Device-Independent Navigation and Interaction in Virtual Environments

Chris Faisstnauer, Dieter Schmalstieg, Zsolt Szalavári
Vienna University of Technology, Austria

***Abstract**. We present a new approach to the integration of input devices into virtual environment software systems. Our approach employs a so-called Mapper module as an intermediate between input device drivers and virtual environment application. Major advantages of our approach are full device-independence, including the easy integration of new input devices and emulation of missing device capabilities, interactive reconfiguration, sharing of input devices among multiple applications, automatic selection of devices and interaction appropriate for the task, and high-level support for a large variety of navigation styles in virtual environments.*

1. Introduction

Successful human-computer interaction requires efficient transfer of information from humans to computers. Such communication is mediated via input devices, which have become more diverse with the introduction of virtual reality systems that frequently use 6 degree of freedom (DOF) trackers and other devices that cannot really be considered commodities yet.

While most input devices are conceptually simple (e. g., a digital joystick is nothing more than a set of switches), a lack of standards, especially in 3-D interaction, prevents input devices from being interchangeable. Applications are generally designed to work with a fixed set of input devices. While this allows to exploit such devices to the fullest both in terms of function and performance, it has several disadvantages: The application may present a limited choice of input devices. Support for a new device may require changes in the application, or the application may not be able to fully support the features of the new device (if operated in backward compatibility mode). Also, the application may depend on certain devices to be present and may refuse to operate otherwise. Separation of input device handling and application is necessary to overcome these restrictions.

Beyond general purpose interaction in 2-D and 3-D, virtual environments place additional requirements on the software dealing with user inputs. For a convincing simulation of a virtual world, simple and efficient navigation is very important. Traditionally, navigation was a task of the virtual environment application, that simulated a particular navigation metaphor from raw data obtained from input devices. This usually requires the presence of certain input devices (e. g., 6-DOF tracker), and also leads to re-implementation of the code that simulates the navigation metaphor. A software module that implements a user configurable navigation metaphor may levitate the virtual environment application from simulating navigation and untie it from specific input device hardware.

This paper presents an approach for device-independent navigation and interaction that overcomes the limitations mentioned above. Its defining features are:

- Complete separation of management and querying of input devices from the application. Data coming from the input devices is transformed in a device-independent format that allows applications to process input data of a specific input class (e. g., position) regardless of the source. In this way, new input devices can be integrated without the need to alter the application. It also allows to emulate input behavior in case the most appropriate device is not available (e. g., emulating a mouse with the keyboard).
- A module capable of making educated guesses - called the Mapper - processes the applications' requests for interaction and navigation and automatically selects the most appropriate device and interaction style from the available input devices.
- A navigation module performs the necessary processing for simulating a wide variety of navigation metaphors and delivers high level data to virtual environment applications where it is immediately useful.

Mapper and navigation module encapsulate generalized world knowledge about input devices and navigation metaphors, respectively. In combination, they allow a virtual environment system to specify interaction and navigation requirements on a very high level of abstraction.

2. Related work

While the focus of research work has been laid on specific interaction styles for 3-D and VR applications, little attention has been paid to the issue of device independence. Both experimental (e. g., MR [Shaw93]) and commercial (e. g., DVS [Ghee94]) virtual environment systems try to provide a driver architecture that allows incorporation of new devices, and ship with an extensive list of built-in drivers for common VR devices (tracker, data glove etc.). However, applications are still designed for a particular device setup and cannot operate independent of the available devices.

The only work towards device-independent interaction that we are aware of was presented by He and Kaufman [He93]. Their device unified interface (DUI) pursues the same goals as we do, but aims at general interaction for

3-D systems. Consequently, their approach covers only general purpose 3-D interaction (equivalent to our interaction layer covered in section 5), but lacks an equivalent to automatic selection of input devices and support for navigation metaphors. Instead of a built-in intelligence for the selection of devices, they provide a comprehensive "device information-base" and rely on the application to perform the selection.

3. Overview of the Mapper

The Mapper is a software module with the task to provide applications with input data from the human user, no matter which devices are available at runtime. It does so by employing a set of device-independent data classes, which can be grouped into four layers as outlined below (Figure 2). The Mapper is launched as an independent background process, and opens network connections labeled "logger" and "provider".

To make input devices available to the Mapper, it is necessary to write a device driver that follows a set of specifications for the communication with the Mapper. At runtime, the device driver connects to the "logger", so that the Mapper can query the input device. As part of the connection setup, the device driver transmits a description of the device it manages to the Mapper, which maintains an inventory of all connected devices.

Applications which want to request input data connect to the "provider". Upon connection, the new application is added to the Mapper's application list and can now issue requests to the Mapper. The Mapper selects the most suited input device for request from those devices in the inventory that have not been occupied with previous requests. If the data provided by the input devices does not fulfill the application's requests, the Mapper enhances and processes the raw data in order to construct the data types requested by the application. The Mapper works with user configurable preference lists containing a ranking of input devices in reference to their suitability for specific tasks. Figure 1 shows a schematic view of the Mapper.

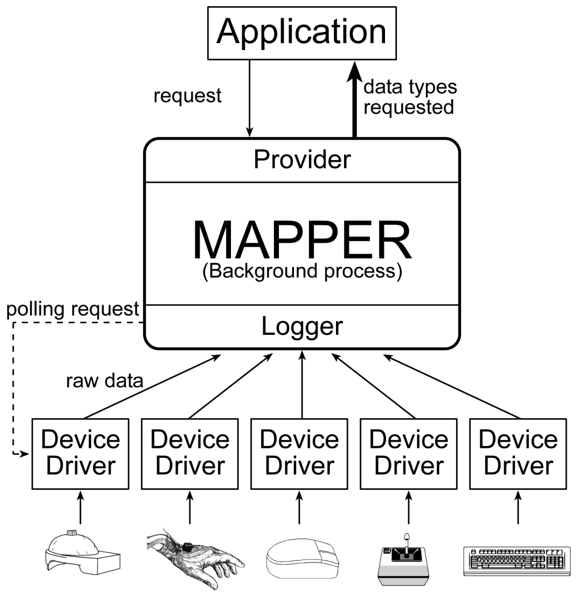

Figure 1: The Mapper provides the application with a unified view of the input devices' capabilities

The selected configuration is communicated back to the application, whose responsibility it is to instruct the user of the device usage, which may be different on workstations with different connected input devices. The set of input devices connected to one particular workstation rarely changes, so that adaptation to a particular usage of a known application is not needlessly compromised. Furthermore, use of a particular device may be enforced by modifying the preference lists.

3.1 Layers

As the Mapper is the only connection from input devices to applications, it must provide every type of data the application could need from the user. Furthermore, to implement specialized types of navigation and interaction not natively provided by the Mapper, the application must also be able to obtain low-level data from the devices. Furthermore, the Mapper should serve a large range of possibly concurrent 2-D and 3-D applications including virtual environments.

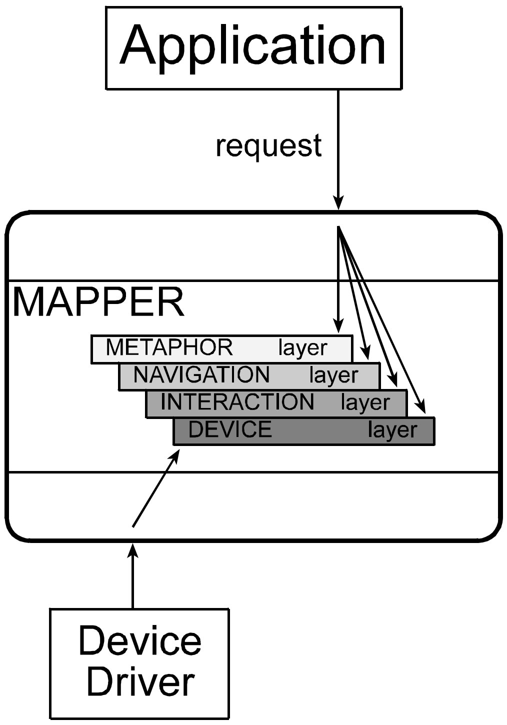

Figure 2: The application may request data on four layers of increasing abstraction

These considerations motivate the use of four layers of abstraction for the requests that may be issued to the Mapper:

1. **Device layer**: This layer delivers raw device data from specific input devices without any processing on the side of the Mapper.
2. **Interaction layer**: This layer supports generic interaction data classes, that allow simplified construction of any type of 2-D or 3-D graphical user interface, such as 3-D selection or choice from a set of options. Intelligent selection of appropriate input devices for the requested interaction task are carried out on this layer.

3. **Navigation layer**: By describing the desired style of navigation (e. g., ground following or flying, constant velocity or acceleration), an application can request the Mapper to interpret the user's input as navigation commands and directly deliver the resulting movement to the application.
4. **Metaphor layer**: As a further step of abstraction, the application can be freed from describing the navigation style in detail, and rather pick from a set of predefined, intuitive navigation metaphors such as "car driving", which cover the majority of navigation needs.

The layers depend hierarchically on each other, so that requests to a higher level layer can be translated into requests on a lower level layer, possibly with some additional processing. The four layers are shown in Figure 2 and described in detail in the following sections.

3.2 Mapping strategy

The task of the Mapper is to match requests for input data on the mentioned layers coming from the application to input devices or device components that are currently available. A brief examination of the problem reveals two fundamentally different strategies to solve this problem:

- An application may request complete information about all currently available input devices (type, DOF, absolute/relative positioning etc.), and then decide on its own. The selection of the device mapping will be completely application specific. While this strategy will always grant the maximum amount of flexibility, it requires that all processing of device selection is implemented for every application. It therefore defeats the goal of achieving true device independence: the application may be relieved from dealing with device specific hardware details, but it still needs to consider trade-offs in specific input devices.
- The Mapper may decide on the device mapping based on a static characterization of each available device, such as DOF, accuracy, number of buttons etc. It therefore completely frees the application from any input device considerations at all. Unfortunately, such an approach fails to encapsulate real-world knowledge that is used by humans to decide the appropriateness of a particular interaction device or style for a particular task. For example, a car simulator is better controlled with a steering wheel than with a mouse, but this fact cannot be represented using a technical classification only.

We therefore decided to implemented a device mapping strategy executed by the Mapper, but driven by preference lists provided by the application or the user. The preference lists define an order of choices regarding devices to use for a particular request. They allow a detailed modeling of suitability of particular devices for particular tasks (the latter being defined by the application's request). Reference lists but do not force applications or users to deal with device selection. In fact, a user satisfied with the default preference list may never need to deal with Mapper configuration at all.

4. Device layer

On the lowest level, the device layer routes raw input device data to the application. No or very little processing is done to the data. The Mapper does not need to know anything about the input device parameters or the purpose of the requested data. This layer is used if applications wants to have direct access to the device, and requires the application to know device-specific details.

A characterization of the input devices (see also [Fole90] and [Burd94]) we used for our implementation is given in Table 1.

Device	Parameters
Analog joystick(*)	Resolution (X/Y), number and type of buttons (front, top, base)
Digital joystick(*)	Number of directions (4 or 8); number and type of buttons (front, top, base)
Mouse	Resolution (X/Y); number and type of buttons (left, middle, right)
Trackball	Resolution (X/Y); number and type of buttons (left, middle, right)
Keyboard	Number of function keys; numerical keypad available?
Tracker	Resolution (for all 6DOF); number and type of units (mounted to head, torso, dominant hand, non-dominant hand); physical setup (offset emitter, offset receivers); if hand-tracker ("flying mouse"): number and type of buttons
Spaceball	Resolution (for all 6DOF); number and type of buttons (left, middle, right)

Table 1: Input devices and their characterizing parameters

(*) Note that multi-joysticks (flight sticks with an extra joystick, e. g., "coolie hat" etc.) are described in this context like a group of multiple separate joysticks.

5. Interaction layer

The interaction layer provides basic interaction classes to support generic human-computer interaction. These data types provide the information typically needed by a wide range of graphical applications to interact with the user.

The data types delivered by the interaction layer and higher layers are device independent and completely free the application from any consideration regarding hardware specific details. The supported interaction data classes (see also [Fole90]) are given in Table 2.

If an interaction class is issued to the Mapper, it consults the preference lists to look up the most appropriate device mapping. We distinguish between devices and device components, the latter being parts of devices like individual buttons or functional groups like the cursor keys. Note that components can overlap in their utilization of a device (e. g., the numerical pad on a keyboard can be used to enter numbers or control a cursor, but not both tasks at the same time). Device components are the unit of assignment handled by the Mapper, in other words, a device can be shared by multiple requests as long as the assigned components do not overlap.

Class and Characterization	Parameters
Confirmation: *Command issued to application*	-
Selection: *Selection from a set of discrete values*	Number of choices
Position: *Input of a 1-D, 2-D or 3-D position*	X-axis: yes/no? Y-axis: yes/no? Z-axis: yes/no? Value range(s) Absolute/relative
Orientation: *Input of a 1-D, 2-D, or 3-D orientation*	Yaw: yes/no? Pitch: yes/no? Roll: yes/no? Value range(s) Absolute/relative
Direction: *Boolean values with directional meaning; used for simple steering without physical movement model*	X-axis: yes/no? Y-axis: yes/no? Z-axis: yes/no?
Quantity: *Abstract numerical value*	Value range Absolute/relative
Velocity: *Special quantity indicating "velocity"; prefers absolute device*	Value range
Acceleration: *Special quantity indicating. "acceleration"; prefers relative device*	Value range

Table 2: Interaction data classes and their description

The Mapper first picks a candidate device from the device preference list, and then consults the device component preference list for that device to find a suitable mapping to device components. Starting from the top of the preference list, the Mapper searches for a component that is both physically available and not already occupied. Once such a selection is made, the Mapper marks the component as occupied and starts delivering input data to the application (for details see the next section).

To make the basic algorithm apt for practical use, a number of additional features can be controlled by the user:

- **Use of modifiers**: It is common user interface design practice to use so-called modifiers to better exploit limited input device resources. By concurrently pressing a key or button, an interaction is modified in its behavior, essentially, the same input device is used for two or more, mutually exclusive tasks. Examples are the use of the shift-key to distinguish cursor navigation vs. selection in word processors, or moving the mouse vs. dragging (with the mouse button pressed). While the use of modifiers slightly increases the cognitive load on the user, it may still be desirable, especially if few devices are available (e. g., in a typical desktop system where the only efficient 2-D input device is a mouse). Consequently, requests to the Mapper can be attributed with up to two modifier keys that may be assigned to the Mapper to reuse a particularly suitable device component that is already in use.
- **Total vs. partial device assignment**: By demanding "total" device assignment, a user may indicate that a particular request does not share a device with other requests even if some components of the device are left unused. This is necessary if one wishes to create a dedicated device to a particular task.
- **Grouping**: Frequently, a task may require multiple channels of input data that belong together semantically. It is therefore good design practice to assign these requests to a single device or a small set of devices. The standard mapping as described above does tend to spread sequential requests over multiple devices according to the most suitable available device. By defining a group of requests rather than a sequence of individual requests, a user may override this behavior and instruct the Mapper to attempt to use as few devices as possible, including the use of modifiers. The latter may be restricted to avoid "overload" of a particular device.

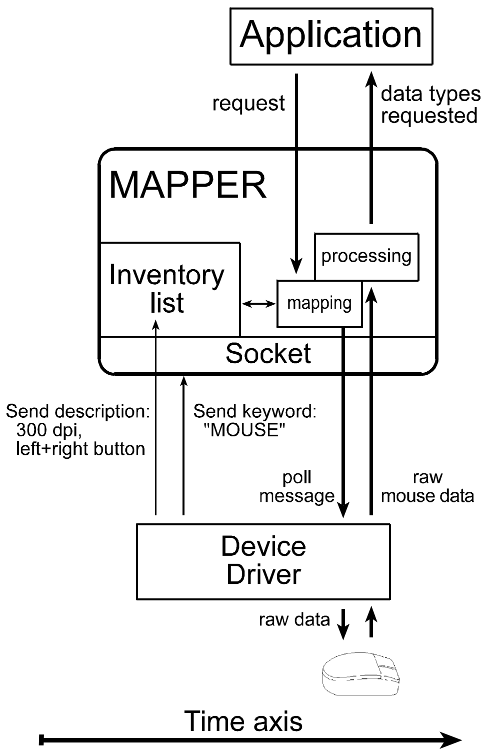

Figure 3: Example of interaction among device driver, Mapper and application requesting data

- **Depth search vs. width search**: Once the Mapper has picked a device, in order to find an available device component it may either prefer to check all devices of the same type if the initial choice is unavailable, or it may stick with the chosen device, stepping through all entries of the device component preference list. This behavior can be selected by specifying width search or depth search, respectively.
- **Absolute vs. relative input**: The application may specify a preference to whether an absolute or a relative input is preferred for a particular request. The Mapper tries to take this preference into account, but is free to pick a relative device for absolute values and vice versa and simulate the requested behavior, if the request cannot otherwise be satisfied.

- **Direct vs. indirect selection**: selection can either be direct (every choice has a 1:1 correspondence to a device component, e. g., a key) or indirect (the choice is made by simulating an array of choices, e. g., by cycling through the choices with a joystick and confirming with a joystick button). Indirect selection is necessary if the set to select from is larger than the available set of suitable device components. A request for selection may allow indirect selection, or it may enforce direct selection, even if the ranking of the chosen device is lower in the preference list.
- **Semi-dependent device components**: Some device components such as the X and Y-axis of a mouse do not allow fully independent manipulation (i. e. when moving the mouse, it is not easily possible to move it strictly along one axis). Therefore a request may be set to *isolating*, meaning that it cannot share a semi-dependent device component with another request.
- **Continuous or polling mode**: The application must indicate whether it wishes to receive data in regular intervals, or if it data is sent at runtime only if the application asks for it. Furthermore, the application can specify whether the Mapper should continuously update its internal representation of the input devices' state independent of the reports to the application.

Figure 3 shows an example of how the Mapper processes a request for interaction by the application and mediates between device driver and application (the same principle applies to higher layers).

6. Navigation layer

The navigation layer supplies a convenient way for an application to control the user's avatar in the virtual environment without having to care about the particularities of navigation. The supported navigation was designed with the intention to cover a large range of navigation styles.

6.1 Design considerations

A fundamental component of virtual environment software is the user's representation (avatar) that can be directed through the simulated space, often controlling the virtual camera that is used to generate the three-dimensional image sequence. However, there are other important issues to navigation that are usually hard-coded in the application, but can freely be specified when using the Mapper:

Structure of avatar: A humanoid avatar suitable for direct control may contain the following body parts: head, torso, left hand, right hand (compare [Robi93, Robi95]). If only one body part is simulated, it may be seen as a combined torso/head, and defines the user's point of view. If head and torso are both present, the torso is taken as the frame of reference (defining the "overall" position of the avatar), while the head is defined in relation to the torso and defines the user's point of view - head and body may be moved independently. Selection of the direction of movement and general interaction can only be made via the line of sight unless a dominant hand (or cursor) is present, which can be moved independently of head and torso. Advanced models include a non-dominant hand as a fourth part; feet are generally omitted. Another dimension of the model is whether the simulated parts are also drawn by the application, so that the user can see his or her own body. Figure 4 illustrates the relationship of the avatar's body parts, a reference coordinate system and a tracking system.

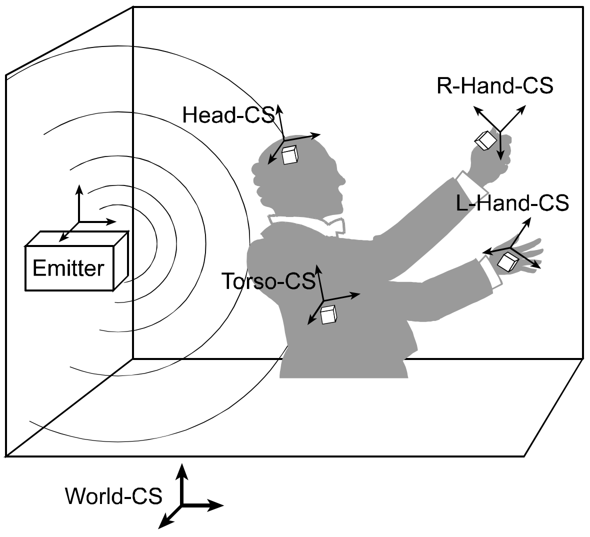

Figure 4: Structure of the avatar with relevant coordinate systems

Point of presence: Navigation is very dependent on how the user relates to the avatar. We speak of 1st person presence, if the user perceives the world "through the eyes" of the avatar. 3rd person presence is established if the user observes and controls the avatar from a viewpoint other than the avatar's. According to this distinction, several models can be distinguished:

1st person models:

- No body: The user directly controls the point of view, and there is no visible representation of the user's body. The appeal of this models is mainly its simplicity.
- Direct control of an avatar with body: The user directly controls an avatar composed of one or multiple body parts and is also able to see these body parts (torso, hands) if gazing down.
- In vehicle, no body: The user is situated in a simulated vehicle such as a car or plane, and controls the viewpoint via simulated cockpit instruments. Body parts are neither simulated nor drawn
- In vehicle, with body: The user is situated in a simulated vehicle, but may see and control body parts. Instruments are operated by employing the simulated hand(s). While the visual effect may be appealing, the double indirection (control simulated hands to control simulated instruments) can deteriorate performance.

3rd person models:

- Body controlled from outside: The body of an avatar is controlled while being viewed from outside. This is a rather unconventional approach more often found in computer games than in immersive virtual environments.
- Vehicle controlled from outside: The user operates a vehicle in a simulated "remote control" model much like a model airplane.

For 3rd person presence to be useful, it must be ensured that the vehicle does not leave the user's field of view. This is usually done by coupling the user's viewpoint to the vehicle (e. g., a camera that trails behind a racing car at a fixed distance), or by using a fixed camera.

Physical setup: For successful implementation of a navigation style, consideration has to be paid to the intended type of physical setup. We distinguish two variations:

- The user is standing and free to walk around (in a limited area). This is generally used for fully immersive virtual environments (using head mounted displays) or augmented reality. As it is inconvenient to use desktop-devices such as mouse or keyboard when standing, this setup assumes the use of 6DOF trackers.
- The user is sitting at a workstation, and is not expected to physically move around. The workstation is regularly equipped with desktop devices, trackers cannot be expected.

Dimension: Navigation may be constrained to a ground plane, effectively walking about in 2-D only, or may allow unconstrained 3-D (flying) movements.

Velocity: Specification of velocity may be given by impulse (either standing still or moving at a preset speed), linear (any speed from a preset range may instantly be selected), acceleration (a simplified physical simulation that requires an acceleration phase to reach the desired speed). Negative velocities (backwards movement) may or may not be allowed.

6.2 Navigation data classes

The application communicates to the Mapper which type of avatar it wishes to implement; the Mapper selects an appropriate control mechanism based on the available input devices, and supplies the application with position and orientation of the avatar's parts. The only task left to the application is to draw the avatar's visual representation to provide visual feedback to the user. The navigation layer relies on the interaction layer to provide basic input data; however, it employs separate navigation preference lists together with rule-based knowledge in order to select the appropriate input devices for the control of the avatar.

Selection of a particular style is done by a number of parameters, the most important of which is the navigation class, which instruct the Mapper on how the user should control the avatar. Choices are: *walking human*, *flying human*, or *vehicle*. The parameters are listed in Table 3.

Walking human: This class simulates a human that is able to walk in a virtual environment following ground level. This model simulates a style of movement very close to our real world experience. The human user physically walk around in a small area bound by the range of the measuring device. Because humans are bound to stay on the ground level, the movement is essentially 2-D (although eye height can be varied). 6 DOF are the preferred input device configuration for the walking human class, as they allow direct correspondence between the user's movements in reality and movement in the virtual environment. This also means that movements are constrained by physical limitations such as tracker range, cable length or walls.

Flying human: This class extends the walking human class to the third dimension. The simulated human may navigate freely in three dimensions. Direct correspondence with physical movement is no longer possible (as humans cannot fly), so the scenario of usage is that the human remains more or less stationary and navigates by indicating direction and velocity of the movement. The movement can also artificially be limited to 2-D, to achieve a ground movement while remaining physically stationary (unlike walking).

Vehicle: The vehicle class simulates traveling in an artificial vehicle. While walking and flying are geared towards "direct" navigation by using ones body in an immersive virtual environment, the vehicle class is introduced for desktop setups, where the user controls a simulated vehicle via a "dashboard" of desktop devices while sitting and looking at a screen. While the control style is quite different to the flying human class, the request parameters are mostly the same. However, the vehicle class allows no choice of direction specification: in 2-D, direction is always specified via heading, in 3-D via heading and pitch.

Navigation class	**Parameters**
Walking human	Body parts: head, torso, left hand, right hand
Flying human	2D or 3D; body parts: head, torso, left hand, right hand; preferred direction specification [Mine95] (head direction, torso direction, hand direction, line from head to hand, line from torso to hand); preferred velocity specification (impulse, linear); maximum velocity; negative velocity allowed?
Vehicle	2D or 3D; body parts: head, left hand, right hand; preferred velocity specification (impulse, linear); maximum velocity; negative velocity allowed?

Table 3: Navigation classes with their characterization

Because they are designed to establish direct correspondence between a user's movements and the navigation in the virtual environment, the walking or flying human model is only supported by the Mapper if 6DOF tracking is available. Each request for a particular body part may be mandatory or optional. For every mandatory part, a tracker must be available. Body parts for which the request is optional may be omitted by the Mapper if too few trackers are available. If the application's requirements cannot be fulfilled or desktop device must be used, the Mapper automatically selects a vehicle metaphor, which may be controlled by desktop devices.

7. Metaphor layer

The possibilities for navigation as provided in the navigation layer are numerous, and most of the capabilities may not be needed for the majority of possible application. To simplify the selection, the metaphor layer offers preconfigured choices of navigation style (metaphors) describing the most useful parameter combinations. Metaphors are well understood by experience or cultural knowledge by most people, and therefore give designer and users alike a good impression about the structure and abilities of the avatar. Some examples like “remote controlled car” or “Superman” are given in Table 4.

Metaphor	**Class**	**Dim.**	**Body**	**Velocity**
Walker	Walking	-	Head, torso, hand	
Skater	Flying	2-D	Head, torso, hand	Linear
Superman	Vehicle	3-D	Head, torso, hand	Linear w/ neg.
Flying carpet	Vehicle	3-D	Head, torso, hand	Impulse w/ neg.
Remote c. car	Vehicle	2-D	-	Linear
Remote c. plane	Vehicle	3-D	-	Linear
Car driver	Vehicle	2-D	Head, hand	
Airplane pilot	Vehicle	3-D	Head, hand	

Table 4: Overview of the metaphors

8. Implementation

The Mapper has been implemented in C++ under Linux running on a PC, while the graphical applications run on an SGI workstation. Connections via Ethernet allow multiple workstations to be serviced by one device PC. This setup It functions as a part of the network infrastructure of the “Remote Rendering Environment” developed by our group [Schm96].

Linux as a platform was chosen for multiple reasons: While polling of the input devices and data processing is a simple task, it occupies a substantial amount of CPU power. Compared to the cost of a workstation-based solution, a dedicated PC is an extremely inexpensive way to add multiprocessing and decoupled simulation [Shaw93] to the system. It also allows the use of PC-based input devices (e. g., joysticks and steering wheels), which are cheap and readily available in a large variety. The communication of Mapper and application is carried out via a local network, which separates the Mapper from any particular graphics platform. The only constraint introduced by this setup is that the Mapper PC must be in physical vicinity to the workstation so that the connected input devices can be used together with the workstation.

The network protocol is based on standard TCP sockets. All communication regarding registration of devices and applications, interaction and navigation request and mapping is done using simple humans-readable ASCII strings. For the delivery of input data from the Mapper to the application, a compact binary format is adopted for reasons of network efficiency. Each binary data packet is headed by a tag. The tag allows the receiving application to identify which of its requests the packet belongs to and therefore conclude how to decode the binary format.

9. Evaluation

To illustrate the operation of the Mapper, we conducted a number of simple experiments.

Experiment 1: Control of one or more vehicles. We choose this task to demonstrate the Mapper’s response to interaction requests, and how multiple successive requests for interaction (multiple tanks) are satisfied. Simple movement of a tank is controlled by specifying a heading (1-D orientation) and absolute velocity, both constrained by an interval. The command issued to the Mapper is

```
ORIENTANGLE ABS H -180 180 +
             VELOCITY -10 10
```

In the configuration we tested, a joystick was available and chosen as the most appropriate device for the request:

- Heading was mapped to the X-axis of the joystick
- Velocity was mapped to the Y-axis of the joystick

Using the upper, front, and finally both buttons of the joystick as modifiers, the Mapper is able to satisfy up to four requests for concurrent tanks, running as independent applications. In other words, a single joystick controls four tanks at once (Figure 5). A fifth tank would be controlled using the keyboard.

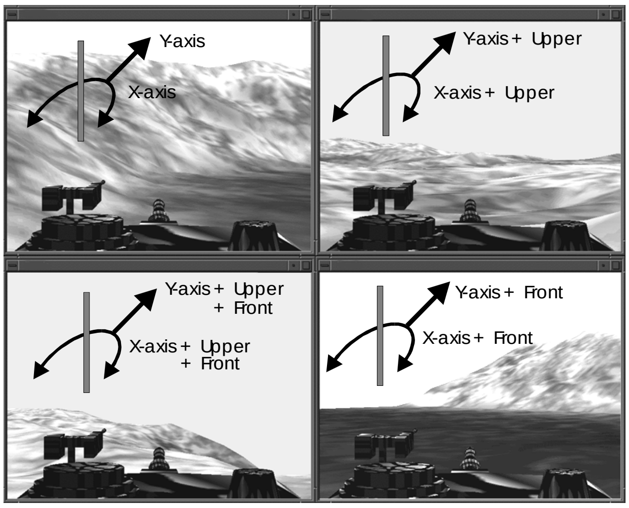

Figure 5: Controlling four tanks simultaneously with one mouse and button combinations

Experiment 2: **Navigation with different device configurations**. The requested navigation is aimed for exploring a virtual environment: A flying human in 2-D (note that terrain following is provided by the application), consisting of head, torso, and an optional right hand (to be omitted if no tracker available). Travel direction should be specified by flying in crosshair direction (from head to right hand: HEADRHAND, see Figure 6) at variable velocity (max. speed: 10) determined by distance between head and hand. The requested navigation is accompanied by an interaction request to select among 10 objects and use the currently selected object (confirm). Issued requests are

```
NAVIGATION FLYING 2D HEAD! TORSO!
           RHAND VEL 10 HEADRHAND
SELECTION 10 + CONFIRM
```

At the first attempt, trackers for head, torso and hand (flying mouse with 1 top and 3 front buttons), mouse and keyboard were available. The resulting mappings:

- Head, torso, hand controlled by head, torso, hand tracker, respectively
- Movement triggered by flying mouse button 1 (top button)
- Selection cycled by flying mouse buttons 3 and 4 (front buttons)
- Confirm triggered by flying mouse button 2 (front button)

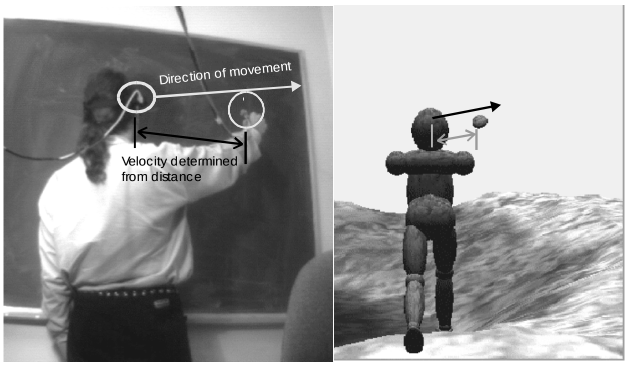

Figure 6: Cross-hair specification of the direction is used for navigation using a head-mounted display and tracker.

The second attempt tried to demonstrate the emulation behavior when only desktop devices are present. Consequently, only mouse and keyboard were made available to the Mapper. As a result, the Mapper changes the navigation style from “flying human” to “vehicle” (the torso coordinate system becomes the vehicle coordinate system) and the optionally requested right hand is omitted. The resulting mappings:

- Vehicle heading controlled by mouse X-axis
- Vehicle velocity controlled by mouse Y-axis
- Head heading controlled by mouse X-axis + left button
- Head pitch controlled by mouse Y-axis + left button
- Selection controlled by middle and right button
- Confirm is controlled by keyboard (space-key)

10. Conclusion

We have presented an architecture aimed at achieving independence of virtual environment applications from particular input devices. This is realized by introducing a separate component, the Mapper, that is capable of supplying the application with the desired user input processed to represent interaction and navigation. An algorithm to select appropriate devices based on the applications’ requirements and the presently available input devices frees the application from any concerns about devices. Our experimental implementation verifies that this approach is feasible and eases the development of virtual environment software.

Acknowledgments. The authors would like to thank M. Eduard Gröller for proofreading and Anton Fuhrmann for assistance with the hardware setup. This work was sponsored by the Austrian Science Foundation (FWF) under contract no. P-11392-MAT.

Further information and software for download can be obtained from our web site

http://www.cg.tuwien.ac.at/research/vr/mapper/

References

[Burd94] G. Burdea, P. Coiffet: Virtual Reality Techology. John Wiley & Sons (1994)

[Fole90] J. Foley, A. van Dam, S. Feiner, J. Hughes: Computer Graphics: Principles and Practice. Addison-Wesley Publishing Co. (1990)

[Ghee94] S. Ghee, J. Naughton-Green: Programming Virtual Worlds. SIGGRAPH’94 Course, No. 17 (1994)

[He93] T. He, A. Kaufman: Virtual Input Devices for 3D Systems. Proc. IEEE Visualization’93, pp. 142-148 (1993)

[Mine95] M. Mine: Virtual Environment Interaction Techniques. SIGGRAPH’95 Course, No. 8 (1995)

[Robi93] W. Robinett, R. Holloway: Implementation of Flying, Scaling, and Grabbing in Virtual Worlds. SIGGRAPH’93 Course, No. 43, pp. 6.1 - 6.4 (1993)

[Robi95] W. Robbinet, R. Holloway: The Visual Display Transformation for Virtual Reality. Presence, Vol. 4, No. 1, pp. 1-23 (1995)

[Schm96] D. Schmalstieg, M. Gervautz, P. Stieglecker: Optimizing Communication in Distributed Virtual Environments by Specialized Protocols. Virtual Environments and Scientific Visualization‘96 (ed. M. Göbel), Springer (1996).

[Shaw93] C. Shaw, M. Green, J. Liang, Y. Sun: Decoupled simulation in virtual reality with the MR toolkit. ACM Transactions on Information Systems, Vol. 11, No. 3, pp. 287-317 (1993)