Exploiting Distant Pointing Gestures for Object Selection in a Virtual Environment

Marc Erich Latoschik and Ipke Wachsmuth

Technische Fakultät, Universität Bielefeld,
Postfach 100131, D-33501 Bielefeld, Germany
e-mail: {marcl,ipke}@TechFak.Uni-Bielefeld.DE

Abstract. Developing state of the art multimedia applications nowadays calls for the use of sophisticated visualisation and immersion techniques, commonly referenced as Virtual Reality. While Virtual Reality meanwhile reaches good results both in image quality and in fast user feedback using parallel computation techniques, the methods for interacting with these systems need to be improved. In this paper we introduce a multimedia application that uses a gesture-driven interface and, secondly, the architecture for an expandable gesture recognition system. After different gesture types for interaction in a virtual environment are discussed with respect to a required functionality, the implementation of a specific gesture detection module for distant pointing recognition is described, and the whole system design is tested for its task adequacy.

1 Introduction

When moving on from the two-dimensional window-based environments to large-screen multimedia and Virtual Reality settings, the classical WIMP interface turns out insufficient. In a Virtual Reality environment, mouse and keyboard are obviously limited and restrict users when acting in large-screen display environments like a projection wall, a *CAVE* [3] or in front of a *Responsive Workbench* [6]. For many users it would be desirable not to be governed by discrete hardware devices to express their intentions, but to use the natural modalities like gesture and speech. The main interaction types that have to be accomplished in VR can be separated in two areas: navigation and manipulation. There are a couple of input devices [10] that adapt the two-dimensional input methods to a virtual environment by adding at least one or more degrees of freedom. But the principal type of interaction with the system does not change. E.g.: VR systems often use a space-mouse, the three-dimensional counterpart of the 2D mouse, that adds the possibility to manipulate an object's 3D position and orientation. Like in WIMP based systems such an object can be as well a cursor or a pointer. In contrast to its counterpart on a 2D computer desktop, it is now located in the virtual space to enable the selection of items, objects or menus, again using the windows- and icon-based interaction metaphors.

Based on early ideas from the *Put that there System* [2] and on our own research results [7] we want to extend human-machine interfaces by speech-supported gestural input and, in particular, integrate such an interface in a multimedia application, a Virtual Reality construction system. The current paper focuses on the design of a virtual environment for gesture-driven interaction and on the evaluation and detection of distant pointing gestures for selecting objects and locations.

Along this path, the whole evaluation and development task can be separated in two levels. On the conceptual level we describe the scenario and the application, pointing out what kind of interaction should be accomplished. Further, we examine gestures more closely to find out how they can be classified and which gesture types are appropriate to build a new interface upon. Since we are actually constructing a running system, we also take a closer look at the type of computational problem raised by gesture detection. This leads us to the second, namely the technical, level. The problem we are attacking here is to develop a system architecture that is capable of dealing with a number of different constraints. Building an interface for VR, real-time and low latency have the first priority to achieve both, an instant system feedback and a guaranteed frame-rate to avoid user discomfort from lags in the display stream. Finally, the system architecture should be expandable for a gradual approach of our goals by way of incremental prototyping.

In the following sections we introduce a multimedia application that makes use of enhanced immersion techniques and an agent-based architecture for developing an expandable multi-modal interface for interacting with that system[1]. Our first functional goal for this interface is to establish gesture-driven methods for selecting objects and locations, described in section 3. Design and implementation details are given in section 4.

2 The Virtual Application Environment

The testbed for our work, the multimedia backend, is the Virtual Constructor, VC (cf. [13]), a system for assembling and disassembling virtual objects and aggregates. This application uses mouse and typed text input as interaction methods. Using the mouse for a specific action, the user first has to trigger the action-type with a WIMP-style tool-bar click. In contrast to that, with a typed input, the user can express the semantics of an intended interaction with appropriate verbs, e.g.: '...*connect* the grey bar with the yellow bolt...' or '...*detach* the left wheel...'. Selecting parts is done in both ways, objects can be referenced and manipulated with commands, e.g. '...turn the blue square bar ...' or with a mouse-click that enables virtual handles around the selected object, which then can be manipulated by a mouse operation on these handles. There are different manipulators for rotating about the three main axes and for moving. E.g.: dragging and dropping of objects, that means selecting and moving them in the virtual scene until they reach a user-desired location, is done by dragging the translation handle of one part. If a collision between the graphical representation

[1] This work is embedded in the SGIM project (Speech and Gesture Interfaces for Multimedia) and is partly supported by the Federal State of North-Rhine Westfalia in the 'Virtual Knowledge Factory (Virtuelle Wissensfabrik)' joint project effort.

of two approaching objects is detected during such a dragging action, and both objects have matching ports at their collision area, the two objects are virtually connected to form a new aggregate.

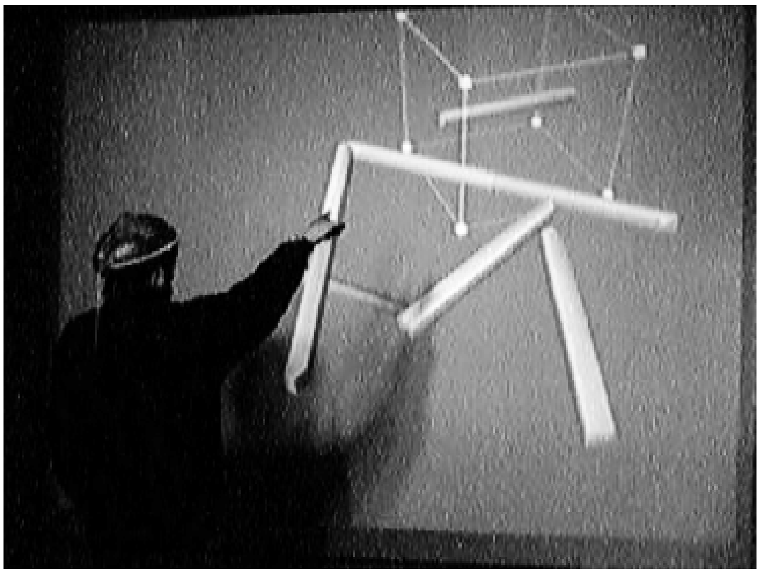

Fig. 1. Interacting with the Virtual Constructor in front of the projection wall: User pointing to select an object and enable the manipulation handles.

From this short specification and the given examples, the minimal necessary interface needs to include possibilities to select objects for moving and manipulating. Fig. 1 illustrates this type of interaction; it shows a user standing in front of a 2m x 3m projection wall while pointing for selecting the geometric representation of a virtual bar. Such bars are the basic parts for the project's overall construction target: The mobile platform 'citymobil' (Fig. 2), a small vehicle for inner city transportation of up to two persons.

To give an overview of the whole virtual working scene, Fig. 2 and Fig. 3 visualise a side-view of the real vehicles basic parts and a virtual model of the basic frame. The application task is the construction of a complete citymobil using basic virtual building parts like pipes, square bars or bolts.

3 Pointing in the Context of Gestural Interaction

In this section we discuss a functional classification of gestures which groups similar body expressions and evaluates the usefulness of these groups for building

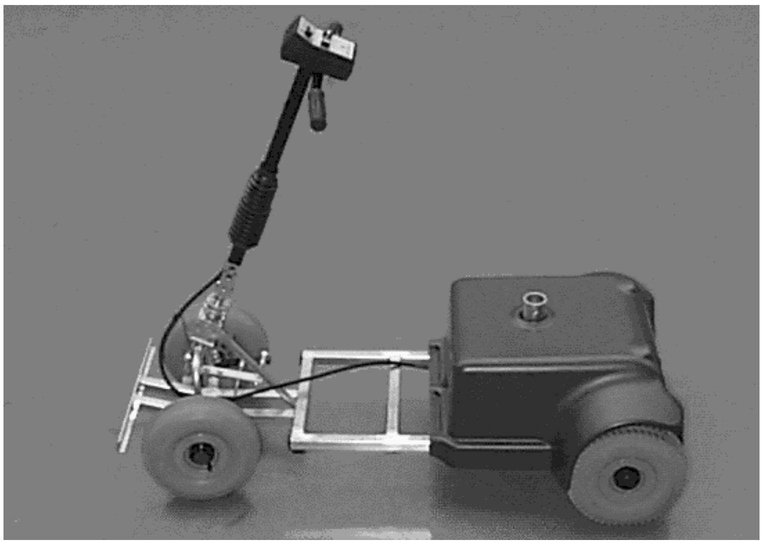

Fig. 2. Citymobil side-view of the real vehicle (shown without seat and hood).

a gesture-based interface. We already emphasised the usage of distant pointing for spatial referencing objects and locations and we explicitly add this as a type of deictic gesturing to a classification that sorts different gesture types according to their functional role in a communicational act. Therefore, in contrast to work in [4] and [5] we follow [8] and [14] and incorporate deictic gestures as an independent class. Table 1 represents this classification scheme.

Being aware of the fact that distant pointing can be very ambiguous, we plan to include further the user's gaze, as a hint to the part of the scene representation that is in the users current focus, thereby reducing the number of possibly referenced objects/locations. This spatial orientation of the head and the eyes belongs to the same group of deictic gestures because our view often focuses on a referenced object during communication about a specific item or location. Additionally, gestures belonging to group II, mimetic gestures, seem to be useful for interaction in a three-dimensional environment. When the upper limbs are used as place-markers for objects, changes in their location and orientation can be reflected in the application by an analogous location or orientation change of a selected object or group of objects. Next, almost all human-machine interfaces have a fixed set of commands to trigger specific actions. These commands can be obviously coded with symbolic gestures of type IV, e.g. special hand postures to shut the system down or, in our specific scenario, to toggle between assemble and disassemble mode. In the following section we describe the implementation for an expandable gesture-driven virtual reality interface that makes use of deictic gestures, according to table 1 type I, by evaluating distant pointing.

Type	Classification name	Description
I	**1.** deictic	Used as a spatial reference to places and objects. E.g.: Pointing to a chair and asking: ,Can you please get that chair and put it here?'
II	**1.** mimetic **2.** iconic **3.** object-related	Using the upper limbs as place-markers for actions or behaviours of objects or states. E.g.: Hitting both fists together when speaking about a car crash.
III	**1.** physiographic **2.** kinetographic **3.** pantomimic	To show how to use something. Acting without the required objects. E.g.: Demonstration of hammering with imaginary tools.
IV	**1.** symbolic **2.** mode-setting **3.** emblematic	Contain a unique meaning. Change the mode in which a verbal statement has to be interpreted. E.g.: Pinkie and thumb build a ring ⇒ OK. Hand rests parallel to the ground and starts shaking. ⇒ Speaker is not quite sure about her statement.
V	**1.** ideographic **2.** metaphoric **3.** iconic	Represent a spatial metaphorical manifestation of an internal state. Relate close to an interpretation. E.g.: Somebody says that he feels dizzy and turns the index finger in the air.
VI	**1.** beats **2.** gesticulation **3.** speech marking **4.** self regulating	Supply a speech rhythm. Accentuation and gestural expression occur at the same time. E.g.: Speaker emphasises and outlines important parts of her speech by hitting the index finger on a surface.

Table 1. Gesture types grouped according to their functional role (adapted from various sources; see text)

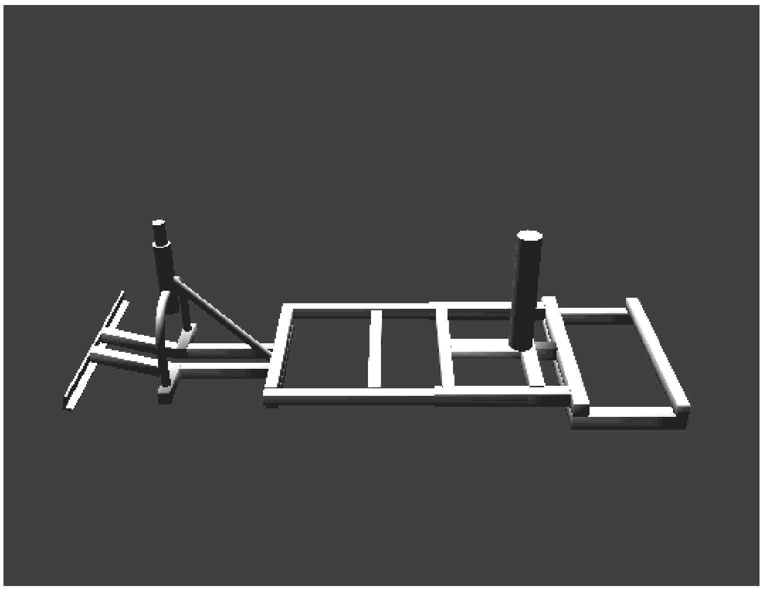

Fig. 3. Citymobil: structure of the virtual basic frame

4 Design and Implementation

4.1 Constraints for the Application

The system design has to take three important restrictions into account that are all related to the computational cost of the different tasks. Starting with the first one, detecting gestures is closely connected to pattern matching problems like speech analysis. In both cases a stream of discrete sensor data might embed a sense-carrying symbol (a specific gesture like a point-to) that we want the system to recognise. In contrast to a trackball or a mouse, there is no unique 1:1 relationship between user input and the desired system reaction. Furthermore, [10] pointed out that gestures are high-level, mapping directly to user intent. Therefore the interpretation is ambiguous, leaving us with the need to take other sources of information into account, i.e., verbal user input and knowledge about the situation context to resolve this equivocalness. Hence from a computational point of view this process is very time-consuming. Our task is to connect such an interface to a VR visualisation system, thus real-time capability has the first priority. To guarantee a fixed frame rate, avoiding user sickness from lags and unsteadiness in the display stream, the rendering loop, the second-expensive computational part, cannot be altered by other time-consuming analysis. On the other hand the backend application itself also embeds functionality that demands huge amounts of processing time. Therefore it is again desirable to have this module separated from the rendering and detection. Hence we have to distribute the whole system, according to the specific single-modules computational requirement, on independent processors. As a result, the next task is to establish the minimal optimised communication interface between all parts, avoiding to gain computational power in exchange for a higher communication overhead.

4.2 Distributed System Architecture

We pointed out that from the engineer's point of view a loose coupling between the modules gesture detection, application, and rendering is the main directive for the connection (cf. Fig. 4). In our approach the visualisation is accomplished through a Performer[2]-based viewer whereas the detection is done by a hierarchical multi-process agent-based system. Because our prototype uses only a minimal subset of the applications facilities, the backend and the rendering module can be hosted by one workstation, keeping in mind that the final system version will need more computational power and will have to be separated on different processors.

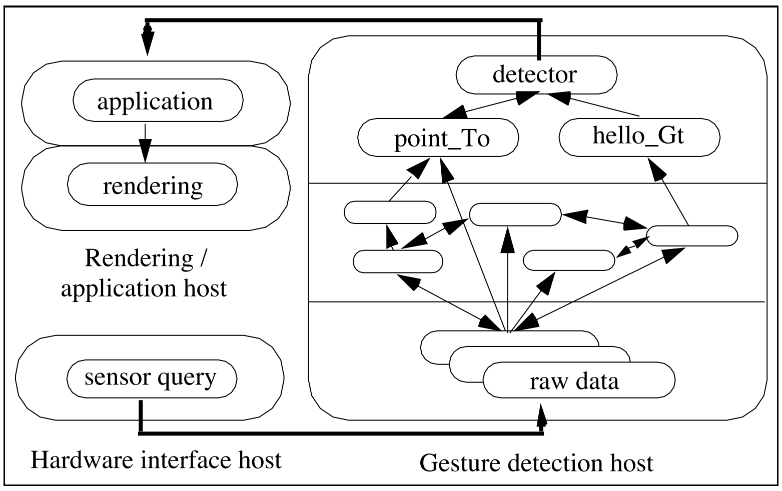

Fig. 4. Distributed system design with agent-based gesture recognition module

IPC (Inter-process Communication) [11] capabilities of different kinds establish the communicational transport layers between rendering and detection as well as between the different layers of the gesture recognition and the application itself. Because separation on different workstations is desirable, we use, based on [9], UNIX/PC sockets and ICE[3] for the communication transport layer. This

[2] Performer is a Silicon Graphics toolkit for 3D rendering of virtual scenes that offers real time and multi-process options.

[3] ICE: INTARC Communication Environment.

enables us to run all tasks - detection, rendering and application - on different hardware platforms, making the rendering process almost independent from the analysis and recognition.

4.3 Agents for Detecting Gestures

To deal with the pattern matching problem involved in gesture recognition (see section 4.1) , different approaches for a computational solution are possible. One is the use of neural networks, e.g.[12] and [1], where the significant classification vectors of gestures are found during a training phase before the actual system use. But in contrast to that, taking into account the type of gestures we want to recognise, we can explicitly describe the significant features of a gesture (see section 4.4). Therefore we follow ideas for a feature detection like in [14], and we define gestures by their describing attributes based on the gestural time-spatial expression. Fig. 4 illustrates the coupling between the distributed system and the internal structure of the gesture detection. In the latter the different tasks are split into separate processes or agents. For instance, a process on the highest level evaluates significant features that describe a pointing gesture, like the elongation of the index finger combined with a rest in the acceleration of the forearm. This information is distributed and delivered from processes on lower levels that are specialised on the detection of the described pointing gesture key features. Therefore the hierarchical level on which such a process resides in the detection module is a raw measurement for the level of semantic feature integration it has to accomplish. On the base level, input agents analyse the data stream from Data Glove sensors for the hand postures and five 6DOF sensors for the relative position and orientation of specific upper limb parts, the head, both elbows and both hand wrists. These data records are distributed via a network multicast mechanism to all workstations in our domain and the low level processes receive, calibrate, integrate and finally broadcast the data again via fast IPC communication (shared memory) to the next higher levels. Due to this mainly number crunching task, combined with the fact that these processes do not embody any semantic attributes like an agent for finger-elongation detection does, we call them light-weight agents in contrast to the higher-level agents, e.g. for elongation or pointing detection, both from types like we describe in the following section. As illustrated in Fig. 4 the dataflow is not only established from bottom to top, but the information is distributed to any other agent that can make use of the results. One example for this cooperation is the calculation of dynamic attributes, which in fact uses results from agents detecting static features. This leads to the desired system scalability with a minimal overhead of equal multiple calculations. We have to keep in mind: Our goal -in later steps- is to additionally evaluate the users point of view and human voice input as further sources of information. In the next section, we introduce and examine two definitions for pointing gestures based on descriptive higher predicates. These gesture definitions are expressed in a simple rule system that evaluates the agent's results and triggers an action.

4.4 Defining a Pointing Gesture

Recognizable gestures are defined by quantified feature predicates and attributes of specific bodyparts. Analysing the posture sequence of a pointing gesture (see Fig. 6), we can use the following definition:

```
Point_To := Elongating(Index) AND
            Elongate (Index, Min(threshI)) AND
            Elongate (Middle, Max(threshM)) AND
            Elongate (Ring, Max(threshR)) AND
            Elongate (Pinky, Max(threshP))
         => Action(Select Vector)

Where threshX is the threshold for the given
attribute and finger X
```

Fig. 5. Simple definition of a pointing gesture

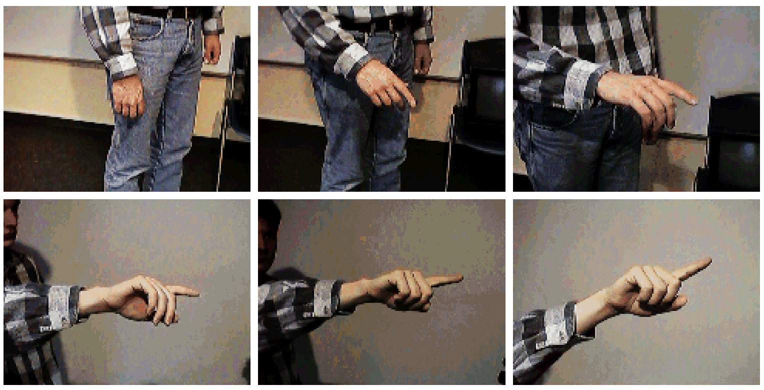

Fig. 6. Sequence of postures during a dynamic pointing gesture

This rule is reflected by one high-level agent (Point_To) receiving input from six different sub-agents, each representing one of the predicates Elongate(X,tx) and Elongating(Y). After completing the rule and integrating all data into one discrete event, the action part, in this specific case the selection of the object/location pointed to, is triggered. Because this rule is evaluated from left to right, there is an intrinsic order of importance for each attribute. If, for example, the index finger would not be elongating, the other attributes would be ignored. A more elaborated version of defining a pointing gesture in both, the expressiveness and in the computational cost, would take all attributes into account and weight them according to their significance (cf. Fig.7).

```
Weight_Point_To = w(El_I) * Elongating(Index)
                + w(E_I)  * Elongate (Index, Min(threshI))
                + w(E_M)  * Elongate (Middle, Max(threshM))
                + w(E_R)  * Elongate (Ring, Max(threshR))
                + w(E_P)  * Elongate (Pinky, Max(threshP))

if (Weight_Point_To > thresh_point)
                => Action(Assert_Hypo(Point_To,eval(Weight_Point_To)))


where w(X) are the weights and thresh_point the overall threshold
```

Fig. 7. Definition of a pointing gesture with weighted attributes

The weight-attributed approach (Fig. 7) for describing a pointing gesture supports another feature we are currently working on. Instead of triggering a specific action, a hypothesis should be generated for assertion into an integration module. This is the first step to deal with competing other high-level gesture detection agents, stressing the fact that different gestures do not have to be exclusive at any time. This fact is illustrated in Fig. 6. During the first three phases in pointing, there is also a rotational part about the hand-wrist, letting both kinds of gestures occur simultaneously for a specific time-frame. Since this is dynamic, we have to examine both gestures in parallel until the whole pointing or rotating phase ends or reaches a certain threshold. Then we can find a final solution and we can decide which gesture did, and which one did not occur. If we would detect one gesture exclusively based on a key-feature, we might have to back-track when this attribute was detected wrongly. But information about the possible gestures can be worthy during the detection phase and before the gesture reaches its climax. We mention this fact with reference to the outlook when we discuss information integration with different modalities, because detecting them has a need for additional information about possibly detected gestural expressions.

4.5 Assigning Workload to Agents

Now we have to take a closer look at the tasks a single agent has to work on. We already emphasised the importance for separating the three main modules on different processors. On a smaller scale we must also concentrate on the internal structure of the agent system. How can we divide the computational work into parts that justify the communication overhead[4] between the agents? Are there

[4] running on a single processor there will always be an overhead for switching the process-context between agent processes and enabling a communication

any tasks with results that can be shared? Is there a need to further distribute the whole agent system to separate processors? To find an answer to these questions we evaluated three different organisation structures with both gesture definitions Fig. 5 and Fig. 7, running the application and detection module based on each of the definitions on a single processor, and separating application and detection based on Fig. 7 on different hosts.
A rough worst case estimation[5] of the computational costs for the pointing gesture detection based on raw glove data comes to the following result: Evaluating one set of data takes approximately 300 mathematical and data-dereference floating point operations. Using a detection rate of 100Hz there are about 30.000 operations/sec. Compared to the possible 5.000.000 operations/sec[6] a standard workstation nowadays is capable to compute, this illustrates that there is no direct need for splitting these operations. But in contrast to that, running both the detection and the rendering on one processor decreased the frame-rate about one third when the detection was done based on the definition given in Fig. 7. Using the definition for point_to shown in Fig. 5, no significant negative influence could be recognised. As a result the detection module is only split in separate processes where the same data is used by more than one higher-level process.

5 Conclusion and Outlook

In this article we gave a description of an expandable gesture recognition system developed for a distributed multimedia application. A module for the scene visualisation, the so-called viewer, as well as a module for detecting distant pointing gestures have been implemented from scratch using the underlying architecture, showing satisfying results. Based on this work, our system is currently able to pick out, from a continuous gesture stream, significant key features; to (1) classify them as belonging to a pointing gesture and (2) to identify the direction pointed to. The rendering loop works with a minimal frame-rate around 20 frames/sec and the pointing recognition works with a very low response latency, making it appropriate for time-critical VR applications. Furthermore, the Virtual Constructor backend has been adapted to cooperate with these modules. Communication methods for both, host-to-host and interprocess information exchange have been evaluated and integrated into the system. As an overall result of a performance test, our expectation about the advantages of a distributed architecture was confirmed. Further work has to be done on the lower scale for structuring the gesture detection agent system. In addition to that we have started to integrate dynamic manipulation of selected objects, like transformation and rotation, by way of mimetic gestures. A further step is the integration of speech input, e.g., to issue '*...connect...*' or '*...disconnect...*' commands for

[5] This estimation does not compare computational time on a 'used processor cycles' basis, but illustrates the differences in order of magnitudes

[6] These results were gained on a SGI Indigo2 Workstation with a MIPS 4400 running on 175MHz and 96MB RAM. The language was C++ and no compiler optimisation was enabled

selected aggregate objects. Finally, the conceptual and technical work we have accomplished so far readily allows to include two-handed gestures; these could be very helpful for, e.g., stretching or shrinking of selected objects.

References

1. K. Böhm, W. Broll, and M. Sokolewicz. Dynamic gesture recognition using neural networks; a fundament for advanced interaction construction. In S. Fisher, J. Merrit, and M. Bolan, editors, *Stereoscopic Displays and Virtual Reality Systems, SPIE Conference Electronic Imaging Science & Technology*, volume 2177, San Jose, USA, 1994.
2. R. A. Bolt. Put-that-there: Voice and gesture at the graphics interface. In *ACM SIGGRAPH—Computer Graphics*, New York, 1980. ACM Press.
3. C. Cruz-Neira, D.J. Sandin, and T.A. DeFanti. Surround-screen projection based virtual reality: The design and implementation of the cave. In *Computer Graphics Proceedings, Annual Conference Series 1993*, pages 135–142. ACM SIGGRAPH, 1993.
4. D. Efron. *Gesture and Environments.* King's Crown Press, Morningside Hights, New York, 1941.
5. A. Kendon. Current issues in the study of gestures. In J.-L. Nespopulous, P. Rerron, and A.R. Lecours, editors, *The Biological Foundations of Gestures: Motor and Semiotic Aspects.* Lawrence Erlbaum Associates, Hillsday N.J., 1986.
6. W. Krüger, C.A. Bohn, B. Fröhlich, H. Schüth, W. Strauss, and G. Wesche. The responsive workbench: A virtual work environment. *IEEE Computer*, 28(7), 1995.
7. B. Lenzmann and I. Wachsmuth. Eine multimodale Eingabearchitektur. In M. Thielscher and S.-E. Bornscheuer, editors, *Fortschritte der Künstlichen Intelligenz / Aus den Workshops der 20. Jahrestagung für Künstliche Intelligenz*, page 93. Dresden University Press, 1996.
8. D. McNeill. *Hand and Mind: What Gestures Reveal about Thought.* University of Chicago Press, Chicago, 1992.
9. I. Schnüll and L. Franzen. Evaluation des Kommunikationstools ICE und Integration in das Multiagentensystem VIENA. KI-NRW Report 96/04, Universität Bielefeld, Technische Fakultät, 1996.
10. S. Sherr. *Input Devices.* Academic Press, 1988.
11. R. W. Stevens. *Advanced Programming in the UNIX Environment.* Addison-Wesley, Reading, MA, USA, 1992.
12. K. Väänänen and K. Böhm. Gesture driven interaction as a human factor in virtual environments - an approach with neural networks. In *Virtual Reality Systems, conference proceedings.* British Computer Society, Academic Press, 1992.
13. I. Wachsmuth and B. Jung. Dynamic conceptualization in a mechanical-object assembly environment. *Artificial Intelligence Review*, 10(3-4):345–368, 1996.
14. A. D. Wexelblat. A feature-based approach to continous-gesture analysis. Master's thesis, Massachusetts Institute of Technology, Advanced Human Interface Group, Cambridge, MA, USA, june 1994.