Virtuelle Werkstatt: A Platform for Multimodal Assembly in VR

P.Biermann, B. Jung, M. Latoschik, I. Wachsmuth
Laboratory for Artificial Intelligence and Virtual Reality
University of Bielefeld
http://www.techfak.uni-bielefeld.de/ags/wbski/

Abstract: In this paper we describe ongoing research that aims at the development of a generic demonstration platform for virtual prototype modeling by utilizing multimodal – speech and gesture – interactions in Virtual Reality. Particularly, we concentrate on two aspects. First, a knowledge-based approach for assembling CAD-based parts in VR is introduced. This includes a system to generate meta-information from geometric models as well as accompanying task-level algorithms for virtual assembly. Second, a framework for modeling multimodal interaction using gesture and speech is presented that facilitates its generic adaptation to scene-graph-based applications. The chosen decomposition of the required core modules is exemplified by an example of a typical object rotation interaction.

Keywords: *Virtual Reality, Virtual Assembly, Multimodal Interaction, Interaction Decomposition, Task-Level Interfaces*

1 Introduction

The project "Virtuelle Werkstatt" aims to combine and improve research results in multimodal human-computer interaction and immersive CAD systems to develop a generic demonstration platform for Virtual Reality (VR) based prototyping. Multimodal interaction is concerned with the processing of gesture and speech based user input to drive the modifications of a 3D-visualized scene. Immersive CAD is concerned with the design, exploration and evaluation of virtual prototypes. The intention of our project is to provide a facility for a natural, mutimodal communication in a virtual construction scenario, which is similar to the way two humans would interact with speech and gestures. To enable natural multimodal interaction in virtual environments, results from different research communities have to be considered: If speech recognition as an already active research field is put aside, multimodal interaction can be realized by combining VR with the advances of Artificial Intelligence (AI). The integration of multimodal interaction in existing state-of-the-art VR-principles should hide the specific internal functional details and should provide a form of reusable interface components.

The project is carried out in a newly built VR-system consisting of a 3-sided Cave, projected by 6 D-ILA projectors and using passive stereo via circular polarization filters. The gestures of the user are tracked with a marker-based infrared camera system. For precise hand-posture tracking we use two wireless Data-Gloves. The sound set-up is an eight channel system with loudspeakers in each corner of the cave. The computer cluster for application and rendering is an 'Artabel Fleye 160' Linux cluster, including 5 server nodes (double Pentium III-Class PCs) and 8 graphic nodes (single Pentium IV-Class PCs) with NVIDIA GeForce 3 graphic boards. The nodes are connected via a 2GBit/s Myrinet network for distributed OpenGL rendering.

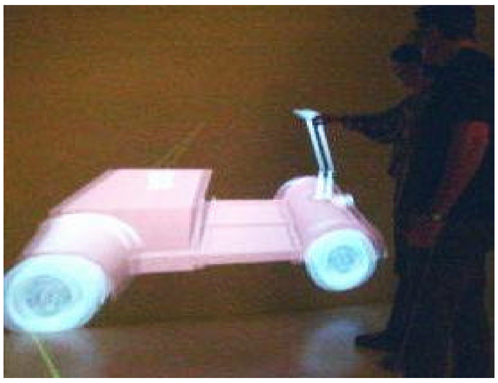


Figure 1: A "Citymobile" vehicle in a VR-demo setting

The first part of this paper presents the notion of port-concepts and connection properties of assembly objects and how they can be extracted from purely geometrical descriptions. It also shows the representation of these connection properties in a knowledge base and task-level algorithms for assembling and disassembling aggregates and adjusting existing connections. The second part deals with a framework for multimodal interaction that handles integration of detected gesture and speech via a temporal augmented transition network (tATN) integration scheme and the application adaptation in form of scene graph components.

2 Assembly Simulation for Interactive Virtual Prototype Modeling

2.1 A Knowledge-Based Approach for Representing Connection Properties

To interact with and assemble components in virtual reality, it is necessary to augment the plain CAD-models with information about their connection properties. A conceptual model of part matings has been developed that involves connection-sensitive part subvolumes or *ports* as well as constraints imposed by different kinds of connections [Jung98]. Supplementing conventional lower-level CAD descriptions, port representations provide a more abstract modeling means closer to the human conceptualization while still providing enough detail for virtual reality simulation of various assembly-related operations. Knowledge bases of predefined port and connection concepts capture the most common mating properties of CAD-based parts, to facilitate the reuse of once developed models.

The port knowledge base consists of a taxonomy of port concepts which is organized around geometric properties, which differentiate the port concepts at the higher level and mechanical properties which distinguish the concepts at the lower levels. At the top-level, the taxonomy is divided in the following concepts:

- *Extrusion ports* model connection properties of object subvolumes with extrusion geometries. For example, extrusion ports model connection properties of objects involved in peg-in-hole type assembly operations. In general, connections between Extrusion Ports afford one translational and one rotational degree of freedom.
- *Plane ports* model connection properties of planar object surfaces. Connections between two plane ports afford two translational and one rotational degree of freedom.
- *Point ports* model point-like object connections that induce no translational degrees and up to three rotational degrees of freedom when objects are connected.

Overall, the port taxonomy currently consists of over 20 concepts which have proven useful in several applications. The connection knowledge base defines a set of useful connection concepts such as ‘screw’, ‘insert’ or ‘glue’. The taxonomy of connections is based on the work of Roth [Rot94] and uses a variation of his “Freiheitsmatrix” representation to model the kinematic constraints that restrict the relative movement of two connected parts.

```
<PortDef object="SCREW">
   <port type="OneSidedThreadedShaft"
name="shaft">
        <connectability type="screw"/>
        <capacity begin="8" end="-8"/>
   </port>
</PortDef>
```

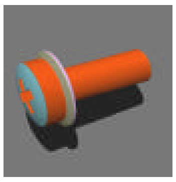


Figure 2: Port definition for a CAD-based part.

The predefined port and connection concepts capture stereotypical mating properties of CAD-based parts. Specific CAD-based parts are annotated with descriptions that reference the predefined port concepts and, by inheritance, can access all kinds of knowledge contained therein. Figure 2 illustrates a minimalistic annotated CAD model of a screw whose shaft is modeled as an instance of OneSidedThreadedShaft (a special kind of extrusion port). Optional properties of port descriptions include hotspots, which describe preferred positions and orientations of a port when being connected to another port, more detailed geometry descriptions, and scaling information. The next section describes a method for automatic generation of port descriptions from CAD-objects of which only polygonal geometry models are available.

The port representations contain sufficient information to determine if two ports are connected to each other in an assembly situation. To this end, formal mating conditions are specified that define the legal relative placements of two connected ports. As connections between all port-types afford some degrees of freedom on the relative positioning of the ports, the mating conditions do not fully determine but rather only constrain the relative placement of ports in object connections. The mating conditions allow the testing whether two parts are connected in the virtual environment. During the simulation of assembly processes they are also used to bring two parts into a connected state.

Assembly simulation in virtual prototyping environments generally not only involves the two parts to be connected but also third parts with which the two parts being mated must not interfere. Non-interference with other parts is a further requirement which is dealt with by the simulation algorithms described in section 2.3.

2.2 Extracting Port Knowledge from CAD-Data

One way to author the port descriptions associated with a CAD part is to model them by hand (supported by a visual editor). As a more comfortable alternative, a method for automatic recognition of port features has been developed [Bie00]. Since there is no standard description of shapes (features) in polygonal CAD-Models, the main information about the model, which can be reliably obtained, is the polygonal boundary representation in form of single triangles. Since there is often no guarantee that this polygonal shape forms a mathematical valid boundary representation that holds the 2-manifold property [Män88], robust heuristic methods were developed to gain as much information as possible about the shape of the components and their parts. The main idea is to separate the object boundary in several clusters, which contain adjacent triangles, whereas their neighbors form an obtuse angle. This simple approach generates clusters, which in most instances match an intuitive division in generic shapes. Figure 3 shows the clusters of a simple screw. If this approach fails to generate the desired division, the user can intervene to select special clustering methods for cylinders or planes.

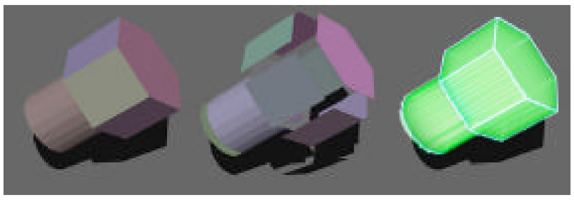


Figure 3: A screw from an inventor file, an exploded view after the clustering, and a visualization of the voxelization.

Several features of these clusters are computed: the dimensions and directions of the oriented bounding box, the total area, the average of plane normals and the average direction of cross products between normals of adjacent triangles. These features can be used to distinguish between several generic shapes. Figure 4 shows a set of geometric primitives and gives an overview of their specific attributes. A second source of information is a volumetric representation, which can be computed based on the cluster data.

The representation is an extension to the Spatial-Occupancy Enumeration [FDFH96], where the voxel can take four different values: inside, outside, on the boundary, and on the edge of two connected clusters. This data is gained by moving a test-voxel through all grid positions of the bounding box and test for a collision with one (which results in a border-voxel) or more (which results in an edge-voxel) clusters. In the right part of Figure 3 a visualization of a voxelized screw is shown (voxel on the edges are shown in white). These voxels are used to verify the hypothesis of shapes, which is gained from the feature analysis, but also to gather further information, for example, about the peculiarity of cylinders, which could border a hole or a bolt, or to get information about the accessibility of the detected ports.

In the example of the screw the system detects the shaft as a cylindrical port (bolt), which is accessible from one side. Furthermore the system recognizes the head as a polygonal extrusion, the disk, and the plane at the two ends of the screw.

This information is utilized to define the ports of the component and it is possible to generate a constructive solid geometry representation, which is needed in order to scale parts of the object and for inhomogeneous scaling of complex objects. For example, the user can scale the shaft of a screw instead of scaling the whole screw or lengthen a ledge without deforming its holes (see Section 2.4). It is also possible to generate linguistic terms such as "longish" or "round" that describe the shape of objects or object parts. This could help to resolve references to objects or object parts in the instructions of the user.

All this information of object parts, colors, shapes, linguistic descriptions, and scaling information is stored, together with a polygonal representation, in a hierarchical database which can be accessed from semantic entity nodes in the scene graph (see Section 3).

2.3 Task-Level Algorithms for Assembly Simulation

The preferred level of interaction with virtual environments is the task-level [Zel91, ZG96]. Ideally, there should be a 1:1 correspondence between the tasks performed by a human in the real and in the virtual world. Aiming at a task-level instructability of virtual prototyping environments, a set of algorithms has been developed for transforming high-level specifications of various assembly-related tasks into corresponding changes to the lower-level graphics representations of the assembly scene. These task-level algorithms implement the logical interaction with the virtual prototyping environment, i.e. they are inherently independent of particular user interfaces, supporting both manipulation- and com-

Geometry	OBB	Voxel data	Normals	Area	Cross products
			$\tilde{N} = \sum_{\substack{all\\triangles\,i}} \vec{N}_i \cdot A_i$	$A_{total} = \sum_{\substack{all\\triangles\ :i}} A_i$	$\tilde{Z} = \frac{\sum_{\substack{all\_adjacent\\planes\,a,b}} N_a \times N_b}{num of.cross products}$
Plane	Main length of OBB is eq.0	Edge voxel define shape	$\frac{\lvert\tilde{N}\rvert}{A_{total}} \approx 1$		
Disk	Main length eq. 0 Other lengths eq. 2 * r	All voxel on circle are edge voxel	$\frac{\lvert\tilde{N}\rvert}{A_{total}} \approx 1$	$A_{total} = p \cdot r^2$	
Rectangle	Main length eq. 0 Other lengths eq. width(w), and length(l)	All voxel on edge of OBB are edge voxel	$\frac{\lvert\tilde{N}\rvert}{A_{total}} \approx 1$	$A_{total} = w \cdot l$	
Cylinder	Main dir. || Axis Main length eq. height (h) Other eq. 2 * r	Border voxel on cylinder hull; Information about accessibility	$\frac{\lvert\tilde{N}\rvert}{A_{total}} \approx 0$	$A_{total} = 2 \cdot p \cdot r \cdot h$	$\tilde{Z} \approx 1$
Cone (acute-angled)	Main dir. || Axis Main length eq. length Other eq. 2 * rmax	Border voxel on cone hull	$\frac{\lvert\tilde{N}\rvert}{A_{total}} \approx 0.1 - 0.3$	$A_{total} = p \cdot r\sqrt{r^2 \cdot h^2}$ (for pointed cones)	$\tilde{Z} \approx 0.8 - 0.95$ $\tilde{Z} \parallel \tilde{N}$
Sphere	All lengths are equal	Border voxel on hull	$\frac{\lvert\tilde{N}\rvert}{A_{total}} \approx 0$	$A_{total} = 2 \cdot p \cdot r \cdot h$	$\tilde{Z} \approx 0$

Figure 4: Some geometric primitives and their features.

mand-based interaction styles.

For example, the task-level algorithm for connecting two parts or aggregates in the virtual environment performs the following steps:

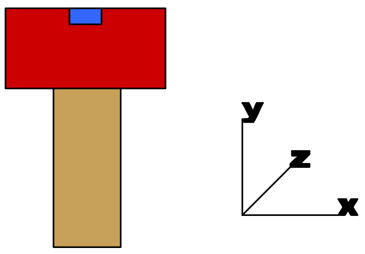
1. The input descriptions of the parts to be mated are refined by selecting connectable ports and hotspots (that define the target position for snapping) on each of the ports.
2. Preliminary mate: The parts are repositioned in the graphics scene as suggested by the now fully filled input descriptions. Already consumed capacities on the ports (resulting from already existing connections) are also accounted for, which possibly require modification of the target position for the mate.
3. Collision avoidance: The mated parts are tested for (polygon-precision) collision with third parts. If such collision occurs, the transformed part's position is altered in small increments until no collision occurs.
4. Update of the port and connection representations: The resulting assembly is checked for new connections resulting from the assembly step. Typically, exactly one new connection is produced. It is however also possible that multiple connections result from the assembly step, or that – if in every position where the mating conditions hold also a collision with third parts occurs – no new connection can be established. In the latter case the assembly operation has failed. The port representations are updated such that their capacity descriptions reflect the new connections.

Similar task-level algorithms have been developed for the disassembly of parts and assemblies as well as adjustment operations that modify the relative position or orientation of two parts of the same assembly constrained by the remaining degrees of freedom of their connection. Common features of all task-level algorithms include:

- They exploit knowledge about ports, their connections, and part geometry. They do not require deep physical models of part behavior (although such knowledge could be added if desired).
- They expect only high-level information as input that can be extracted with reasonable effort from virtual reality interactions based on speech, gesture, or direct manipulation.
- They incorporate various snapping strategies to compensate for incomplete or vague input.
- They integrate collision detection and avoidance to produce physically plausible assembly states.
- They are computationally inexpensive and thus suitable for real-time virtual environments.

2.4 Parametric Parts and Scaling Operations

In the "Virtuelle Werkstatt", we not only consider virtual assemblies of rigid bodies, but also of parametric parts that allow an interactive scaling in the virtual environment. Since most scaling operations of complex objects cannot be expressed as a homogenous scaling of the whole object, the system has to provide some information about how to scale parts of an object.



```
<Part Name="Screw" PolyRep="screw.iv"
        Center="(0,0,0)" ScaleCenter="(0,0,0)" CSG="C+">
    <ScaleModes>
        Scale-Y -> Children-Y
        Scale-XZ -> Children-XZ
    </ScaleModes>
    <SubPart Name="Shaft" GenRep="<cylinder diam=2 len=8>"
            Center="(0,-4,0)" ScaleCenter="(0,0,0)" CSG="+">
        <ScaleModes>
            Parent-Y -> Scale-Y
            Parent-XZ -> Scale-XZ
            Scale-Y
            Scale-XZ
        </ScaleModes>
    <\SubPart>
    <SubPart Name="Head" PolyRep="head.iv"
            Center="(0,-4,0)" ScaleCenter="(0,0,0)" CSG="C+">
        <ScaleModes>
            Parent-Y -> None
            Scale-Y -> Children-Y
            Scale-XZ -> Children-XZ
        </ScaleModes>
        <SubPart Name="Body" GenRep="<cylinder diam=4 len=2>"
                Center="(0,1,0)" ScaleCenter="(0,0,0)" CSG="+">
            <ScaleModes>
                Parent-Y -> Scale-Y
                Parent-XZ -> Scale-XZ
            </ScaleModes>
        <\SubPart>
        <SubPart Name="Slot" GenRep="<box x=.5 y=.5 z=4>"
                Center="(0,1.75,0)" ScaleCenter="(0,2,0)" CSG="-">
            <ScaleModes>
                Parent-Y -> Move-Y
                Parent-XZ -> Scale-Z
                Scale-Y
                Scale-X
            </ScaleModes>
        </SubPart>
    </SubPart>
</Part>
```

Figure 5: XML-Description of scaling and CSG information of a screw

For example if the user wants to lengthen a screw, only the shaft of the screw should be scaled. Therefore the scaling operation – scale in Y-direction – cannot be performed on the geometry of the whole screw, but the screw has to be divided in two individually scaleable parts: the head and the shaft. The scaling operation is forwarded to these subparts. The shaft scales with a given factor, and the head in this example does not scale at all. If the part contains holes or depressions which must not be deformed when scaling the object, the object has to be represented as a CSG-tree with complementary parts.

Also the user can select subparts of the object for individual scaling, for example to broaden the slot of the screw by scaling the complementary CSG-part "slot" (see Figure 5) in X-direction. Figure 5 shows an example of an XML-description which specifies the dependencies of the scaling operations of the parts of a screw. These dependencies are listed in the <ScaleMode> section:

'Scale-Y -> Children-Y' means the scaling in Y-direction is done by inhomogeneous scaling of the child-parts that define their scaling behavior as 'Parent-Y -> ....' in this case. 'Scale-Y': scaling of the whole part in Y-direction with respect to the scaling centre.

'Scale-XZ': scaling of the whole part in X- and Z-direction with the same factor.

'Move-Y': translate the part in Y-direction instead of scaling it.

The move operation is needed if parts have to be moved as result of a scaling operation of the parent-part. For example if the cylinder which describes the head of the screw is elongated, the slot has to be moved in Y-direction to remain on top of the head. The granularity of the partitioning into subparts depends on the scaling operation. A scaling of the shaft of the screw in our example only requires a division into head and shaft, whereas the scaling of the head requires also a division of the head into body and slot.

For visualization of CSG components two different techniques are used. During the interactive inhomogeneous scaling we need real-time rendering of the changing CSG-tree representation. The computation of the object surface for every step in the interaction is not possible in real time, but a real-time visualization can be archived by an OpenGL-implementation of the Goldfeather-Algorithm [Goldf86]. This algorithm generates the visual effect of CSG operations, without generating the surface itself, by using the OpenGL stencil buffer [Ake95] to mask out the parts of the CSG primitives which are not visible.

At the end of the interaction a polygonal representation has to be computed to reduce the complexity of multi-pass rendering using stencil buffer operations and to enable collision detection, which depends on a polygonal representation of the objects. We use the ACIS modeler to compute a representation in form of primitive triangles, which can be inserted as a geometry-node in the scene graph in order to replace the interactive scalable CSG-representation.

3 Multimodal Interaction

Interaction with and in large-screen virtual environments is an interesting research topic of its own. In the work described here, we follow approaches from, e.g., Cavazza et al [Cav95] and Lucente [Luc98] as well as from our own projects, e.g., in [Wac97] and [Lat98], that dealt with the incorporation of multimodal – speech and gesture – input as interaction media for computer graphics and VR applications. The goal is to overcome deficiencies related to WIMP (windows, icons, menu, pointer) interfaces in VR-setups. The main idea is that there should not be any need for specialized input devices besides the users' natural modalities. Having a closer look at multimodal interaction, we find that the related problems can be classified in the following four categories:

- Gesture detection and analysis
- Speech recognition and interpretation
- Speech and gesture integration
- Application adaptation

In this section we focus on the progress we have made concerning the last two of the areas (including the interpretation step), namely the integration formalism currently under exploration and the chosen scene graph oriented functional decomposition for the required real-time 3D application adaptation, which is the virtual assembly scenario described above. An extensive survey about the gesture processing system can be found in [Lat01b].

3.1 Integration and Interaction Specification

There have been several approaches to multimodal integration in computer graphics. The first project usually referred to, is the famous "Put-that-there" by Bolt [Bol80]. In this system, the time when a pointing gesture had to occur was predefined with respect to specific words from speech. This type of speech-driven approach was followed by several other researchers but is nowadays neglected in favor of two more general integration schemes. The earlier ones of those can be categorized as frame-based, e.g., in [Koo93] or [Vo96]. More recent work often utilizes unification algorithms to handle the integration task, Johnston [Joh98] being a prominent example of that. Although the latter offers a clean declarative way to predefine multimodal correlation, even Johnston notices its computational complexity and recently turned to a finite-state-based integration [Joh00]. This approach has some similarities to the *temporal ATN's* (augmented transition network) [Lat01a], which builds the core of our own multimodal integration module; an example of a tATN branch currently used is depicted in Figure 6. What differentiates a tATN from its counterpart – a common ATN – will be clarified in the following section.

The nodes in Figure 6, e.g., `(B1)` or `(R1)`, depict the different states the tATN can reach during the traversal. The connecting arcs are labeled with constraints implemented either by complex test functions like in "`(Rotate)`" `(B1>R1)` or by tests for lexical atoms like in "`about`" `(R2>R31)`.The complex test functions can be (1) sub-tATNs for parsing reoccurring phrases, e.g. "`(ObjDesc1)`" branches to a sub-tATN for parsing noun phrases, (2) gesture test functions like "`is?(rotating)`" at the traversal `(R41>R42)` or (3) test functions for application internal states.

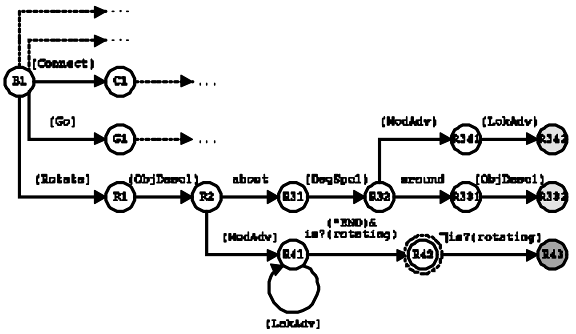


Figure 6: A fragment from one temporal ATN currently employed. The lower branch parses combined gesture and speech utterances for specifying desired object rotations.

A typical example for the latter – which is not exemplified in the above figure – is a test function for querying if an object is currently selected. `(R42)` is a special state that represents a switch to a pure gesture driven application state. This special interaction mode is called **continuous interaction** in contrast to **discrete interactions** which is performed instantly when reaching certain end states, denoted by the nodes colored in light grey. The end states for continuous interactions are depicted in dark grey. For example, if `(R42)` is active, the continuous mode can only be left by stopping the previously started rotation gesture to reach `(R43)`. Typical gesture/speech utterances handled by this tATN branch are, e.g., "Rotate [pointing gesture] this thing about 30 degrees like [rotation gesture] this to the right" or "Rotate the yellow wheel like [rotation gesture] this". Figure 7 shows an ongoing continuous rotation interaction which is processed by the outlaid tATN branch and initiated by the second utterance example.

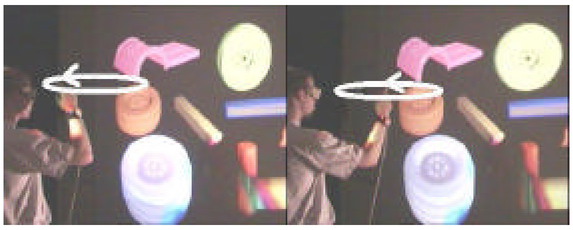


Figure 7: A sample continuous interaction. The tATN from Figure 6 is at state (R41). The gesture processing takes over the application control due to the negated is?(rotating) constraint at arc (R41>R42). The scene shows a snapshot from an earlier prototype system running on a single large screen display (wall) with an electromagnetic tracking system

The already introduced predicate "`is?()`" leads to the difference between an ATN and its temporal version. In addition to the former, the temporal ATN enables and supports the processing of parallel occurring events by labeling all states with time-stamps. For each transition function there is a special time-register called `reach` and an appropriate `reach.set` function for the register value. `reach.set` is currently predefined in the following way: If a state `(S2)` is reached by an arc from state `(S1)` that has a lexical constraint, `S2.reach` will be set to the beginning of the lexical units utterance. If `(S2)` is only reached by a test for an internal application state, `S2.reach` will be set to `S1.reach.` For example, `is?(rotating)` at `(R41>R42)` checks if a rotation type gesture is currently uttered at the time when state `(R41)` has been reached. To clarify what happens if a gesture test function leads to `(S2)`, we will now explain how the results of the gesture detection are represented here at the integration level. Though the gesture processing is latched into the rendering loop, e.g., to determine spatial relationships between deictic gestures and the displayed scene, its rate is independent from the image generation by using parallel computation methods. This is to ensure undisturbed input processing under varying frame rate conditions. Synchronization between the gesture module and the integration and application is done via so called **attribute sequences,** containers which hold the time stamped results of the underlying gesture detection networks. For each frame, there can be an arbitrary number of values in one specific attribute sequence, depending on the ratio between the gesture processing rate (currently 100Hz) and the actual frame rate. Using this buffering scheme, the `reach` time stamps at the different states allow testing for temporal relations with the time stamped values in the attribute sequences. In contrast to approaches that handle gestures as atomic events with a specific occurrence in time, the attribute sequences enable a more general view because they allow gestures to have an interval type temporal progression. This is particularly important regarding so-called *mimetic* or *kinemimic* gestures, gestures which are particularly useful regarding continuous interactions. These gesture types communicate or pantomime an action by executing during a longer time period (see Figure 7).

3.2 Application Adaptation

To handle multimodal interaction in VR setups, special care was taken about the functional decomposition of needed modules. All components are scene graph oriented and follow ideas from Open Inventor, VRML and X3D to incorporate specialized objects for input processing into the graph structure. To allow a scene graph based application to make use of a multimodal interaction interface, special node types have been developed to handle (1) gesture processing, (2) gesture analysis, (3) interaction intermediation and (4) speech-gesture integration and interpretation using the tATN. Regarding the latter, it is important to emphasize that the interpretation step must access knowledge about the currently rendered scene and the respective user position, orientation and his/her view of the scene to understand deictic utterances, e.g. "…the left wheel…" and hence is naturally to be coupled into the hierarchical scene description. On the other hand, interpretation of multimodal utterances profits from concepts developed in the area of language understanding which are often realized using artificial intelligence (AI) methods. This resulting incorporation of AI and VR principles greatly influenced the following approach. As an example for the chosen decomposition, a closer look at the implementation of the already introduced continuous interactions follows. From the plain geometric view, human gestures are rarely performed in a perfect manner. If a user wants to describe, e.g., a desired object rotation by circling his or her hand (see Figure 7), the resulting movement will be imprecise regarding its trajectory. But the desired object change should – in the general case – reflect the perfect circle regarding its path and it should move in appropriate speeds to allow fine grain interaction control. To achieve this in a scene graph oriented design, three specialized node-types are introduced, starting with the so called **actuators.** (see Figure 9). Actuators encapsulate movement data which is significant for a user's utterances, e.g., the fingertips of the index fingers, the angle between upper arm and forearm or certain directions like the view direction, palm normal vectors or pointing directions. As can be seen from Figure 8, the data exchange from the actuators is done via the already introduced attribute sequences (in Figure 9 labeled ***FM(t)***).

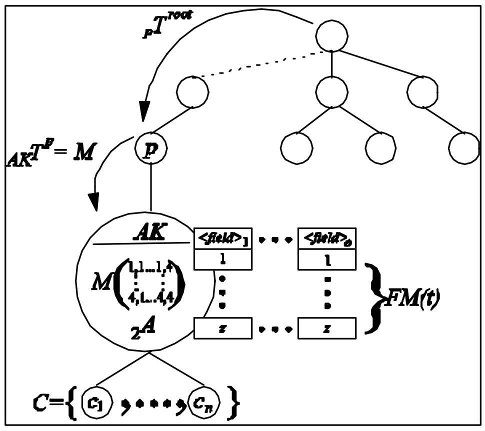
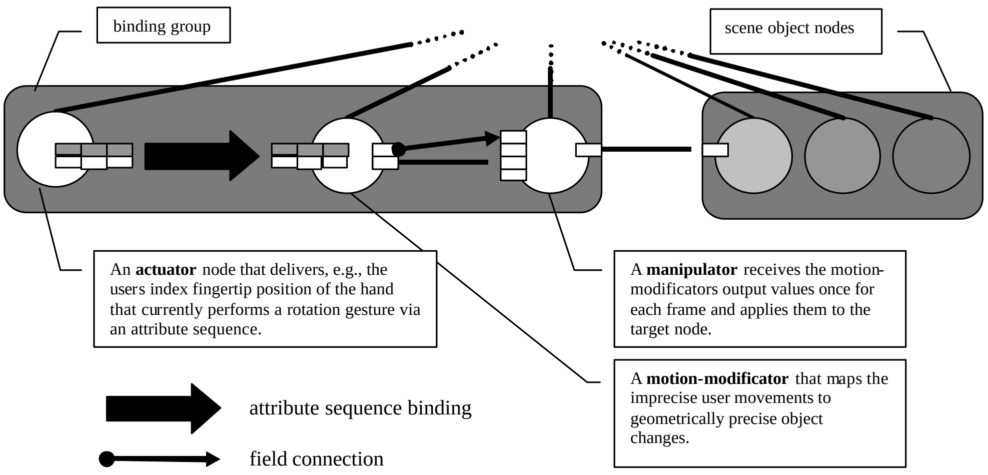


Figure 9: An actuator node as a special node type for encapsulating a user's movement data

Attribute sequences can be **routed** to other nodes with appropriate input sequences just like normal field connections in VRML and similar toolkits. By linking actuators into the scene graph, an explicit user representation – regarding the significant body parts for the utterances – is inserted into the graph. The respective matrices in Figure 9 allow an easy modification of the body data, e.g., to adjust unsuitable sensor fixation points. Internally, the actuator layer which supports the described node type performs still some more actions. For example, it handles and may integrate sensor data of more than one source and it provides a common data rate even with unsynchronized sensor streams.

In the context of a continuous interaction, an actuator routes its data to a **motion-modificator** (**mm**) node which then triggers a manipulator for the desired object manipulation. Figure 8 depicts the resulting setup which is called a **binding** for a continuous interaction. Here, the motion-modificator acts as a filter and controller. It receives the actuators data in the world reference system, e.g., the trajectory of the fingertip, and tests this data according to certain trajectory constraints. If those are satisfied by the



binding group
scene object nodes
An actuator node that delivers, e.g., the users index fingertip position of the hand that currently performs a rotation gesture via an attribute sequence.
A manipulator receives the motion-modificators output values once for each frame and applies them to the target node.
A motion-modificator that maps the imprecise user movements to geometrically precise object changes.
attribute sequence binding
field connection

Figure 8: A binding between an actuator, a motion-modificator and a manipulator to establish a continuous interaction, here the rotation of an object.

movement, the mm outputs new trigger values for the desired manipulation. For example, for a rotation, the mm will output a rotation normal (in world space) and a rotation angular velocity. Both may change during the interaction in predefined ranges to allow a finer grained interaction control. To achieve this in a reliable manner, the mm is instantiated with certain raster values for each of its output values. Regarding the rotation example, the mm has list of possible rotation normal vectors – e.g., the list ((0,0,1), (0,1,0), (1,0,0)) would only allow rotations around the main world axes – and rotation angular velocities. This approach borrows its idea from the *grid* concepts well known from typical 2D and 3D graphics modeling programs to support precise object manipulation. This method of pre-restricting manipulations through an online parameter specification is particularly important in the virtual assembly scenario. Here, two objects can only be connected if they both have matching port types at the connection spots (see section 2.1). If such a connection can successfully be established, only certain degrees of freedom will still be available. Since the mm does not make any assumption about the source of the manipulation constraints, e.g., the list of possible normal vectors, this concept seamlessly enables the adaptation to different requirements. Before an mm is instantiated, possible constraints caused by the specific application, here the virtual assembly scenario can alter the mm's raster values to reflect a proper simulation behavior. Finally, the manipulator node takes the control. It applies the desired manipulation – in the example the next rotation step - for each frame according to the mm's output data by considering possible transformations between the world and the objects reference frames. For a further interaction control, it is noticeable that all of the manipulation values can be altered during the ongoing binding due to their implementation as connectable fields.

The presented example for a decomposition of functional units has also been applied for several more areas. This includes the whole gesture detection process as well as the scene evaluation which is also handled by specialized node types To allow a traversal type access to scene graph structures during an interpretation task, e.g., to find out which objects that have been looked at by the user have the color "green" or are of type "wheel", nodes in the scene graph can be augmented by another special type called **semantic entity (se)**. A se represents the connection layer between an external of multimodal utterances requires access to several types of knowledge base that provides, e.g., conceptual and lexical information about the objects – the AI. This includes perceptual (colors, shapes, positions,…), conceptual (types, functions, roles,…) as well as linguistic information. Some of this knowledge can not be stored statically due to (a) its highly fluctuating quality, e.g., utterances like "*left of…*" or "*behind…*" in a head-tracked or dynamically changing environment and hence is generated on the fly (again using specialized pre-evaluation node types) or (b) because it is not linked to an actual visual representation. For example, if we say "give me a red bolt" there certainly does not have to be one instantiated in the scene already.

Still, the access to the remaining as well as to most of the mentioned information sources should be handled in a uniform manner to establish a common query interface. To allow a traversal type access to scene graph structures during an interpretation, e.g., to find out which objects that have been looked at by the user have the color "green" or are of type "wheel", nodes in the scene graph can be augmented by another special type called **semantic entity (se)**. A se encapsulates the connection layer –between the different knowledge bases and the plain computer graphics representation of the objects in the graph. If a node of type se is visited during a traversal – which might in one instance e.g. search for green and connectable objects – the se enables the traverser to query the nodes underlying knowledge base(s). The most important supported functions to enable a generic data query via attribute query-value pairs are:

```
(1) isOfType(TYPE)
(2) getRefNode()
(3) hasAttribute?(ATTR), hasAttribute!(ATTR)
(4) getAttributeValue(ATTR), setAttribute-Value(ATTR)
```

The first two are for the traversal management, (1) to find out if the node is of type se and (2) to get the according scene graph node whose semantic values are represented by the specific se (currently that one references just the se's parent). (3) and (4) test or set for an attribute and its value. A simple example for a typical traversal is explained in the following pseudo code algorithm (see Figure 10) which performs an object-by-attribute-value search.

```
Begin proc Query (node, attr, value)
  Set ObjectList := NIL;
  If  node.isOfType (GROUP) then
    ForEach child in node.children() do
      ObjectList.append
           (Query (child,attr,value));
    done
  If node.isOfType(SE) then
    If node.hasAttribute (attr) then
      If node.getAttributeValue (attr)
                       = value then
        Return(node.getRefNode ());
  Return (ObjectList);
End proc
```

Figure 10: A typical traversal function to query objects by a specific attribute-value tupel.

This se-approach does not force a specific format of the knowledge base due to the simple query interface specification. For example, the current implementation accesses a plain text database with stored attribute-value pairs for lexical and some ontological information together with a semantic net which handles more complex linguistic and conceptual relations about the construction task.

The given examples shall clarify our approach to stick as close as possible to a scene graph oriented design by a functional decomposition of needed modules. One advantage of attaching se's to scene graph nodes instead of choosing another connection scheme between AI-related knowledge bases and VR-systems is, that it is reasonably simple to use already existing scenes with the multimodal interaction interface. There are, of course, several ways to design and implement a reusable multimodal interaction engine by breaking down the overall tasks and problems into smaller modules which can be implemented as node types. Though our choices work well for our application domains, we are currently investigating even more general node types. These should allow a concept reuse as well as a possible export and exchange formalism using a common file format and hence an easy adaptation to new applications. Our current implementation is based on the AVANGO toolkit [Tra99] of the IMK group at the Fraunhofer Institute in Bonn and makes use of OpenGL Performer's scene graph. But since all new node types are defined just by their clean input/output behavior, it should be quite easy to port them to different toolkits.

4 Summary and Outlook

Virtual environments raise the demand for an intuitive interaction with computer generated surroundings. The desktop-oriented input devices like keyboard and mouse etc. together with their related interaction metaphors often do not seem to be adequate with respect to the requirements and the user's mobility in large-screen immersive surroundings like workbenches, walls or caves. Since 'traditional' input devices and the WIMP (Windows, Icons, Menu, Pointer) style interaction is not suitable for human-computer-interaction (HCI) in an immersive virtual environment, more intuitive HCI has to be developed. In the "Virtuelle Werkstatt" project, we are interested in intuitive and natural ways of interacting in virtual environments for virtual prototype modeling. Generic concepts for multimodal integration using speech and gesture, enrichment of probably imprecise geometric models with information for interaction and assembly, and a knowledge-based approach for describing, establishing, and modifying connections have been developed.

Our current work in the area of multimodal interaction methods aims at an XML specification for defining a tATN. This is done to tackle some problems related to the somehow complex way of constructing or modifying a tATN. Though all concepts are already specified using a scripting language (SCHEME), side effects of a state or constraint modification sometimes lead to quite unexpected results. A plain declarative method would be preferable. The goal is to specify an interaction in XML using a simple text editor, and have this specification compiled in the target structure, namely a tATN.

For the inhomogeneous scaling operations the mapping from the multimodal user input to the correct scaling operation is currently developed. Particularly this will consist of suitable motion-modificators for the interactive part of the scaling operation and a scene graph structure which allows the selection of different divisions of an object in CSG-combined subparts. Furthermore scaling interactions described by two-handed gesture should be recognized and processed and the scaling of already assembled parts should be possible.

Future work will also include more advanced kinematic simulations, which will be used to test the assembled mechanisms of the connected joints with an integrated collision detection. For instance it should be possible to simulate the effect of moving the handle bar of the "Citymobile" on the movement of its front wheels.

Acknowledgement

This work is partly supported by the Deutsche Forschungsgemeinschaft (DFG) under grant Wa 815/2 and the Ministry of Science and Research of the Federal State North-Rhine-Westphalia (MSWWF) under the grant OZ IV A3-107 032 96.

Literature

[Ake95] Kurt Akeley. *OpenGL reference manual: the official reference document for OpenGL.* Addison Wesley, 1995

[Bie00] Peter Biermann. *Interaktives VR-System zur halbautomatischen Generierung von Wissen über Verbindungsmerkmale CAD-basierter Bauteil-Modelle.* Diplomarbeit, Universität Bielefeld, 2001.

[Bol80] R. A. Bolt. Voice and gesture at the graphics interface. In *ACM SIGGRAPH-Computer Graphics*, ACM Press, New York, 1980.

[Cav95] M. Cavazza, X. Pouteau and D. Pernel. Multimodal communication in virtual environments. In Symbiosis of Human and Artifact. Elsevier Science B. V., 1995, pp. 597-604.

[FDFH96] James D. Foley, Andries van Dam, Steven Feiner and John F. Hughes. *Computer Graphics: Principles and Practice.* 2nd edition in C, Adison Wesley Publishing Company, 1996.

[Goldf86] J. Goldfeather, J. Hultquist, H. Fuchs. Fast Constructive Solid Geometry in the Pixel-Powers Graphics System, *Computer Graphics (SIGGRAPH '86 Proceedings),* ACM, Vol.20, No.4, pp. 107-116, 1986.

[Joh98] M. Johnston. Unification-based multimodal parsing. In *Proceedings of the 17th International Conference on Computational Liguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING-ACL 98)*, 1998.

[Joh00] M. Johnston and S. Bangalore. Finite-state Multimodal Parsing and Understanding. In *Proceedings of COLING-2000*, 2000.

[Jung98] B. Jung, M. Latoschik, I. Wachsmuth: Knowledge-Based Assembly Simulation for Virtual Prototype Modeling. *IECON'98 - Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society, Vol. 4*, IEEE, 1998, 2152-2157.

[Koo93] D. B. Koons, C. J. Sparrel and K. R. Thorisson. Integrating simultaneous input from speech, gaze and hand gestures. In *Intelligent Multimedia Interfaces*, AAAI Press, 1993.

[Lat98] M. E. Latoschik, M. Fröhlich, B. Jung and I. Wachsmuth. Utilize Speech and Gestures to Realize Natural Interaction in a Virtual Environment. *IECON'98 - Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society, Vol. 4*, IEEE, 1998, pp. 2028-2033.

[Lat01a] M. E. Latoschik: Multimodale Interaktion in Virtueller Realität am Beispiel der virtuellen Konstruktion. *Phd thesis, Faculty of Technology, University of Bielefeld*, infix DISKI volume 251, Akademische Verlagsgesellschaft Aka GmbH, Berlin, 2001.

[Lat01b] M. E. Latoschik. A Gesture Processing Framework for Multimodal Interaction in Virtual Reality. In A. Chalmers and V. Lalioti, editors, *Afrigraph 2001, 1st International Conference on Computer Graphics, Virtual Reality and Visualization in Africa, 5 - 7 November 2001*, New York, NY 10036, 2001, ACM SIGGRAPH, pp. 95-100.

[Luc98] M. Lucente, G. Zwart and A. D. George. Visualisation space: A testbed for deviceless multimodal user interfaces. In Intelligent Environments Symposium, American Assoc. for Artificial Intelligence Spring Symposium Series, 1998.

[Män88] Martty Mäntylä. *An introduction to Solid Modelling*. Computer Science Press, Helsinki University of Technology, 1988.

[Rot94] K. Roth. *Konstruieren mit Konstruktionskatalogen*, volume I. Berlin, Springer-Verlag, 2. edition, 1994.

[Tra99] H. Tramberend. A distributed virtual reality framework. In *Proceedings of the Virtual Reality Conference 1999*, 1999.

[Vo96] M. T. Vo and C. Wood. Building an application framework for speech and pen input integration in multimodal learning interfaces. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, Atlanta, 1993.

[Wac97] I. Wachsmuth, B. Lenzmann, T. Jörding, B. Jung, M. Latoschik and M. Fröhlich: A Virtual Interface Agent und its Agency. *Proceedings of the First International Conference on Autonomous Agents* (pp. 516-517), 1997.

[Zel91] D. Zeltzer. Task level graphical simulation: Abstraction, representation, and control. In N. Badler, B. Barsky, and D. Zeltzer, editors, *Making Them Move*. Morgan Kaufmann, 1991.

[ZG96] D. Zeltzer and S. Gaffron. Task-level interaction with virtual environments and virtual actors. International *Journal of Human-Computer Interaction*, 8(1):73-94, 1996.