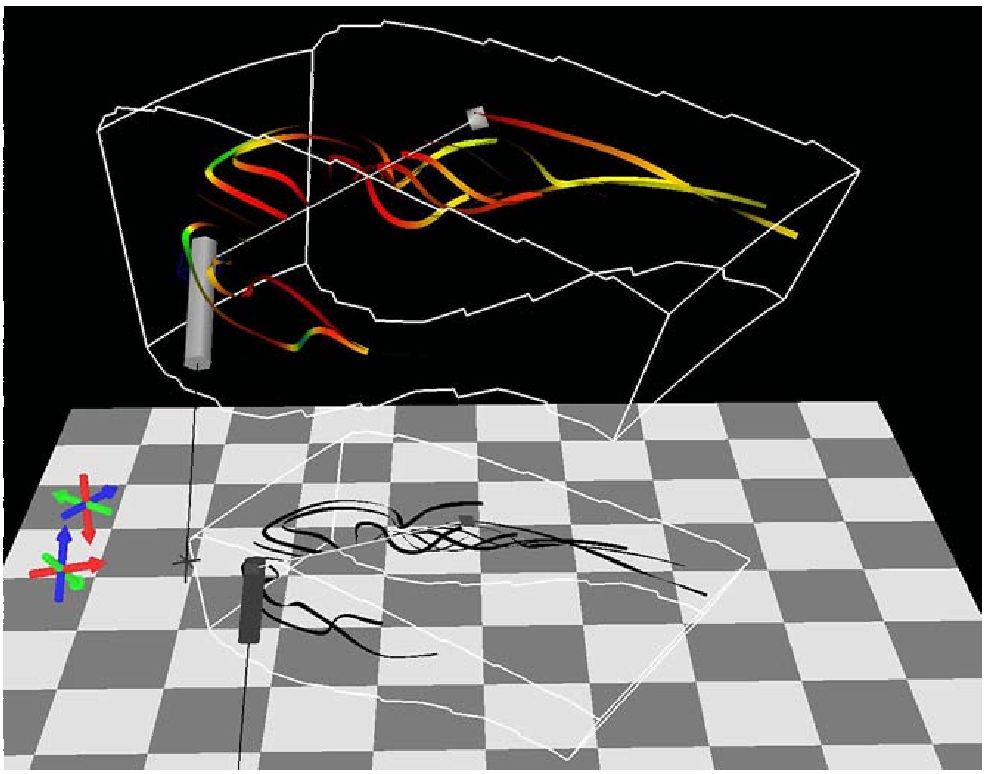

Fig. 6 Stage: The stage consists of a checkerboard and two coordinates axes (*on the left*). Shadows on the stage enhance depth perception

stage, see Fig. 6. These shadows make it much easier to see what the spatial position of an object is.

The number of objects in the virtual world can be high, resulting in many shadows. Apart from the high demands on rendering time, the high number of shadows may result in a clutter of shadows on the stage. In that case the shadows are of no use, as we cannot see what object caused what shadow [9, 25]. We are, however, only interested in the spatial positions of certain objects:

- The items that can be picked and manipulated (e.g. the handles of the 3D widgets).
- The cursor region. We are interested in the position of the cursor region in relation to the position of the pickable objects.
- Some reference objects, e.g. objects that are important for positioning manipulable objects, for example, the outline of the data set.

Therefore, in VR-VTK the user can define which objects should generate a shadow. By default all widget handles and the picking regions generate shadows, all other objects do not generate shadows, unless the user specifies differently.
The shadows themselves might also be used as 3D widgets [7]. When the shadows are implemented as pickable objects, these shadows can be selected and moved over the stage. An object (e.g. a handle of a widget) in the scene can be manipulated by manipulating its shadow.

5 Application control

Besides direct 3D object manipulation, VTK applications need support for non-spatial input, e.g. for control tasks. Examples of these tasks in our VR-VTK environment are: enabling all kind of options such as stereo, shadows, the world-in-hand and the stage, enabling widgets for 3D interaction (see Sect. 4.3), enabling the pipeline browser (see Sect. 5.2.2), zoom in and out, and setting properties for the cursor region. Examples of these tasks in a user application might be: enabling visualization methods, setting parameters, setting properties for objects, and modifying color tables.

In a 2D desktop application these controls are usually implemented as widgets using a 2D user interface (UI) toolkit, such as Tcl/Tk or Java Swing, operated by mouse and keyboard. However, for our 3D interface this is not the most suitable solution. Both hands are used for steering trackers, so we have to drop one of the trackers to be able to use the mouse or keyboard. Moreover, we have to move our attention from the 3D environment to the 2D GUI or the keyboard. Therefore, other solutions are required.

5.1 Speech input

Speech has shown to be a useful modality for system control [2, 17]. Commands can be issued in parallel with spatial interaction while user's attention can stay focused on the 3D environment.

The VR-VTK interface contains a small "standard" vocabulary to support basic visualization control. It includes commands to control picking, setting the enhanced 3D interaction methods, setting the enhanced depth cues and zooming. Furthermore, it contains commands to control the rendering process, such as enabling stereo and making screen dumps. Important is the command to transfer interaction from interaction with the data to interaction with the UI-toolkit (see Sect. 5.2) vice versa.

The vocabulary is easily extended with speech commands for the user application. As the speech handler converts speech events into character events, the user only has to specify the mapping from speech command to a character. The application itself should listen to character events.

5.2 2D widgets in a 3D world

Since many people have experience with 2D widget interfaces on the desktop, it is an obvious choice to use them in the virtual world as well. Mine et al. [14] discussed how these widget systems should be used integrated in a virtual environment, and concluded that hand-held widgets performed best.

5.2.1 VR-VTK UI

Therefore, the VR-VTK UI toolkit consists of a virtual cube containing 2D widgets, and a virtual pen as selection device to manipulate the widgets. Each side of the virtual cube can be equipped with all kind of widgets, such as buttons, pop-up menus, sliders, labels, message boxes, etc. Having six sides on a cube makes it possible to arrange widgets in groups. Desktop applications often use a main menu and several windows. In our system the main menu and each window can be placed on another side of the cube.

The virtual cube and virtual pen are controlled by trackers, in our system by graspable input devices [10]. Instead of using the selection device, in future, it will also be possible to manipulate the 2D widgets by speech commands.

By pressing a specific button (e.g. a pedal) or by a speech command, control switches from visualization interaction to UI manipulation. The widget interface is shown over the visualization. The user can now change some control parameters on the cube by manipulating the 2D widgets. Results of these changes are directly visible in the data visualization. When the specific button or speech command is issued again, the widget interface disappears and direct 3D interaction with the data visualization can proceed.

5.2.2 VTK pipeline browser

The VR-VTK pipeline browser is a special GUI built with the VR-VTK UI toolkit. It enables the user to inspect and modify all parameters of all objects in the VTK pipeline. The pipeline browser has the same functionality as the tcl/tk pipeline browser [16]. However, our browser is rendered and controlled inside the same 3D space as the VTK visualization.

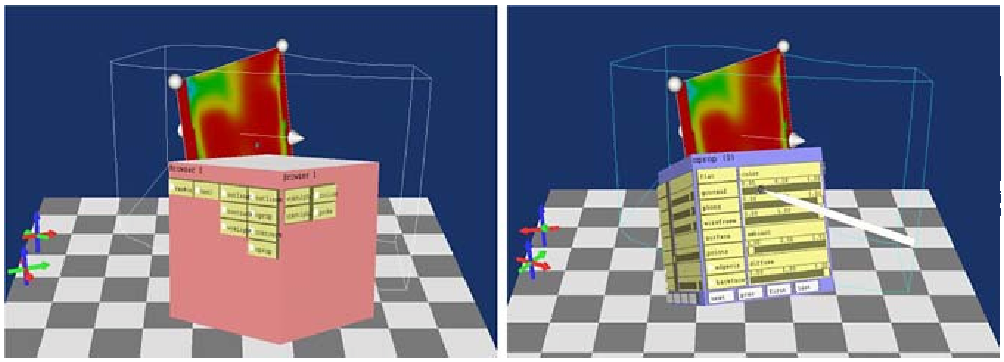

Fig. 7 The pipeline browser: the pipeline interface (*left*), the object interface (*right*)

The pipeline browser supports two views (see Fig. 7). One view of the VTK pipeline shows a connected graph on one or more faces of the cube (the pipeline interface). It contains one widget (checkbutton) for each object (data, filter, mapper, actor, light, etc.) in the visualization pipeline. The other view shows the properties/methods of VTK objects (the object interface). Users can use the pen to select one or more objects in the pipeline interface. A user interface for each selected object will be rendered on a face of the cube. The object's set/get methods are automatically shown as sliders, (check) buttons or menus. Object properties can be modified by manipulating these widgets.

The pipeline browser interface is generated automatically. However, since a pipeline can consist of many VTK objects, and each pipeline object can have many manipulable properties, the user can choose for a subset of these for display.

6 Examples

We ported several existing VTK applications to run in our virtual reality system, the Personal Space Station (PSS), by applying our new multimodal VR interface.

6.1 The Personal Space Station

The Personal Space Station (PSS) is a near-field, mirror-based desktop VR/AR environment [15], see Figs. 8 and 9. A head-tracked user is seated in front of a mirror, which reflects the stereoscopic images of the virtual world as displayed by the monitor. The user perceives the stereoscopic images of the display as a virtual world located in front of him. The user can reach under the mirror into the virtual world to interact with the virtual objects.

Spatial interaction is performed directly in the 3D workspace with one or more graspable input devices (props). The interaction space is monitored by two or more cameras. Each camera is equipped with an infra-red (IR) pass filter in front of the lens and a ring of IR LEDs around the lens. Patterns of retroreflective markers are applied to the tracked input devices. The optical tracking system uses these marker patterns to identify and reconstruct the poses of the devices. In this way, graspable, wireless input devices can easily be constructed.

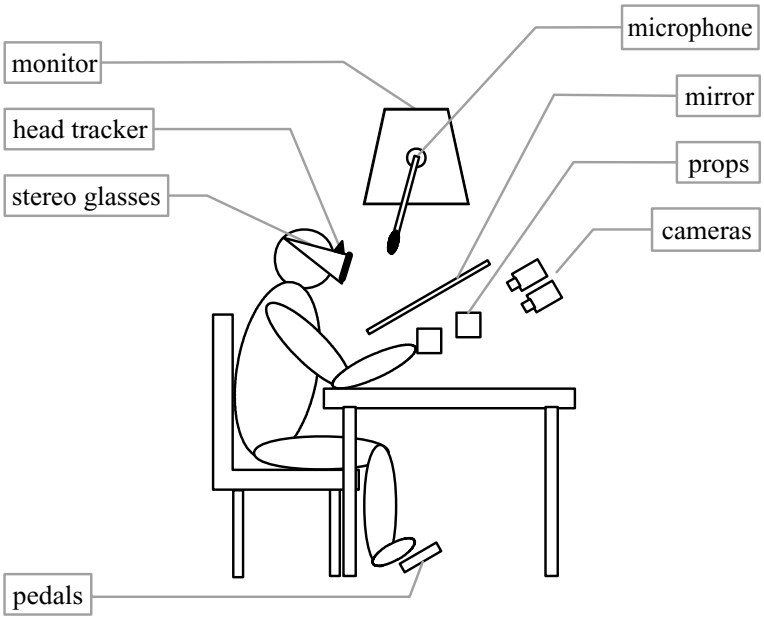

Fig. 8 Schematic view of the Personal Space Station

Besides these input devices, the PSS is equipped with three foot pedals as an equivalent to standard mouse buttons. A microphone is used for the speech recognition system. Furthermore, standard desktop input devices, such as a keyboard and mouse, can be used.

Figure 9 shows the PSS in use. The user holds two 6 DOF input devices to directly manipulate VTK pickers and VTK widgets.

VR-VTK is not limited to the PSS and can be used in other virtual environments, such as the CAVE.

6.2 Diffusion tensor imaging

MR diffusion tensor imaging (DTI) measures the diffusion of water molecules in tissue. The diffusion anisotropy is an indicator of the underlying structure of the tissue, for example, of the fibers in the brain. The structure is fairly complex. It is difficult to understand and to get insight in the structure when, for example, the number of fibers is reasonably large. Therefore the use of 3D visualization and 3D interaction can help on getting a better understanding of such complex data. A visualization tool has been developed, which allows the visualization of the 3D structure that can be obtained from the diffusion tensors [28]. This tool is used for brain and muscle studies.

Figure 10 shows DTI visualization and interaction using our 3D interface. Interactive visualization is provided by:

- *Orthogonal slice planes*: The slice planes visualize the anisotropy by mapping anisotropy indices to color or grey values. Visualization and interaction with these planes is controlled by three image plane widgets. They allow selecting and moving the slice planes through the data set.

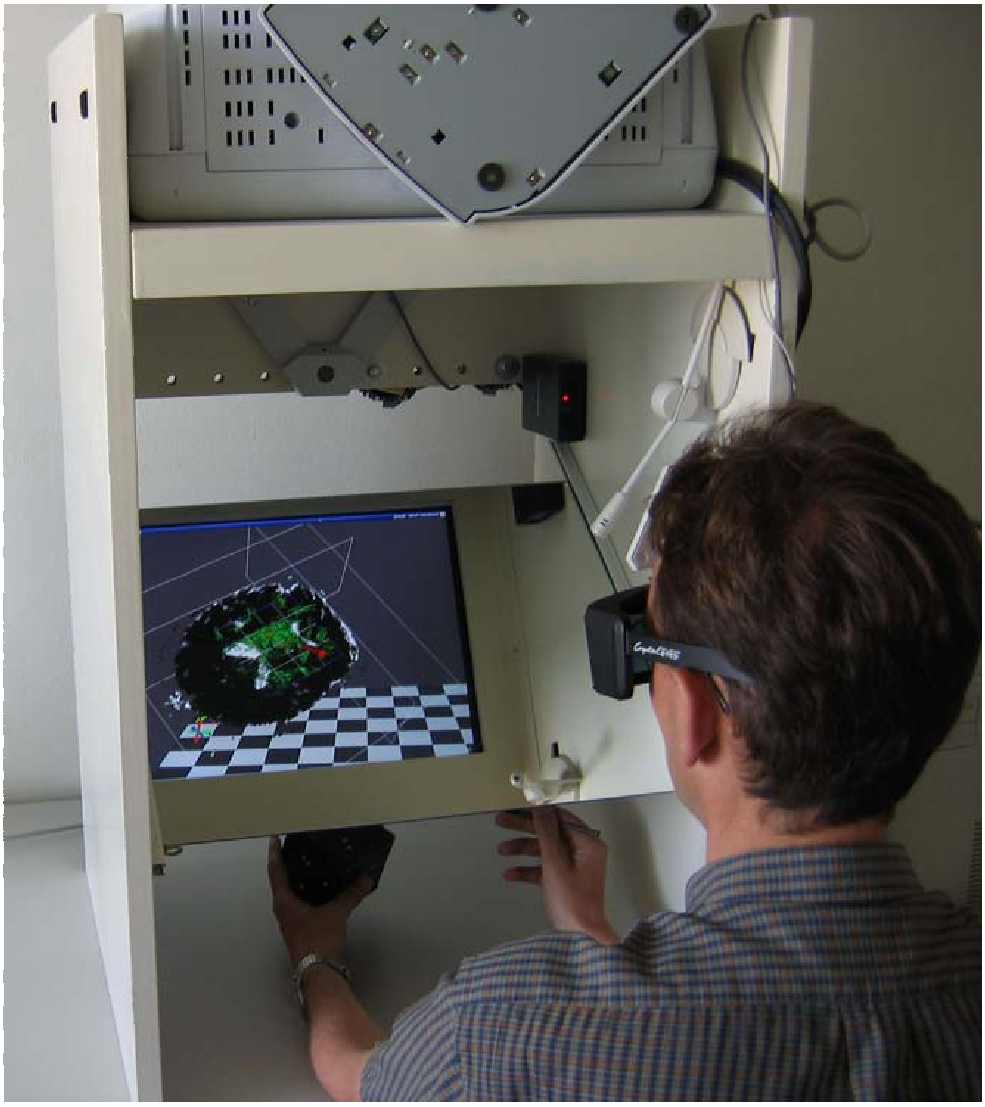

Fig. 9 Using spatial devices to directly manipulate VTK objects. A pen and cube device to position fibers in a diffusion tensor data set. The cube is used to position the data set, while the pen is used for precise interaction. System control is provided by speech

- *Glyphing*: The diffusion tensors are visualized by ellipsoid or cuboid glyphs. The sample points of the glyphs are defined interactively by widgets. Depending on the number and organization of requested sample points a widget is chosen (point, line, plane, or box widget). The point widget controls the position of only one sample point, while the box widget controls the size, orientation, and position of a volume with sample points.
- *Fiber tracking*: The diffusion tensor data can be visualized by streamlines, hyperstreamlines and streamtubes. The seed points are defined interactively with widgets, just like the sample point definition of glyphing.
- *World in hand*: The user can hold the data set (brains) in his hand and move and rotate it to get the best view.

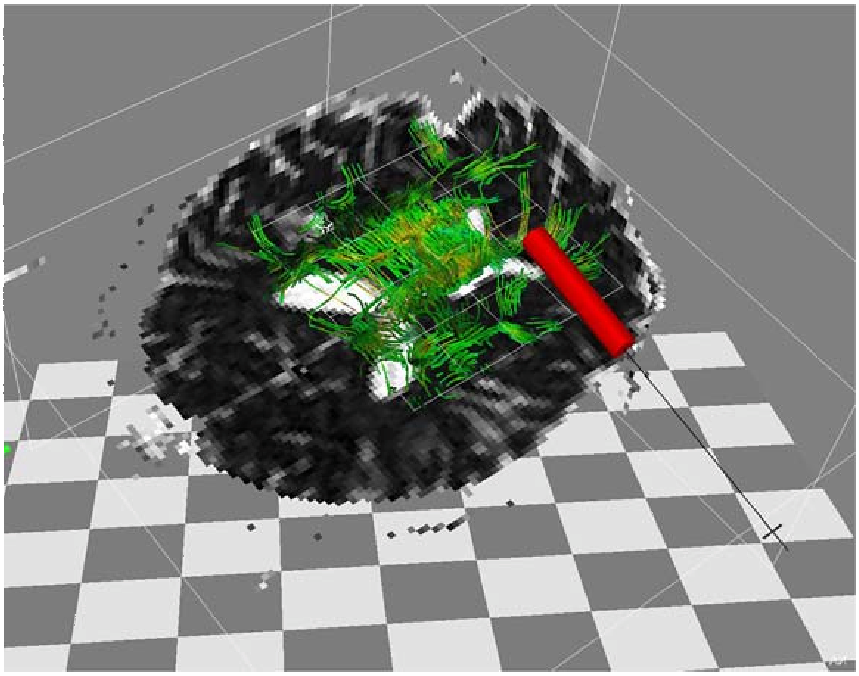

Fig. 10 Diffusion tensor imaging: slice planes and fiber tracking

– *Speech input*: Speech is used to define the interaction and visualization by the standard speech commands. New speech commands are added to enable/disable the different visualization modes with their 3D widgets, and to change the color coding of the visualizations.
– *VTK pipeline browser*: Several visualization parameters can be modified using the VTK pipeline browser.

Manipulating the widgets (e.g. moving a volume with seed points), and controlling the visualization by speech and the pipeline browser will constantly update the visualization.

6.3 Bouncing balls

The bouncing balls application simulates the behavior of moving balls inside a bounding box. Several parameters control the simulation: the number of balls, the radius of the balls, the size of the box, the damping factor of the medium in which the balls move, an attraction force among the balls, the field force that is applied on the balls, the position of each ball, etc. Each step of the simulation computes new positions for the balls and new forces of attraction among the balls. Balls are represented as spheres and small arrows which indicate the direction of the ball movements. The force field around a ball is represented with an isosurface.

Figure 11 shows the application in action. Interaction in this application is provided by:

– *Picking and moving balls*: Balls can interactively be selected and moved to a new position. The simulation will continue computing with the new position of the moved ball.

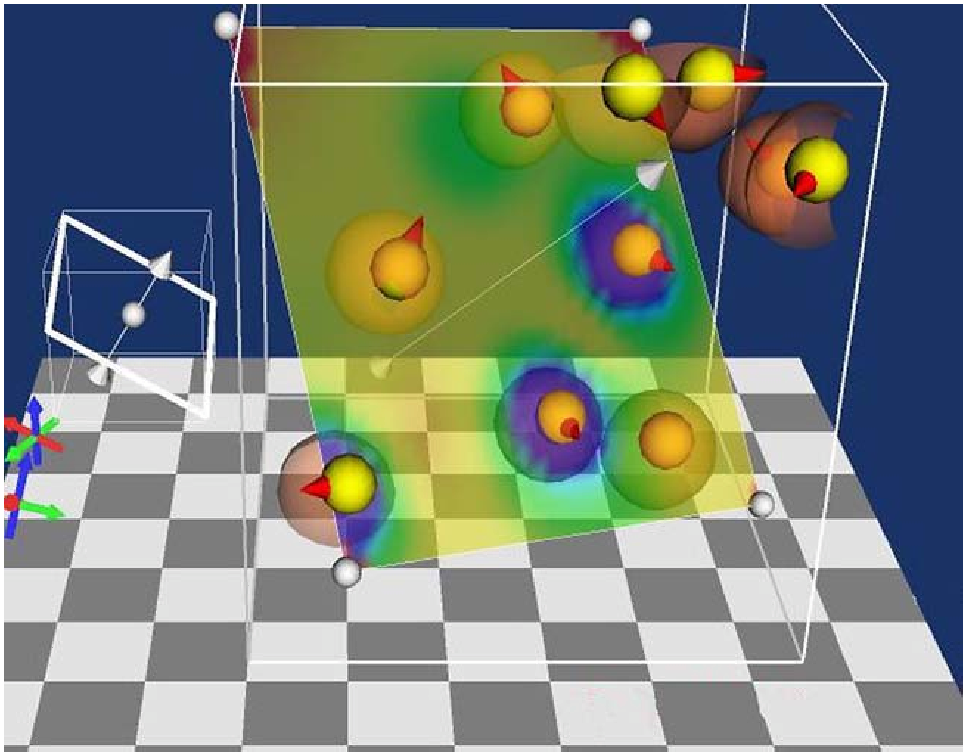

Fig. 11 Bouncing balls simulation

- *Application of a field force*: The white arrow left to the data visualization is an implicit plane widget that controls the direction of the external field source. Rotating the arrow changes the direction of the field.
- *Color mapping of the forces in the data set on a slice plane*: A plane widget controls the position and orientation of the semi-transparent slice plane that visualizes the forces within the cube.
- *Definition of the bounding box*: The bounding box is defined by a box widget. Manipulation of this widget allows for scaling, moving, and rotating the simulation volume.
- *Speech input*: Speech is used to enable/disable the widgets.
- *VTK pipeline browser and VR-VTK UI*: Numeric values for the simulation, such as the number of balls, radius of the balls, the damping factor, etc., are set by 2D widgets on the VR-VTK UI and the VTK pipeline browser.

The user can 'steer' the simulation by interactively manipulating the balls and widgets and/or changing the simulation parameters by speech or the widget cube. The visualization can be controlled using the pipeline browser.

7 Discussion

We presented a new multimodal virtual reality interface to VTK. Main contributions are the addition of real-3D direct manipulation with the visualization and the integration of visualization and system control by different modalities.

The VR interface allows efficient and comfortable direct manipulation of the visualized world in three dimensions. Our 3D cursor regions enable easy selection of objects, and overcome the problems of point selection in 3D. Manipulation of the visualization is provided by the use of 3D widgets. These widgets can be controlled by different modalities: from direct manipulation in 3D using 6 DOF trackers to speech only control.

To overcome problems with spatial positioning in 3D, several features are added: a stage, shadows, and two-handed manipulation. Application and visualization control is provided by the use of speech commands and a 2D widget system with a pipeline browser that is integrated in the virtual world. The pipeline browser offers the user an interface to the VTK objects in the visualization pipeline. We believe that these features are very useful for all interactive visualization systems.

Porting existing VTK applications primarily involves exchanging some of the VTK classes in the application by their VR-VTK versions. The next step is to replace the existing 3D interaction by 3D-widget interaction (as most existing applications do not use 3D-widgets). Furthermore, the interaction style (e.g., cursor region parameters, use of stage, shadows) must be specified. The result is a 3D interactive application, including basic system and visualization control provided by speech input and 2D widgets. To have the application controls, usually provided by a 2D GUI, available in the 3D environment, these extra controls can be implemented as speech commands or as 2D widgets on the VR-VTK UI.

Several existing VTK applications were ported to include the new interface. Porting was easy. It required only slight modifications to the existing VTK code.

Several users have experienced the applications, as described in Sect. 6, with the new VR interface.

Users of the DTI tool have evaluated the VR version of this tool. We asked them to interactively define a region of interest with seed points for the generation of streamlines. Goal is to get a good insight into the data. This is one of the basic tasks in the DTI tool. The choice of seed points is very important: If too many points are chosen, the scene gets cluttered with streamlines, and if the number of seed points is too small, important features of the data can be missed.

The users were very enthusiastic. Several observations were made:

- Direct 3D interaction with the application is much easier than interaction in 2D. It required fewer operations to define the region of interest, and the result was more accurate.
- The world-in-hand option is an essential interaction method to be able to manipulate all handles. However, users constantly want to enable/disable the world-in-hand option. Speech has proven to be very useful for this task.
- Speech is very useful for fast switching between the different interaction (by enabling and disabling widgets) and visualization methods in the applications. The users indicated that they preferred speech (although they had to learn some commands) over 2D menus (as in the standard version of the DTI tool). It was faster, and they could keep their attention at the data visualization.
- Speech needs more feedback. The only feedback a user gets from a speech command is the command being executed. However, speech commands are not always recognized as being valid commands. So, for the user, it is not always clear whether his spoken command was correct or not. The system

needs to present visually that an invalid command has been recognized in the virtual environment.

Apart from the recommendation from the user experiences, several improvements can be made. The use of a parallel version of VTK will make rendering faster, and will therefore improve the exploration abilities of VTK. More complex widgets (although not necessary in our current applications), for example, real two-handed widgets, will directly support more complex interaction tasks. The use of speech to control the 3D widgets, as presented in Sect. 4.3, should be further elaborated and tested.

References

1. Arsenault R, Ware C (2000) Eye-hand co-ordination with force feedback. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI 2000. ACM, New York, pp 408–414
2. Billinghurst M (1998) Put that where? Voice and gesture at the graphics interface. ACM SIGGRAPH Comput Graph 32(4):60–63
3. Boritz J, Booth KS (1997) A study of interactive 3D point location in a computer simulated virtual environment. In: Proceedings of the ACM symposium on virtual reality software and technology, VRST'97. ACM, New York, pp 181–187
4. Bryson S (1996) Virtual reality in scientific visualization. Commun ACM 39(5): 62–71
5. Conner DB, Snibbe SS, Herndon KP, Robbins DC, Zeleznik RC, van Dam A (1992) Three-dimensional widgets. In: Proceedings of the 1992 symposium on interactive 3D graphics, SI3D '92. ACM, New York, pp 183–188
6. Goldberg A, Robson D (1983) Smalltalk-80: the language and its implementation. Addison-Wesley, Boston, MA
7. Herndon KP, Zeleznik RC, Robbins DC, Conner DB, Snibbe SS, van Dam A (1992) Interactive shadows. In: Proceedings of the 5th annual ACM symposium on user interface software and technology, UIST'92. ACM, New York, pp 1–6
8. Hinckley K, Pausch R, Proffitt D (1997) Attention and visual feedback: the bimanual frame of reference. In: Proceedings of the 1997 symposium on interactive 3D graphics, SI3D '97. ACM, New York, pp 121–126
9. Hubona GS, Wheeler PN, Shirah GW, Brandt M (1999) The relative contributions of stereo, lighting, and background scenes in promoting 3d depth visualization. ACM Trans Comput-Hum Interact 6(3):214–242
10. Kok AJF, van Liere R (2004) Co-location and tactile feedback for 2d widget manipulation. In: Proceedings of IEEE virtual reality 2004, VR'04. IEEE Computer Society Press, pp 233–234
11. Krum DM, Omoteso O, Ribarsky W, Starner T, Hodges LF (2002) Evaluation of a multimodal interface for 3D terrain visualization. In: Proceedings of the conference on visualization'02. IEEE Computer Society Press, pp 411–418
12. LaViola JJ (2000) MSVT: A virtual reality-based multimodal scientific visualization tool. In: Proceedings of the third IASTED international conference on computer graphics and imaging, pp 1–7
13. Leigh J, Rajlich PJ, Stein RJ, Johnson AE, DeFanti TA (1998) LIMBO/VTK: A tool for rapid tele-immersive visualization. In: CDROM proceedings of IEEE visualizaton '98
14. Mine MR, Brooks FP Jr, Sequin CH (1997) Moving objects in space: exploiting proprioception in virtual-environment interaction. In: Proceedings of the 24th annual conference on computer graphics and interactive techinques, SIGGRAPH'97. ACM/Addison-Wesley, pp 19–26
15. Mulder JD, van Liere R (2002) The personal space station: bringing interaction within reach. In: Proceedings of the virtual reality international conference, VRIC 2002, pp 73–81
16. Rajlich P, vtkPipeline.tcl. http://brighton.ncsa.uiuc.edu/~prajlich/vtkPipeline

17. Schindler E, Kok AJF, Terken JMB (2005) Evaluation of input modalities for interaction tasks supporting 3D object manipulation. In: Proceedings of the ICMI'05 international workshop on multimodal interaction for the visualization and exploration of scientific data, pp 18–25
18. Schroeder WJ, Martin KM, Lorensen WE (1996) The design and implementation of an object-oriented toolkit for 3D graphics and visualization. In: Proceedings of the 7th conference on visualization'96. IEEE Computer Society Press, pp 93–100
19. Schroeder W, Martin K, Lorensen B (2002) The visualization toolkit, an object-oriented approach to 3D graphics (3rd edn). Kitware Inc., New York
20. Shamonin D, VtkCave. http://staff.science.uva.nl/∼dshamoni/myprojects/VtkCave.html
21. Sharma R, Zeller M, Pavlovic VI, Huang TS, Lo Z, Chu S, Zhao Y, Phillips JC, Schulten K (2000) Speech/gesture interface to a visual-computing environment. IEEE Comput Graph Appl 20(2):29–37
22. Shaw C, Liang J, Green M, Sun Y (1993) Decoupled simulation in virtual reality with the MR toolkit. ACM Trans Inf Syst 11(3):287–317
23. Taylor II RM, Hudson TC, Seeger A, Weber H, Juliano J, Helser AT (2001) VRPN: a device-independent, network-transparent vr peripheral system. In: Proceedings of the ACM symposium on virtual reality software and technology, VRST '01. ACM, New York, pp 55–61
24. van Dam A, Forsberg AS, Laidlaw DH, LaViola JJ, Simpson RM (2000) Immersive VR for scientific visualization: a progress report. IEEE Comput Graph Appl 20(6):26–52
25. van Liere R, Martens JBOS, Kok AJF, van Tienen MHAV (2005) Interacting with molecular structures: user performance versus system complexity. In: Virtual environments 2005, IPT-EGVE 2005. Eurographics Association, pp 147–156
26. van Liere R, Mulder JD (1999) PVR—an architecture for portable vr applications. In: Virtual environments '99, EGVE '99. Eurographics Association, pp 125–135
27. van Reimersdahl T, Kuhlen T, Gerndt T, Henrichs A, Bischof J (2000) ViSTA: a multimodal, platform-independent vr-toolkit based on WTK, VTK, and MPI. In: Proceedings of the 4th international immersive projection technology workshop.
28. Vilanova A, Berenschot G, Pul, CV (2004) DTI visualization with streamsurfaces and evenly-spaced volume seeding. In: Proceedings of the joint Eurographics – IEEE TCVG symposium on visualization, VisSym'04. Eurographics Association, pp 173–182
29. Wanger LR, Ferwerda JA, Greenberg DP (1992) Perceiving spatial relationships in computer-generated images. IEEE Comput Graph Appl 12(3):44–58

Author Biographies

Arjan J. F. Kok is an assistant professor at the Department of Computer Science at the Open University of the Netherlands. He studied Computer Science at the Delft University of Technology, The Netherlands. He received his Ph.D. from the same university. He worked as a Scientist for TNO (Netherlands Organization for Applied Scientific Research) and as assistant professor at the Eindhoven University of Technology before he joined the Open University. His research interests are visualization, virtual reality, and computer graphics.

Robert van Liere studied Computer Science at the Delft University of Technology, the Netherlands. He received his Ph.D. with the thesis "Studies in Interactive Scientific Visualization" at the University of Amsterdam. Since 1985, he has worked at CWI, the Center for Mathematics and Computer Science in Amsterdam in which he is the head of CWI's visualization research group. Since 2004, he holds a part-time position as full professor at the Eindhoven University of Technology. His research interests are in interactive data visualization and virtual reality. He is a member of IEEE.