Fachhochschule
Bonn-Rhein-Sieg

Fraunhofer Institut
Intelligente Analyse- und
Informationssysteme

Interaction Techniques for Assembly based Modeling in Virtual Environments

Master Thesis, submitted by David d'Angelo

Supervisors:
Prof. Dr. Rainer Herpers
Prof. Dr. Wolfgang Heiden

External supervisor:
Dr. Gerold Wesche

Fachhochschule Bonn-Rhein-Sieg
University of Applied Sciences
Fachbereich Informatik

17.3.2008

Statement of originality

The material in this Master Thesis "Interaction Techniques for Assembly based Modeling in Virtual Environments" has not previously been submitted for a degree in any university and to the best of my knowledge contains no material written by another person except where due acknowledgment is made in the thesis itself.

David d'Angelo

Acknowledgements

I would like to sincerely thank the many persons who, through their continuous support, encouragement and advice, have helped to complete this work.

First, I would like to especially thank Prof. Dr. Rainer Herpers and Prof. Dr. Wolfgang Heiden, not only for supervising this thesis, but also for insightful conversations during the development of the ideas in this thesis and for valuable feedback on the thesis.

A special thanks goes to all my colleagues at the Competence Center Virtual Environments at the *Fraunhofer Institut für Intelligente Analyse- und Informationssysteme* for many inspiring and fruitful discussions. In particular I would like to thank Dr. Manfred Bogen, Dr. Gerold Wesche and Dr. Maxim Foursa. Without them and their everlasting support this thesis would not have been possible.

I also want to thank the entire INT-MANUS consortium for giving me the opportunity to integrate my system into a real manufacturing process chain.

I am extremely grateful for my family's continuing encouragement and support in any of my endeavors.

Finally, I want to thank Julia for her endless moral support during the sometimes turbulent time writing this thesis.

Contents

List of Figures

1 Introduction

This chapter gives a short overview of the content of this theses. At first, a motivation for assembly based modeling in Virtual Environments in general is given, followed by the goal and the structure of this thesis.

1.1 Motivation

Modeling systems are widespread and successfully used in a lot of application areas, like mechanical part design or character modeling in the film industry. However, in common non-immersive desktop based modeling systems, it is often difficult to conceive three-dimensional shapes out of orthogonal views or perspective projections of the three-dimensional modeling space.

In contrast to this, an immersive modeling system in a Virtual Environment allows direct interaction and perception of the three-dimensional objects in the scene and provides a user with spatial perception cues that simplify the understanding of the modeling environment. Thereby, a user of such an immersive modeling system is provided with the perception Being-In-The-World, opposed to the Window-On-The-World concept of non-immersive desktop based modeling systems.

Major commercially relevant applications of immersive modeling systems are computer aided design applications that involve the design and evaluation of so called virtual prototypes. In general, the goal of a virtual prototype is to enable the testing and evaluation of different aspects of part design and behavior of a modeled virtual prototype without the need to manufacture a corresponding real physical prototype.

A recent trend in virtual prototyping is the development of customized products based on combinations of mature product models. Product customization techniques are extensively used by the automotive industry where a broad range of products are derived from the same set of basic product models. These techniques help to significantly reduce the production costs and the design costs of new products.

1.2 Goals of this thesis

The goal of this thesis is the development of an immersive assembly based modeling system called VEAM *(Virtual Environment Assembly Modeler)*. This includes the concept development and the implementation of novel assembly algorithms and the design of innovative selection, manipulation and system control interaction techniques.

The assembly algorithms to be developed should not rely on the geometric representation of parts, since every polygonal model is only an approximation of the real geometry of a part. Instead, they should use a grammar-based model to encode the connection information of the parts. The core of this product configuration grammar is the so-called *Handle* concept. A handle is an attribute of a part and specifies its connection possibilities. The VEAM system should provide different handle types, each defining a different connection semantic. In general, the handles should fully describe the set of models possible to construct out of a set of parts and should assist users during the actual assembly process.

A data structure called *ConnectionGraph* ought to be used to store the connection information of all assembled models. The *ConnectionGraph* should be used to identify possible breakups of models, which can be caused by the disassembly of parts from an assembled model. In addition, the serialization of models, needed by the model import and export functionality of the VEAM system, should be based on this data structure, since it contains all relevant information to completely reconstruct a model.

A key requirement of the VEAM system is its integration functionality into existing manufacturing process chains. The system should be integrated into the INT-MANUS [FSM+06] manufacturing process chain and this integration ought to be tested in a real life factory. The INT-MANUS project is a European research project coordinated by the *Fraunhofer Institut für Intelligente Analyse- und Informationssysteme.* It targets the development of an innovative control of production processes in manufacturing enterprises. The core of the system is the *Smart-Connected-Control-Platform (SCCP)*, which is capable of autonomously handling dynamic production changes. These dynamic production changes can be beneficial in various situations, such as in case of a machine breakdown where its production capacity must be distributed among the remaining functional machines or in case new parts should be produced.

Another integral objective of this thesis is the development of innovative interaction techniques for the VEAM system. The system should support selection techniques for single and multiple object selections, as well as isomorphic and nonisomorphic transformation techniques. All of these interaction techniques should use well designed control-display mappings for the conversion of user input into system specific functionality.

The VEAM system should not only support single user modeling but also collaborative two-user modeling. For the two-user mode, special interaction techniques ought to be

developed.

For system control tasks, the VEAM system should use embedded graphical menus. These menus should refer to the actual context defined by the active tool and the selected object. To reduce the occlusion effects caused by the menus, all menus should be drawn in a semitransparent mode and should be positioned in the modeling environment according to a rule based placement policy. All of the developed widgets should not be bound to the VEAM system, but should rather be application independent. The developed widget toolkit should provide a variety of different adapted two-dimensional widgets, like push buttons, checkboxes, sliders, file dialogs and widgets for handling symbolic input. In addition to these two-dimensional widgets, the toolkit should support three-dimensional widgets. These widgets can be seen as a combination of geometry and behavior and they should exploit the additional degree of freedom of a three-dimensional environment.

In a first user study, the efficiency of the single user and the collaborative two-user modeling functionalities of the VEAM system should be evaluated and compared to a reference modeling system.

1.3 Structure of this thesis

This thesis is divided into eight chapters.

Chapter two introduces the term Virtual Environments and describes various common output and input devices used in Virtual Environments. In addition this chapter describes the basic concepts of the Virtual Reality framework AVANGO [Tra99] and its scenegraph.

The state-of-the-art of interaction techniques for Virtual Environment applications, with a special focus on system control, selection and manipulation techniques, is discussed in chapter three.

The fourth chapter develops the concepts behind the developed VEAM system and every module of the system is explained in detail. This chapter also contains a discussion about existing immersive modeling systems.

Chapter five discusses the physical input devices used in the VEAM system and describes the concepts of the developed single- and two-user interaction techniques utilized for system control, selection and manipulation tasks.

The software design of the VEAM system and the developed interaction techniques are described in chapter six.

In chapter seven, the evaluation and the conducted user study of the VEAM system and its interaction techniques is described.

Chapter eight summarizes the concepts developed and implemented in this thesis, discusses whether the goals of this thesis were achieved and provides ideas for further improvements.

Appendix A contains an explanation of geometric transformations. These operations are indispensable for any modeling system and are therefore frequently used in the implementation of the VEAM system and its interaction techniques.

The analysis of the questionnaire used in the evaluation together with the comments of the participants is given in appendix B.

1.4 Publication

Within the scope of this thesis, a conference publication with the title: "A Two-User Virtual Environment for Rapid Assembly of Product Models within an Integrated Process Chain" [FWd+08] is going to be published. It contains a description of the ideas and concepts of the immersive modeling system introduced in chapter four and of the interaction techniques as described in chapter five.

2 Virtual Environments

A Virtual Environment is a computer generated environment that provides the sensory experience of being in simulated space. To generate this immersion effect, a Virtual Environment typically stimulates the different senses of a user and provides functionality to interact with virtual objects.

This chapter introduces the basic components of a Virtual Environment and it is divided into three sections. First, common output devices of Virtual Environments are introduced, followed by a discussion about different input devices. The third section describes the software framework AVANGO™ [Tra99] which is used in this thesis for programming Virtual Environments.

2.1 Output Devices

Every Virtual Environment needs output devices that stimulate the senses of a user. Since the human perception system has different senses, various kinds of output devices exist. The majority of them are focused on visual stimulation (section 2.1.1), on the auditory (section 2.1.2) and the haptic (section 2.1.3) sense. In some applications, the olfactoric sense of a user is also stimulated [FDL01].

A computer system is needed to generate input for the output devices, which is in turn translated into a human perceptible format by the output devices. Since only visual stimulation is used within this thesis, the emphasis of this section lies on visual output devices. A comprehensive description of output devices can be found in [BC03, SC02].

2.1.1 Visual Displays

Visual displays use the human visual system to display information. It is the most common display used in Virtual Environments. For converting the information into a perceptible format, various rendering techniques are used. These techniques generate images which are presented to a user by a visual display. Since many different real time rendering techniques have been developed and their description is not the focus of this thesis, further information about them can be found in [MH99].

Display Characteristics

For the classification of visual display devices, some important characteristics must be considered. In the area of Virtual Environments, these key characteristics are:

Field of Regard/View The field of regard (FOR) is the total size of the visual field of the display device surrounding a user, whereas the field of view (FOV) denotes the size of the visual field that can be instantaneously viewed by a user. Both quantities are measured in degrees.

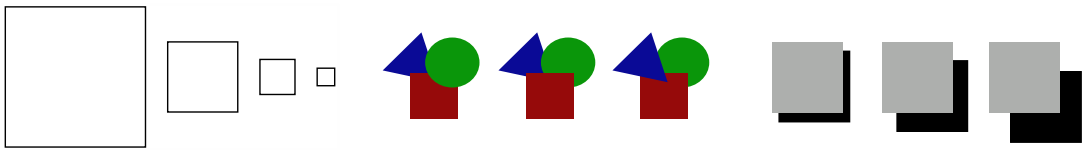
Spatial Resolution The spatial display resolution depends both on the number of pixels of the screen and its physical size. It is typically measured in dots per inch (dpi). A closely related property is the spatial resolution based on the perceptual level. This means, the farther away a user is from a display, the higher the perceived resolution is because the individual pixels are becoming indistinguishable.

Screen Geometry The screen shape is another important characteristic of a display device. There are many different screen shapes, such as rectangular, L-shaped and hemispherical displays. For non rectangular displays, special efforts have to be made to develop non-standard projection techniques to avoid reduction of the visual quality of the projection.

Light Transfer The way how light is transferred from the emitter to the display surface is called light transfer. Common ways to do this are front and rear projections or the use of special optics like a head-mounted display.

Refresh/Frame Rate The refresh rate is the speed of the display device used to fetch a new image from the frame buffer, whereas the frame rate refers to the speed of the generation of new images by the graphic system and its placement in the frame buffer. Both quantities are usually measured in Hertz.

Ergonomics The ergonomic aspect of a display system is especially important when a user has to carry the display devices, as in the case of a head-mounted display. The goal is that a user can interact with the virtual environment in the most comfortable way possible.



(a) Depth cues by relative object size.

(b) Depth cues on the basis of occlusion.

(c) Depth cues caused by shadows.

Figure 2.1: Different kinds of monocular depth cues.

Depth Cues

The understanding of visual depth is crucial for the comprehension of the three-dimensional structure of the scene.

This understanding will assist a user while performing three-dimensional interaction tasks. Thus, it is important to distinguish between different kinds of depth cues.

Monocular depth cues Depth cues that can be extracted out of a single static image using only one eye are called monocular or pictorial depth cues. Amongst others, they can contain interrelation of the objects, like relative size, distance to the horizon, occlusion or shading.

Figure 2.1a contains a set of rectangles with decreasing size. Since no information about the real size of the image is known, the smaller the rectangles become, the farther away they appear to be.

Another kind monocular depth cue is occlusion. Objects that are closer to the user are hiding other farther objects. An example for depth cues by occlusion is shown in figure 2.1b.

Shadows casted on objects or by them onto adjacent surfaces are also used to generate depth cues. The three rectangular patches shown in figure 2.1c are all of the same size, but differ in the location of their shadows. Due to this, the rightmost rectangle appears to stand out the most. Lighting is also an indicator for depth. Many observers also assume that closer objects have higher illumination than object farther away.

Motion Parallax This term refers to the movement of objects relative to an observer or vice versa. Objects closer to an observer are moving quicker across the visual field than objects that are farther away. In contrast to monocular depth cues, motion parallax is a dynamic depth cue since it needs a moving image sequence (see figure 2.2 for an example of motion parallax).

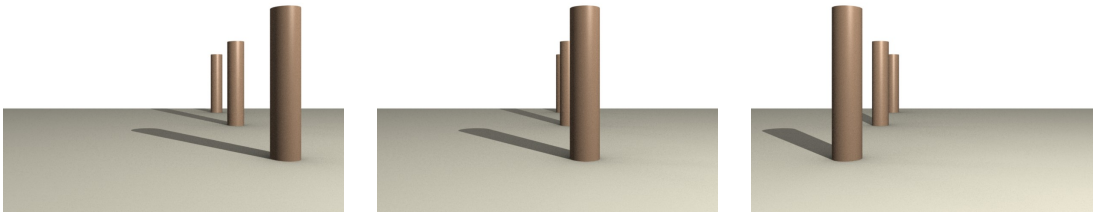


Figure 2.2: The image sequence is illustrating depth cues using motion parallax. The viewer moves from left to right and the cylinder in front moves the most of the three objects.

Binocular Disparity Most humans visually perceive the world with their two eyes which have a slight offset with respect to each other. The resulting disparity of the two images is called *binocular disparity*. Fusing these two images can be used to generate a very strong depth cue called *stereopsis*. Figure 2.3 illustrates binocular disparity and the effect of stereopsis.

Display Types

Since Sutherland's definition of the *ultimate display* [Sut65] many different visual display systems for Virtual Environments have been built. The following paragraphs will briefly introduce the four most common display systems with a focus on projective surround screen displays, since such a system is used in this thesis.

Monitors Common monitors are frequently used for three-dimensional applications. Without any additional hardware, they provide monocular and motion parallax depth cues. In case stereopsis is needed, so-called stereo glasses can be used. Stereo glasses can be divided into two categories, namely active and passive stereo glasses.

Active stereo glasses are commonly called *Shutter Glasses* and are synchronized to open their shutters in the same frequency as the refresh rate of the visual display. Since the display is showing the image for the left and right eye alternately on the screen, the shutter glasses are blocking one eye in order to prevent it from seeing the image corresponding to the other eye. For the synchronization of the shutter glasses with the refresh rate of the visual display commonly infrared signals are used.

Passive stereo glasses either use *spectral multiplexing* or *polarization filters*. In the case of spectral multiplexing, the two images for the left and right eye are encoded with different colors and then combined in one image. The glasses then use color filters to block all light except for the filter's color. Another possibility for the separation of the image pair is the usage of polarization filters for each eye. Common polarization filters use linear (horizontal and vertical) or circular (clockwise and counter clockwise) polarization.

(a) Overview of the scene (b) Image of the left eye (c) Image of the right eye

Figure 2.3: Images illustrating the effect of stereopsis by binocular disparity. In the overview image (a) the two viewpoints used for generating the images (b) and (c) are shown as blue cones. The binocular disparity between (b) and (c) can be best seen by the position and the orientation of the cube.

Autostereoscopic Displays An autostereoscopic display provides motion parallax and stereopsis depth cues without the use of stereo glasses or other viewing aids. The most common kind of autostereoscopic displays in the area of Computer Graphics are the so-called *Parallax Displays*. In general they emit directionally varying image information into the viewing zone and use optical components, like an array of cylindrical lenses or a parallax barrier, to generate the effect of having distinct images visible in the same plane from different view points. A detailed description of parallax autostereoscopic displays is given in [Dod05].

Head-mounted Displays A head-mounted display, often abbreviated as *HMD*, is a visual display that is coupled directly with the head of a user. Its main purpose is the placement of images right in front of a user's eyes using one screen for monoscopic and two screens for stereoscopic viewing. In terms of visual depth cues, a tracked HMD allows monoscopic, motion parallax and stereopsis depth cues. An advantage of a HMD is its field of regard (360 degrees), since a user sees the virtual environment at all times, independent from her actual position and orientation. However, the field of view of a standard HMD is quite limited (typically around 60 degrees in horizontal direction) compared to the field of view of a human (around 180-200 degrees in horizontal direction).

Table-like Displays The Responsive Workbench [KF94] is an example for a stereo projection based table-like display. The system is derived from the workbench metaphor and was designed to model interaction techniques that are taking place at desks or tables. The responsive workbench provides a high spatial resolution, like most of the projection based display systems and is capable of generating monoscopic, motion parallax and stereopsis depth cues, assuming a tracking system is used. In general, it is possible that multiple users share the same workbench, but since most workbenches are only capable of generating one stereo image pair, the perspective is only correct for one user.

(a) The *TwoView* display system.

(b) The *i-Cone* surround screen display system.

Figure 2.4: Two displays systems developed at the *Fraunhofer Institut für Intelligente Analyse- und Informationssysteme.*

Surround Screen Displays A surround screen display has one or more large screens that are commonly projection based. In most cases rear projections are used in order to avoid the casting of shadows by users in the environment. In some cases a front projection can be used without causing any disturbance if the shadows are always casted away from the visual field of all users or are not casted onto the projection surface at all. Surround screen display systems typically provide a high spatial resolution along with a large field of regard. In contrast to HMD systems, a user can use her own peripheral vision system[1]. Therefore, surround screen displays have a large field of view.

In terms of depth cues, a surround screen display equipped with a user tracking facility provides monocular, motion parallax and stereopsis depth cues. Opposed to *Fish Tank VR* [WAB93] (monitor with head tracking) a user has a much better experience of motion parallax because a user can actually walk around in the environment. Stereo viewing of users can be achieved by using active or passive stereo techniques (see paragraph about the monitor as an output device in this section). Additionally, real objects can be augmented with virtual objects. E.g. a chassis of a real car can be placed in the Virtual Environment and a user can use its controls to drive through the environment.

The first surround screen display was called *CAVE* and was build by the University of Illinois in 1993 [CNSD93]. It consisted of four screens in total, three on the walls and one on the floor. Since that time many different surround display systems have been developed.

A cylindrical panoramic surround display system suited for larger groups of participants is the *i-Cone [SG02]* display system (see figure 2.4b). It uses a front projection based conical screen geometry and has a field of regard of 240 degrees. Since a lot of participants are using the display at the same time, no tracking system is used. However, the stereo image is calculated for a fictive position in the center of the display system. If a user is

[1] Peripheral vision is also called *side vision* and describes the ability to see objects and movements outside the center of gaze.

not standing too close at the border of the display, the perspective distortion is within acceptable limits for most applications.

TwoView A display system developed at the *Fraunhofer IAIS*[2] is called *TwoView* and is capable of displaying two active stereoscopic image pairs at the same time. The system uses two projectors equipped with circular polarization filters. Each projector is responsible for creating a stereoscopic image pair for one user. The separation of the image pairs for each of the two users is done by shutter glasses. Figure 2.4a shows the *TwoView* system.

The *TwoView* display system is used as visual output device in this thesis.

2.1.2 Audio Displays

Auditory displays are often overlooked in Virtual Environments [CW95], but can be of great use in many applications. The generation of three-dimensional sound is a large field and it is out of the scope of this thesis. A rich introduction along with further information about the generation of three-dimensional sound and audio displays can be found in [Kap03].

The following paragraphs give a short overview over the most important aspects on how a Virtual Environment application can benefit from the use of auditory displays.

Localization A major goal of an auditory display is to generate a spatial sound that allows a participant to use her own sound localization capabilities. Using the aural sense, a user is getting an audio depth cue. Amongst others, this can be helpful for navigation tasks or for the localization of points of interest (e.g. enemies) in the environment.

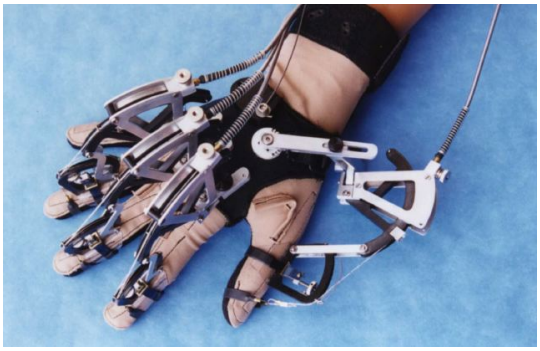
Ambient Effects Adding ambient sound effects to a scene can greatly enhance the immersion of a user (imagine a walk through the rain forest, where the birds are singing and the leaves of the trees are shaking, ...).

Sensory Substitution In case an environment is not capable of stimulating a specific sense of a user, like haptics, sound can be used as a replacement. E.g. pressing a button can play a sound instead of the feeling of touching a real button.

Sonification Sonification describes the conversion of information into sound. E.g. one or more dimensions of a multidimensional data set can be converted into sound, in order to

[2]For more information about the *Fraunhofer Institut für Intelligente Analyse und Informationssysteme* visit: www.iais.fraunhofer.de

(a) Phantom



(b) CyberGrasp

Figure 2.5: Two haptic displays. The *Phantom* (Courtesy of SenseAble, www.sensable.com) is a ground-referenced device and the *CyberGlove* (Courtesy of Immersion Corporation, www.immersion.com) is a body-referenced device.

help a user getting a better understanding of data. This technique is often used in Virtual Reality applications that are dealing with the interpretation of seismic data.

2.1.3 Haptic Displays

Haptic displays are trying to provide a user with the sense of touch while interacting with virtual objects by simulating physical contact between the object and the user. In general, haptic devices can be input and output devices at the same time. The term *haptic rendering* refers to the techniques for controlling haptic displays.

The following paragraphs contain a brief classification of the most commonly used haptic display devices. Further information can be found in [SC02, BC03].

Ground-Referenced Devices These devices are creating a physical connection between a user and the ground. Some devices can be put on a desktop, whereas other devices may need to be connected to the ground. The ground connection enhances a user's comfort, because the user needs not bear the weight of the device. On the other hand, this physical connection of the device between the ground and a user normally limits the interaction range. Figure 2.5a shows a well-known ground based haptic device called *Phantom*.

Body-Referenced Devices Unlike ground based devices, body referenced devices are not physically connected to the ground, but rather linked to some part of a user's body. The main advantage of this kind of haptic devices is the freedom of movement since the device can be carried around. However, this limits their size and their weight. In Figure 2.5b, a body referenced haptic device called *CyberGrasp* is shown.

Tactile Devices Tactile display devices are much smaller and lighter than force display devices (like ground or body referenced devices) and are stimulating the tactile sense of a user. They often use actuators like vibrators or pin arrays. Most of the current game consoles, such as the Sony PlayStation 3[3] or the Microsoft XBox 360[4] are using vibrators in their game pads as tactile displays.

2.2 Input Devices

For designing a Virtual Environment application not only the choice of the output displays is important. An equally important decision is the selection of input devices. An input device allows a user to communicate with an application. E.g. some applications need head tracking whereas other applications may need a microphone for handling speech input. An important distinction must be made between input devices and interaction techniques (see chapter 3). Input devices are physical tools for implementing interaction techniques. In most cases a specific interaction technique can be mapped to any input device. However, some input devices are more adequate for particular interaction techniques than others.

One of the most important characteristics of an input device is its degree of freedom (*DOF*). Common tracking devices provide a three-dimensional world position of the tracked object along with its orientation in three-dimensional space. Thus, such a device would be referred to as a six degrees of freedom input device. A device with a smaller *DOF* can often emulate a device with higher *DOF* by using buttons or modifier keys.

In the following sections different kinds of common input devices are classified according to the frequency they are capable of generating input values. A more detailed description and other classification schemes of input devices are given in [SC02, BC03].

2.2.1 Discrete Input Devices

Devices of this kind generate one event at a time, which can be triggered by a user. In most cases, it is a boolean event that signals if a button is pressed or released. A common example for a discrete input device is the keyboard. Another device is the so-called *Pinch Glove*. It is a pair of gloves that has a conductive material at its fingertips. If two fingers are pressed together, the electronic contact generates a boolean event.

[3] For more information about the PlayStation3 visit: www.playstation.com
[4] More information about the XBox 360 can be found under www.xbox360.com

2.2.2 Continuous Input Devices

In contrast to discrete input devices continuous input devices are generating a continuous stream of events. This can happen on behalf of an action triggered by a user or independently. Devices of this kind are important in case a user wants to get information about the environment without asking for it all the time. Two common continuous input devices are *Trackers* and *Data Gloves*. The following sections will describe these two kinds of devices.

Trackers

An important aspect of Virtual Environment systems is how the correspondence between the physical and virtual world is established. E.g.: many applications need the exact physical position and orientation of the head of a user in three-dimensional space, in order to generate correct motion parallax and stereoscopic depth cues. Hence, it is fundamental for Virtual Environment systems to provide a tracking system.

The main characteristics of a tracking system are its range, latency (time delay between the motion and its report), jitters (noise and instability) and accuracy (difference between the real motion and the reported motion). The following paragraph describes two of the most common tracking techniques. More information about tracking systems and their properties are given in [WF02].

Magnetic Trackers These kinds of trackers use a transmitter device, called *source*, which is emitting a low frequency magnetic field. A sensor often referred as *receiver* is used to estimate its position and orientation with respect to the magnetic source. In the close-up range, common magnetic tracking systems have an accuracy of 1 *mm* with respect to the position and 1 degree with respect to the orientation. The main drawback of magnetic tracking systems is that any conductive material in the environment can affect the shape of the magnetic field by electromagnetic induction and therefore decreases the accuracy. Thus, such systems are primarily used for small display systems, like the responsive workbench or for *Fish Tank VR*.

Optical Tracking In contrast to magnetic tracking, optical tracking measures reflected or emitted light to estimate the position and the orientation of a user by applying *Computer Vision* algorithms. An optical tracking system consists of two general components, light sources and optical sensors. Light sources can be either passive objects reflecting ambient light or active devices emitting light. E.g. natural surfaces in the environment are passive light sources while active light sources can be simple light bulbs or light-emitting diodes

Figure 2.6: A data glove with 18 sensors, which is capable of estimating the bending and the abduction angle of all fingers (Courtesy of Immersion Corporation, www.immersion.com).

(*LEDs*). Light in the infrared spectrum is often used since it is not visible for humans and therefore does not distract the visual perception of a user.

One big advantage of optical tracking systems is their range (in most cases the user is not physically connected to a computer) and their accuracy. A major problem of all optical systems is occlusion. This problem becomes more prominent when multiple tracked users share the same Virtual Environment and can move freely. Adding more cameras and landmarks can reduce the occlusion problem but increases the complexity of the tracking algorithm.

The *TwoView* (see paragraph about surround screen displays in section 2.1.1) system used in this thesis uses an optical infrared tracking system with four cameras in a so-called *outside-in* configuration. This means that the cameras are located at fixed positions in the environment and the tracked objects have landmarks attached to them. In this case, the landmarks are passive retroreflective markers, which are illuminated by infrared light.

Data Gloves

In case precise tracking information about a user's hand is needed, an input device called *Data Glove* can be used. A data glove is a passive input device that is utilised to estimate hand postures and hand gestures (series of postures). Figure 2.6 shows a data glove. In general, gloves are used as input devices for system control (see section 3.2) and navigation (see section 3.3) techniques.

2.3 The VR Software Framework AVANGO™

Many frameworks for developing virtual reality applications exist. They all encapsulate the complex geometry and the rendering in a scenegraph data structure and provide a

highlevel API for programmers. Since all techniques described in this thesis are implemented using the virtual reality framework AVANGO™ [Tra99], this section introduces the main concepts of AVANGO™ with a special focus on the used scenegraph and the interaction module. A complete description of AVANGO™ can be found in the dissertation of H. Tramberend [Tra03].

AVANGO™ is an open source virtual reality framework, which is under constant development by the *Fraunhofer IAIS*. It provides comprehensive event and object models and supports generic display devices. The system also has a full distribution support by a transparently shared scenegraph together with an integrated event distribution mechanism.

The states of all AVANGO™ objects are encapsulated in so-called *fields*. Each object is a *field container* and can contain a collection of arbitrary fields. Connections between fields can be established using a *field connection* mechanism. If a field is connected to other fields and its value has changed, all connected fields will be informed and set to the according value. This mechanism builds a data-flow graph that is orthogonal to the scenegraph. This field connection mechanism can be seen as a modification of the *Observer Pattern* [GHJV95].

The core of AVANGO™ is strictly modular and is written in the *C++* programming language. In addition to the C++ API, a complete language binding to the interpreted scripting language *Scheme* [Dyb96] and an interactive runtime shell is available. This extensive scripting functionality offers an effective mechanism for rapid application prototyping.

2.3.1 Scenegraph Concept

AVANGO™ and most of the other Virtual Reality frameworks, like *VR Juggler* [BJH+01] use a data structure to manage and encapsulate the virtual objects of a scene. This data structure is called scenegraph. Many different scenegraph concepts have been developed so far, whereas all have some common properties.

This section introduces the basic concept of a scenegraph and briefly describes some of the most important properties of the *Performer* scenegraph [RH94] which is used by AVANGO™.

The *Open Graphics Library* (*OpenGL*) allows the programmability of the image pipeline of graphics cards. Common operations are the drawing of primitives, setting different states, like color or textures and manipulating the model-view and the perspective matrix. This basic functionality is enough for small scenes but can become cumbersome and error-prone for larger scenes.

To provide a mechanism for organizing larger scenes, the scenegraph concept has been developed. The objects within the scenegraph data structure are organized in a hierarchy, with the objective of propagating all shared properties towards the root of the hierarchy, while children are inheriting the properties of their parents. Most scenegraphs are using a directed acyclic graph (*DAG*) as an internal data structure. This describes a hierarchy in which each parent node can have an arbitrary number of children.

The nodes of a scenegraph represent the scene and can contain a variety of properties, like geometry, transformations, grouping information, lights or material properties. The edges of the graph represent a logical or spatial relationship between the adjacent nodes.

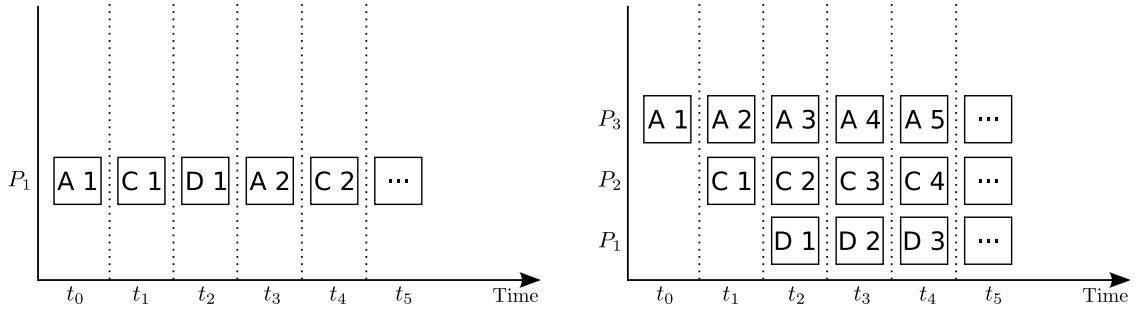
The rendering of a scenegraph is achieved by a traversal of the graph with a virtual camera which defines the view frustum. In most cases, the scenegraph has to be traversed multiple times per frame in order to execute different calculations. The *Performer* scenegraph provides three different kinds of traversals.

Cull Traversal (CULL) During the culling traversal all geometry nodes are checked for visibility. Since the view frustum defines the visible area, a geometric object is only visible if it is lying inside the view frustum. All objects or parts of the objects that are not intersected by the view frustum are clipped and ignored for further calculations. All remaining objects are sorted into draw bins, which are rendered in a specific order. E.g. *Performer* has a draw bin for opaque and another draw bin for transparent geometric objects.

Draw Traversal (DRAW) The set of draw bins generated by the cull traversal is first sorted by their rendering order followed by the creation of display lists containing the geometry of each bin. These display lists are then sent to the rendering pipeline of the GPU for drawing.

Intersection Traversal (ISECT) The intersection of line segments with all visible geometric objects of a scene is done by the intersection traversal. This is especially important for the interaction with the scene, because it can be used to determine the object the user is pointing at, which is a common task in many virtual reality applications.

The split-up of the traversals into these three stages allows an efficient symmetric multi processing. *Performer* uses a pipelining technique aiming at a high efficiency for workstations with several processors. Opposed to this pipelining technique, other systems are targeting at a large-scale computer clusters with thousands of processors. The multi processing rendering pipeline consists of the APP, CULL and DRAW stage, whereas the APP stage executes the application specific calculations. In diagram 2.7 an illustration of the



(a) Sequential execution (b) Parallel pipeline execution

Figure 2.7: Illustration of the sequential (a) and parallel pipeline (b) execution of the *Performer* rendering pipeline.

sequential execution and the parallel execution of the *Performer* rendering pipeline using three processes is shown.

2.3.2 External Device Abstraction

Since interaction is an important aspect of this thesis and the integration of external input devices is needed for each interaction task, the interface of AVANGO™ to the "real world" is described in this section.

An external device daemon is used to handle the interaction with physical input devices via a physical connection, like USB or bluetooth. This process is called *av-daemon* and continuously reads the properties (like position and orientation in space or the button states) of the input devices and writes them into a shared memory segment. Since multiple input devices can be handled simultaneously by the av-daemon, all input device data is tagged with a *station* object. Thus, each physical input device like a space mouse has an appropriate *station* object, which uniquely identifies this input device.

The real AVANGO™ application is running in another process called *av-sh* and uses *sensor* objects to access the shared memory segment in order to get values from the *av-daemon*. Since *station* objects are used to assign data in the shared memory segment to an input device, the *av-sh* process uses specific *sensor* objects to access data of specific input devices in the shared memory segment.

In order to add a new input device using this mechanism, only a new *station* object along with the appropriate *sensor* object have to be implemented in order to use the device.

2.3.3 Interaction Abstraction

AVANGO™ uses the so-called *Tool-Dragger-Concept* as an abstraction to represent interaction patterns between objects. It is a variation of the *Mediator Pattern* proposed by

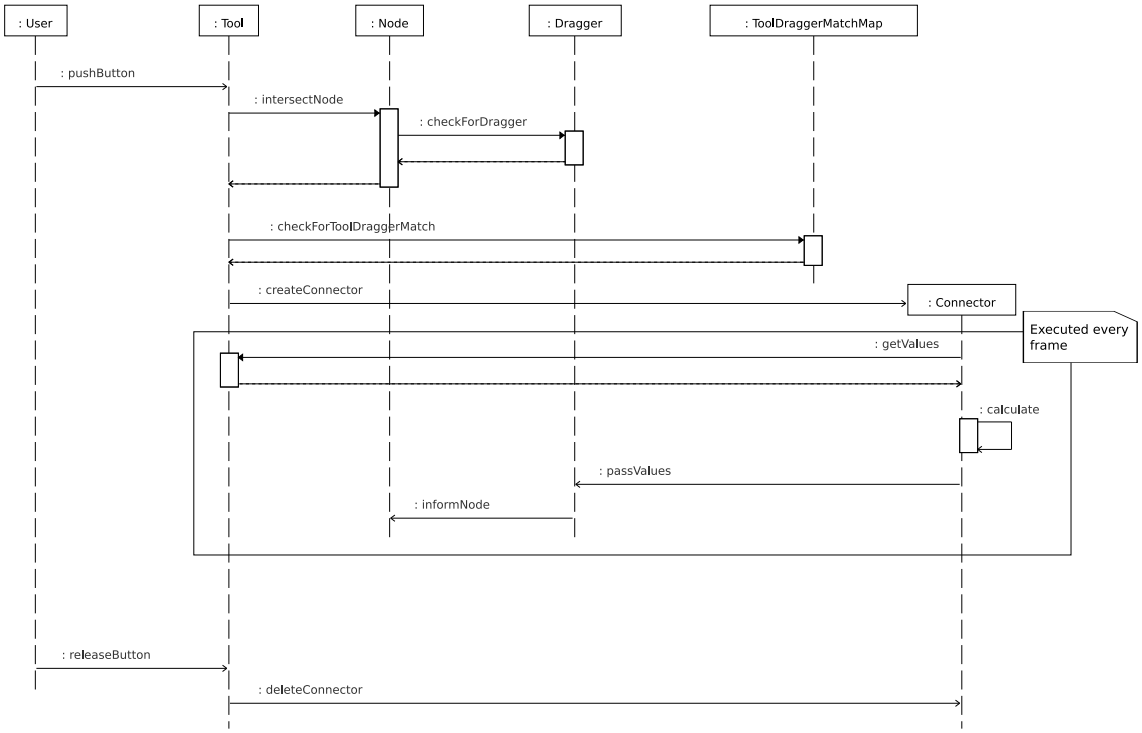


Figure 2.8: Simplified UML sequence diagram illustrating the creation and the lifetime of connector objects using the *Tool-Dragger-Concept*.

[GHJV95]. The goal of the *Mediator Pattern* is to use a mediator object to coordinate connections or state changes between other objects. Instead of distributing the logic over all objects, the logic is encapsulated in one central object. This results in a compact implementation of the logic and in a decreased coupling between all objects.

Tool-Dragger-Concept

The basic idea of this concept is that tools can not interact directly with virtual objects, but rather with draggers which are attached to the objects. In this concept the virtual objects are purely passive and the draggers and the connector objects are mediating between the tool and the virtual object.

Each tool is attached to a toolholder whereas the trigger of the toolholder activates the tool. In most cases the trigger mechanism of a toolholder is connected to a *sensor* object via a field connection.

If the toolholder is triggered, the tool is activated and tries to select a node in the scene-graph. This is mostly done by a selection using a ray casting technique. However, other selection techniques are also possible. In case an object is selected, the system checks if at least one dragger object is attached at this node. If a dragger object is found and the combination of the tool and the dragger is registered in the ToolDraggerMatchMap, a

temporary connector object is created.

The interaction is calculated by the connector object since it knows the exact type of the dragger and the tool. The connector then passes the interaction result to the dragger, which in turn informs the node about the interaction. This communication is commonly performed using the field connection mechanism.

A simplified UML sequence diagram illustrating the above described interaction workflow is shown in figure 2.8.

The following paragraphs describe the intent and the functionality of all relevant modules and classes used in the description of the *Tool-Dragger-Concept*:

Dragger The interface between objects in the scenegraph and the tools that are manipulating them is represented by dragger objects. Every node in the scenegraph can have an arbitrary number of draggers attached to it.

ToolDraggerConnector These objects represent the connection between a tool and a dragger. A connector object is created if a tool has selected an object in the scenegraph which has a matching dragger attached to it. If a combination of a specific tool and a dragger are matching is determined by the ToolDraggerMatchMap. As long as the connection between the tool and the dragger is established the connector object calculates the specified interaction. As soon as this connection is closed, e.g. a user released a button, the connector object is deleted.

ToolDraggerMatchMap This map defines the connector object which is associated with a specific dragger and tool combination. E.g. a matrix manipulation tool matches to a matrix dragger and as a result a temporary matrix connector object will be created.

ToolHolder A toolholder is the software counter part of a physical input device. To get the events of the physical input device, it is connected to a *sensor* object via the field connection mechanism. Each toolholder can have only one active tool at any time, but it can be switched at runtime. The concept of a toolholder can be compared to a screwdriver (toolholder) which is capable of using different bits (tools), but only one bit at a time.

Tool This is the abstract base class of all AVANGO^{TM} tools. Each concrete subclass represents a specific kind of interaction, such as the dragging or scaling of geometry objects. A concrete tool is not manipulating an object directly, but rather manipulates the dragger attached to this object. In case the tool has a three-dimensional representation like a hand or a ray, a suitable geometric object can be added to the tool.

ToolManager This class is responsible for managing all available toolholders and tools and is commonly used as the central instance for accessing information about all registered toolholders and tools.

3 Interaction Techniques

Interaction techniques are methods to accomplish a specific task using a suited interface, in which both hardware and software components are used. The software components of interaction techniques are called *control-display mappings*. They are responsible for translating information provided by an input device into specific system actions. In most cases, a specific interaction technique can be implemented by a variety of different input devices, using device specific control-display mappings. According to [BKJP05], interaction techniques in Virtual Environments can be divided into three main categories: Selection and Manipulation, System Control and Navigation. The following sections describe these three interaction categories. Further details about interaction techniques can be found in [BKJP05, BC03].

3.1 Selection and Manipulation

One of the most fundamental tasks in a physical as well as in a Virtual Environment is the efficient selection and manipulation of objects. This is often a precondition for other interaction techniques like system control or navigation.

3.1.1 Selection and Manipulation Tasks

The efficiency of selection and manipulation techniques highly depends on the tasks which they are applied to. For example, the manipulating techniques used in this thesis are suited for assembly based modeling, but may be rather unusable for a surgical simulation. Thus, it is important to clearly determine the area of use for interaction techniques. All manipulation tasks used within this thesis are spatial rigid object manipulation tasks.

Canonical Task Decomposition

Assuming that every complex manipulation task is composed out of several basic tasks, manipulation techniques can be designed and evaluated for each of the basic tasks. Most direct three-dimensional manipulation techniques try to mimic techniques that can be found in the real world. A spatial rigid object manipulation in the physical world can be seen as a sequence of tasks: grab an object, move it to the desired position and manipulate

its orientation. Therefore, a three-dimensional manipulation technique can be decomposed into the following three canonical tasks:

Selection Select an object or a set of objects out of all available objects. This is also called target acquisition task. It corresponds to the real world action of grabbing an object or a set of objects with one or two hands.

Positioning Move the object to a desired position. The real world counterpart is the movement of an object from a starting position to its target position.

Rotation Change the orientation of an object. In the real world, this task relates to a rotation of an object from a starting orientation to a target orientation.

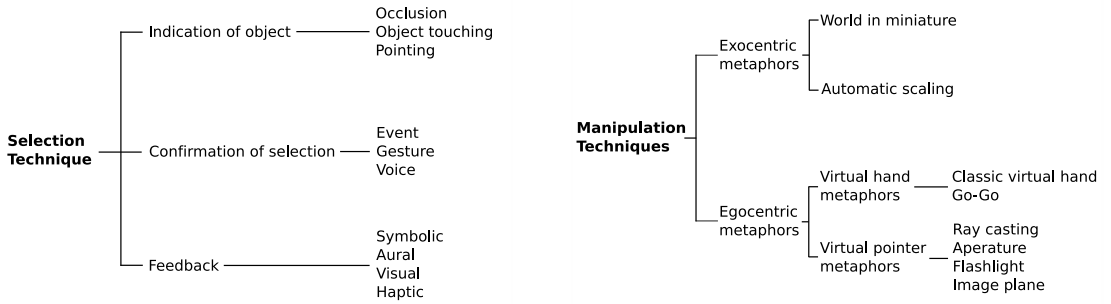
Classification of Manipulation Techniques

Many manipulation techniques are interrelated. Therefore, it is useful to classify them according to their shared features. The following paragraphs will briefly describe three main criteria that can be used for a classification.

Isomorphism Manipulation tasks that have a geometrical one to one correspondence mapping between the hand motion in the real world and the hand motion in the virtual world are called *isomorphic*. Techniques satisfying this isomorphic property often provide a natural feeling and are easy to understand, but they also have some important drawbacks. Isomorphic mappings are often inefficient due to the physical limitation of the human physics. E.g. the arm length of a human is strictly limiting the interaction range. Further problems are constraints of the input device like its tracking range.

Another approach is to divert from imitating the real world and design interaction techniques that are specifically tailored for the use with a specific application. Such techniques are called *nonisomorphic*. They provide magic tools like infinite rays or rubber arms. The goal is to allow a user to manipulate objects in a different way than in the real world and at the same time provide a high usability and performance.

The use of isomorphic or nonisomorphic techniques depends on the requirements of an application. If strict realism is needed, isomorphic techniques should be used, otherwise nonisomorphic techniques can increase the efficiency for particular applications. For a more detailed description along with a user study comparing isomorphic and nonisomorphic interaction techniques look at [BH97].



(a) Selection techniques classified by task decomposition.

(b) Manipulaiton techniques classified by their underlying metaphors.

Figure 3.1: Diagram (a) shows a classification of selection techniques by task decomposition (proposed by [BKJP05]). Diagram (b) shows a classification of manipulation techniques according to their metaphor (proposed by [PIWB98]).

Task Decomposition A complex selection or manipulation technique consists of several basic components, which are all responsible for a specific subtask of the technique. Common basic components of a selection task are shown in figure 3.1a. Every selection technique should provide a user with an indication how to select an object and how to confirm the selection. Each of these components should also give visual, haptic or audio feedback to assist a user. These building blocks can be seen as single suboperations of a complete task. This approach allows the definition of a *construction space*, containing all basic operations. When designing new interaction techniques, they do not have to be completely developed from scratch, because they can be constructed by combing already defined single suboperations. In case the construction space does not contain all suitable basic operations needed for the desired interaction technique, new elements must be added. These elements can be reused at a later time for other interaction techniques. More information about task decomposition can be found in [BJH99].

Interaction Metaphor Another classification of selection and manipulation techniques is based on their underlying metaphors. The use of metaphors is assisting a user to build a mental model about the possibilities and the restrictions of an interaction technique.

Figure 3.1b shows a classification of manipulation techniques according to their basic interaction metaphors (proposed in [PIWB98]). All techniques are first divided into *exocentric* and *egocentric* techniques. When using exocentric techniques, a user interacts with the Virtual Environment from the outside (god's view). An example for this metaphor is the so-called *World in Miniature* (a user interacts with a small copy of the whole environment) technique [SCP95]. Egocentric interaction is more common for Virtual Environments since a user is interacting from within the environment (body centered). This category is further divided into *virtual hand* and *virtual pointer* metaphors. By using virtual hand techniques, a user grabs objects by touching them with a virtual hand. Opposed to this,

in the virtual pointer metaphor a user interacts with the objects by pointing at them.

3.1.2 Techniques

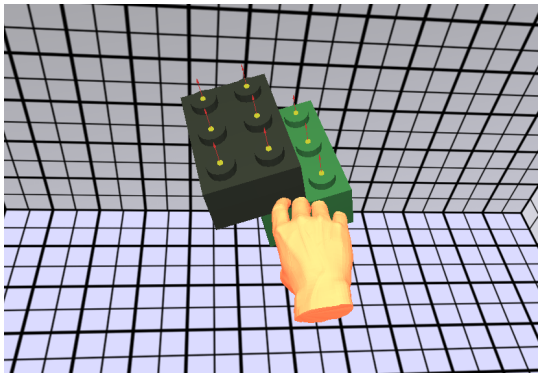
Some of the most important manipulation techniques with respect to this thesis are described in the following section, grouped by their underlying interaction metaphors. Since numerous variations of each individual technique exist, only the fundamental principals needed for the understanding are described here. For a more comprehensive description, the reader is referred to [BKJP05].

Virtual Hand Metaphors

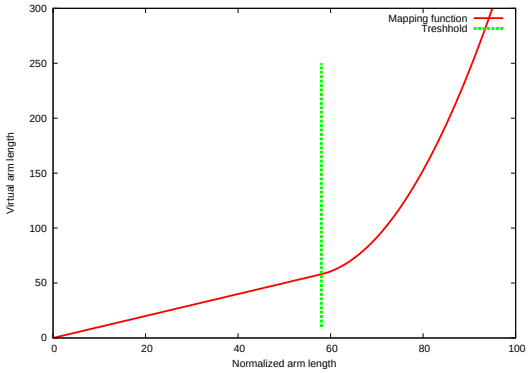
The most intuitive class of manipulation techniques is the class of virtual hand techniques, since a user can directly select and manipulate objects with her hands. Normally, a three-dimensional cursor, often a model of a human hand, is shown at the current location of a user (in most cases the position of a user's hand in the real world). The movement of the user's hand is now coupled to the movement of the three-dimensional cursor (virtual hand) in the Virtual Environment. In order to select an object, a user intersects the virtual hand with the desired object and conforms the selection by activating a trigger (e.g. a button of the input device). Most virtual hand techniques differ in the mapping of the position and orientation of the input devices to the virtual hand.

Simple Virtual Hand This technique uses a direct mapping of the real hand movement to the virtual hand and is the most natural interaction technique. In most cases the real world position is linearly scaled to establish a direct connection from the input device coordinate system to the coordinate system of the Virtual Environment. Figure 3.2a shows an example of a simple virtual hand technique. Furthermore, this technique is an isomorphic interaction technique and a direct emulation of the real world interaction between a real hand and real objects. The biggest drawback of this technique is the limitation of the interaction range to the arm length of a user. Thus, it is cumbersome to select an object out of range, because a user has to move physically inside the Virtual Environment or apply travel techniques.

Go-Go Interaction This interaction technique is an extension of the simple virtual hand technique by using a nonlinear mapping of the real hand movement to the virtual hand movement and was introduced by [PBWI96] in 1996. Here, the interaction range of a user can be dynamically scaled in order to select objects that are far away from a user. The criteria for extending the interaction range is the pose of a user's arm. If the arm is bent and therefore the hand is close to the body, the mapping between the real world



(a) A virtual hand is grabbing two assembled lego bricks.



(b) Nonlinear mapping function used by the Go-Go interaction technique to extend the interaction range.

Figure 3.2: Virtual hand interaction technique (a) and a nonlinear mapping function (b) used to scale the interaction range in the Go-Go interaction technique [PBWI96] .

movements and the virtual hand movement is linear. For reaching a distant object, a user simply stretches her arm. Thus, the distance between the hand and the body is increasing and above a certain threshold, a nonlinear mapping will be applied.

A common way to calculate the mapping function is to transform the tracked hand position into a user centered polar coordinate system. The length of the virtual hand is then calculated by a nonlinear mapping function F :

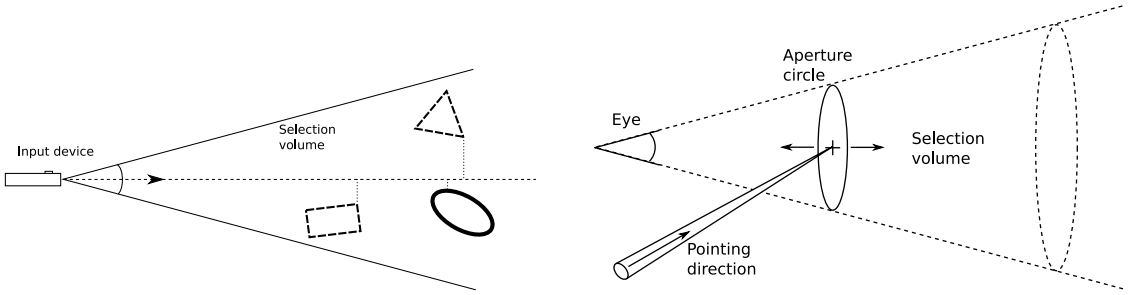
$$F(r_r) = \begin{cases} r_r & r_r \leq \delta \\ r_r + \alpha(r_r - \delta)^2 & r_r > \delta \end{cases}, \tag{3.1}$$

where r_r denotes the length of vector $\vec{r_r}$ pointing from the body center to the hand position. δ is a threshold indicating when the mapping function should switch from a linear to a quadratic mapping. α is a scale factor of the quadratic part of F. Figure 3.2b shows this nonlinear mapping function.

Virtual Pointer Metaphors

Another class of manipulation techniques is based on the pointer metaphor. The underlying idea of these techniques is that a user can easily select and manipulate objects far out of reach by simply pointing at them. When the direction vector of the pointing devices intersects a virtual object, a user can conform the selection using a trigger mechanism and the object is attached to the end of the point vector for further manipulation.

User studies have shown that pointing techniques for selection tasks result in better performance than virtual hand based techniques because pointing requires less movement



(a) Basic flashlight selection. The ellipse will be selected because it is the closest object to the center line of the cone.

(b) Aperture selection technique. A movement of the input device and therefore a movement of the aperture will cause a resize of the conic selection volume.

Figure 3.3: Illustration of the general flashlight selection technique (a) and the extended aperture selection technique (b).

[BJH99].

Ray Casting A virtual ray, often call *Pick Ray*, is attached to the tracked input device. This virtual ray can be seen as an arm extension. The position and orientation of the ray are calculated out of the position and orientation of the input device. For visualization purposes the ray is often attached directly to the virtual hand.

If the pick ray intersects more than one object, different choices for the selection can be made depending on the application. For some applications it might only be suitable to select the first intersected object, whereas it could be useful for other applications to select all intersected objects. Ray casting is a powerful selection technique unless the objects are very small or far away, since a high angular precision is needed for pointing at small objects and in addition to this the amplification of undesired angular hand movement is increasing with the distance.

Flashlight and Aperture Techniques The idea of flashlight and aperture techniques is to have a soft selection mechanism that does not depend on such a high accuracy as the simple ray casting technique. The pointing direction is calculated in the same way as for the simple ray casting techniques. However, the pick ray is replaced with a conical selection volume with its apex at the input device. Because of this conic volume, these techniques are often called flashlight techniques. All objects that are inside of the selection volume are selected. Therefore, it is easy to select a small object even at a far distance. The problem of these techniques is their ambiguity in case more than one object falls into the selection volume. A common heuristic for resolving this ambiguity is to select the closest object to the center line of the selection volume. If multiple objects share the same distance, the closest of these objects with respect to the input device is selected. An illustration of this heuristic is shown in figure 3.3a.

An extension of the flashlight technique is the so-called aperture technique [FHZ96]. It allows the dynamic modification of the selection volume. The pointing direction is defined by the position of a user's viewpoint and the position of the hand sensor, which is used as an aperture (see figure 3.3b). A user can dynamically control the opening angle of the conic selection volume by moving the hand sensor closer or farther away from the body. In addition to the heuristic described in the previous paragraph for resolving ambiguities when multiple objects are selected, the pointing direction of the input device can also be considered.

3.2 System Control

In every non trivial application, a user must have a possibility to send commands to the application in order to change the interaction mode or the state of the system. In two-dimensional desktop based environments a lot of work has been done for developing system control techniques (like pull-down menus, toolboxes, radio buttons, etc...). The most common technique used in desktop systems is the Windows, Icons, Menus and Pointers (*WIMP*) metaphor. However, the direct transfer of WIMP techniques from a two-dimensional environment into a three-dimensional immersive environment is not always the best choice, because three-dimensional interaction differs significantly from desktop based interaction.

3.2.1 System Control Tasks

Despite the fact that the real work of an application is done by tasks like selection, manipulation and symbolic input handling, system control tasks can be seen as the glue that allows a user to control the interaction flow of an application.

In general, a system control task issues a command to:

1. execute a specific system function, like removing a virtual object from the scene.
2. change the interaction mode, such as switching from a drag tool to a rotation tool.
3. change the state of the system, e.g. activating a specific workspace.

Since the issuing of a command is closely related to selection and manipulation techniques, like pressing a button, or select an object from a list, there is certainly an overlap with the interaction techniques described in section 3.1.

Most system control techniques are based upon a few interaction metaphors and their combinations. In figure 3.4a the classification proposed by [BKJP05] is shown.

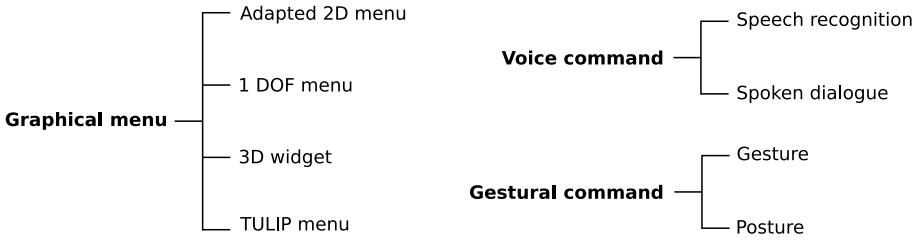


Figure 3.4: Classification of system control techniques proposed by [BKJP05].

3.2.2 Techniques

This section describes the three most common metaphors, graphical menus, voice commands and gesture commands, used for system control techniques. The focus lies on graphical menus because they are used in this thesis. The list of described techniques is not meant to be complete but should rather be seen as a starting point. For a more detailed and comprehensive collection of system control metaphors look at [BKJP05].

Graphical Menus

Graphical menus for a three-dimensional application can be seen as the equivalent of common two-dimensional menus that have been proven to be a successful system control mechanism for desktop systems.

The placement of graphical menus is an important property and directly influences their accessibility and their possible occlusion effects in the environment. In [FMS93], Feiner et al. are proposing four different positioning heuristics for graphical menus.

- World-referenced menus are placed at a fixed position in the virtual world.
- Object-referenced menus are directly attached to objects in the three-dimensional scene. This kind of positioning is not suited for most general purpose menus, but is very useful to display collocated three-dimensional widgets.
- Body-referenced menus, like a virtual tool belt, are providing a strong spatial reference cue. A user study [MFPBS97] has shown that the user's proprioceptive sense (relative locations of parts of the body) can improve the menu usage. Some body-referenced menus even allow eye-off usage, since a user knows exactly where the menu is located.
- Device-referenced menus are always displayed at the same position with respect to the display device. This kind of positioning provides a user with a strong spatial reference. E.g. for some applications it can be useful to display the menus always at the border of the display.

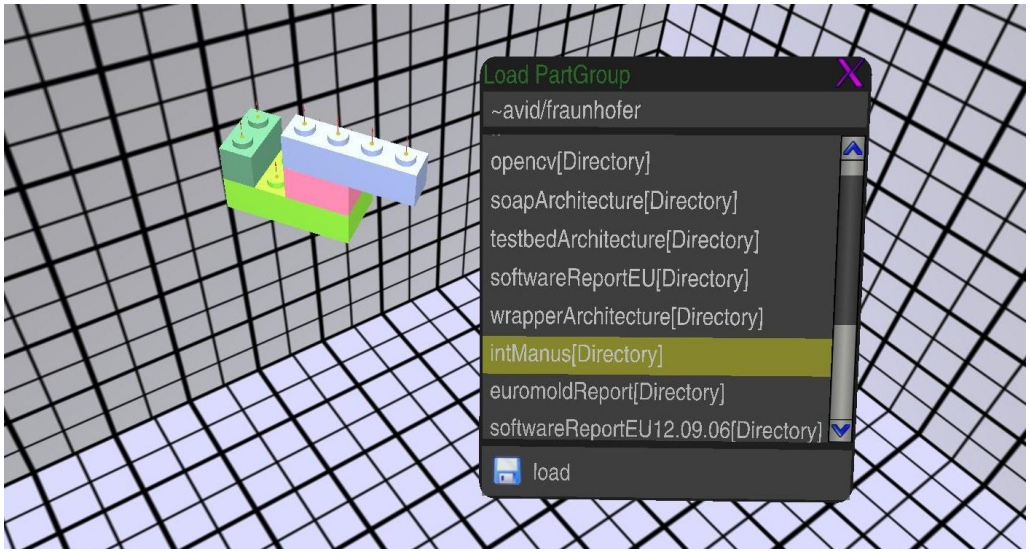


Figure 3.5: This adapted two-dimensional file dialog is directly shown in the Virtual Environment and is used to load models.

Adapted two-dimensional menus Most menus used for three-dimensional system control are simple adaptions of established and well known two-dimensional menus. These menus are handling the user input the same way as their two-dimensional counter parts. Common examples are pop-up menus, scroll areas, floating menus or toolbars. Figure 3.5 shows an adapted two-dimensional file dialog used in this thesis.

One of the biggest advantages of adapted two-dimensional menus is the familiarity of their interaction style, since almost every user will instantly recognize the elements of the menu and know how to use them without the need of any explanation.

A drawback of this kind of menu are possible occlusion effects of the environment caused by them. Semitransparent menus can be used to reduce the occlusion, but for some applications this might not be sufficient. Another problem is that the menus are normally designed for a two-dimensional selection technique, whereas normal selection in three-dimensional environments is a three-dimensional task. One way to address this problem is to constrain the degrees of freedom of the input device. E.g. when interacting with a menu, all tracking data except the two-dimensional projection of the tracked point on the menu can be discarded. This reduces the cognitive load of the selection process.

One degree of freedom selection menus A common task in graphical menus is the selection of one element of a menu. This is a one-dimensional task by nature. Thus, it is beneficial for this task to use a one degree of freedom menu instead of an adapted two-dimensional menu. Often a 1-DOF menu is attached to a user's hand. The elements are thereby arranged in a circular pattern. Due to this circular pattern, these menus are called *ring menus*. The selection criteria is the rotation of the user's hand and the selection is

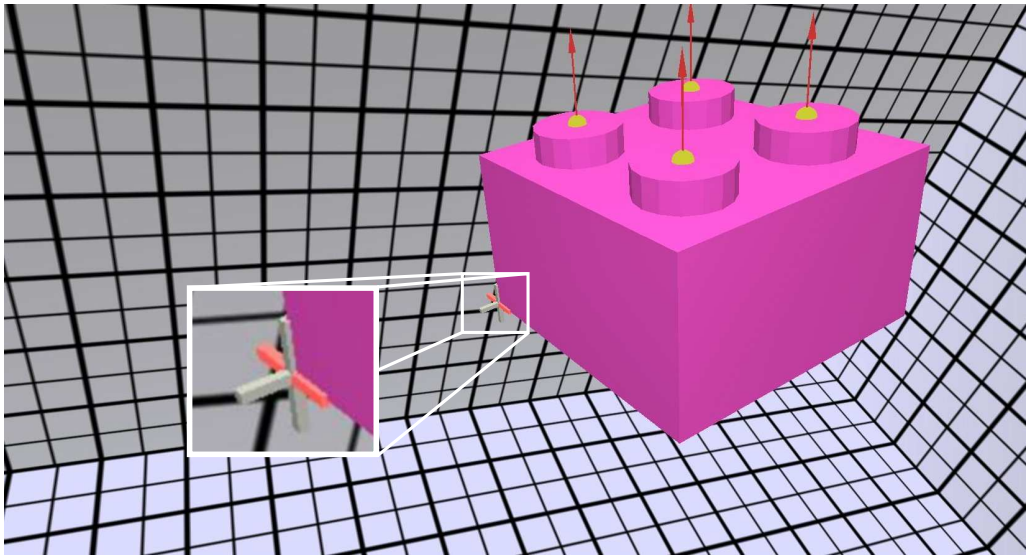


Figure 3.6: A three-dimensional scaling widget. Each of the three axes of the widget corresponds to a scaling operation in one dimension. In the enlarged image the red axis is selected and therefore, the part can be directly scaled in the corresponding dimension.

conformed by a trigger mechanism.

In general, 1-DOF menus are easy to use and for a small number of items an element can be quickly selected. They also provide a strong spatial cue because they are tied to a part of a user's body. Due to this coupling, a user can rapidly access the menu without the need to switch the focus of attention.

3D Widgets This class of menus takes advantage of the extra degree of freedom available in a three-dimensional environment and tries to use this for visualizing complex menu structures along with a high usability. Commonly, 3D widgets are collocated near objects and are therefore highly context sensitive menus. Furthermore, they can be seen as a combination of geometry and behavior. This means that system control functionality is moved from a menu directly onto objects in the environment. An example of a 3D widget for scaling is shown in figure 3.6. Instead of selecting a scale tool from a menu and the successive selection of a part, a user can directly select the scale widget on top of the object and immediately apply the scaling. In general, this is a combination of a system mode change command along with a direct manipulation.

Voice Commands

The use of voice commands as an input for system control techniques can be very powerful, since it is a natural kind of interaction and a user can do this freehand. Furthermore, a user does not have to focus on a menu to issue a command, hence, the interaction flow of

the user is not interrupted. Voice command systems can be roughly divided into simple speech recognition techniques and spoken dialog techniques. Speech recognition techniques are used to issue single commands and spoken dialog techniques provide an interface for spoken interaction between a user and a system (see [McT02] for further details).

The core of both systems is the *speech recognition engine*. Its recognition rate can be influenced by a lot of factors, like the different accents of the speakers or background noise.

In contrast to other system control techniques, speech input initializes, selects and issues a command in one flow. In some applications, it can be useful to couple the initialization with a trigger mechanism like a button (often referred as push-to-talk). Another important property of voice command interfaces is their *invisibility* to a user since a user does normally not know all available commands of the voice interface. Thus, a learning phase or a very precise definition of the interface along with an adequate user feedback is needed.

Multimodal techniques are combinations of different kinds of techniques (like voice commands and graphical menus).

Gestural Commands

One of the first techniques used for system control in Virtual Environments were hand gestures, like in Krueger's Videoplace [KGH85] system. Their advantage is the usage of the hand as a direct input device. This is a very comfortable way for users to issue commands since they have the feeling that they do not use an input device at all. It is important to distinguish between *postures* and *gestures*. A posture is a static gesture like the peace sign (the index and middle finger are forming a V), whereas a gesture describes a hand or finger movement, like waving goodbye.

Like voice commands gestural commands combine the initialization, selection and executing of commands and are also invisible to a user. It is often advisable to use a push-to-gesture approach to make sure a user intends to make a gesture which should be interpreted by the system. The set of used gestures should rather be small and easy to learn, otherwise the interaction will become cumbersome.

3.3 Navigation

The task of navigation refers to the movement in and around a Virtual Environment. Navigation techniques are used in a lot of Virtual Environment applications and should have a low cognitive load, because a user should be able to focus on the application. However, not all Virtual Environment applications need the task of navigation. One

example for this is the developed VEAM system since the modeling space directly coincides with the interaction space of a user.

An extensive and in depth description of various navigation techniques can be found in [BKJP05, BC03].

3.3.1 Navigation Tasks

There are different kinds of three-dimensional navigation tasks. Bowman et al. [BKJP05] classify them into three categories.

Exploration Navigation tasks where the movements of a user have no explicit goal, are called exploration tasks. A user is simply exploring the environment in order to find interesting objects or locations and is building up knowledge about the environment. Since the path of a user is not predefined in a general exploration task, the interaction technique should allow continuous and dynamic control about the movement. However, some applications, do not provide continuous movement control, but rather provide the ability to interrupt a predefined path for a specific period of time. These are mainly applications that have to provide an enjoyable experience in a limited amount of time, like the one described in [PST+96]. All techniques should be easy to use and should have a low cognitive load, because a user should be able to focus on information gathering in the environment.

Search In a search task, a user travels to a specific location within the environment. That does not automatically imply any knowledge about the location of the target. If a user has no knowledge about the position and the path how to get there, the task is called a *naive search* task. In case a user has visited the target location before or has some sort of knowledge about its position, it is denoted as a *primed search* task.

Maneuvering Tasks of this kind are taking place in a local area and are involving short and very precise movements in order to perform a specific task. It can be seen as sort of primed search task, because the target is known. However, navigation techniques suited for a general search task are often too coarse for maneuvering tasks.

4 Interactive 3D Assembly Modeling

Virtual assembly systems using Virtual Environments can be categorized into two main groups [Jun03]: *assembly simulation* and *immersive modeling*.

Most work is centered around assembly simulation, which tries to simulate the actual assembly or manufacturing processes. This involves packaging issues as well as determining required space regions, by calculating assembly paths for all involved parts, generated by the part itself together with the robot arm or the human hand. The goal of the simulation is to calculate an assembly path through space that avoids any unintended collision between all involved parts and results in a cost efficient process. A detailed description of assembly simulation in Virtual Environments can be found in [Zac00].

An immersive assembly based modeling system typically uses CAD *(Computer Aided Design)* models and combines them using different assembly techniques. This is often used to develop new customized products based on combinations of mature product models. Most assembly systems use some sort of snapping mechanism to automatically complete an assembly operation as soon as two parts are moved close together. Some systems are only capable of dealing with specific classes of parts or use other kinds of restrictions to achieve a high usability. Examples of existing immersive assembly based modeling systems can be found in section 4.1.

This chapter covers immersive modeling systems and is divided into eight sections. Concepts of existing immersive assembly systems are discussed in the first section. The concept of the developed immersive modeling system called VEAM *(Virtual Environment Assembly Modeler)* is introduced in the second section and is directly compared to the corresponding concepts of the existing modeling systems. The remaining sections of this chapter describe the core concepts of the developed VEAM system in detail.

4.1 Related Concepts

A lot of work has been done in the field of virtual assembly. This section describes the underlying concepts of four related virtual assembly systems.

Virtual Assembly Design Environment (VADE) [JJW+99] The *VADE* is an engineering application that allows assembly planning and evaluation in a Virtual Environment.

It is tightly coupled with CAD systems. The CAD system is used to generate the geometry of the parts along with the assembly information like part tolerances or connection properties. For each CAD system that should be used, an export and import interface to the VADE system must be implemented. The original VADE system provides an interface for the commercial parametric CAD system *ProEngineer*[1]. The VADE system maintains a direct link to the CAD system at all times in order to access the functionality of the CAD system when needed. After the VADE session the system automatically converts the generated design information and sends it to the CAD system for further processing.

Amongst others, important features of the VADE system are the swept volume generation and the trajectory editing functionality of the assembly path. The generated swept volume is used in combination with a collision detection module to detect positions, at which the assembly path is colliding with the environment. This information is then used to recalculate the trajectory of the assembly path in order to obtain a collision free assembly path.

The system supports both one- and two-handed interaction tools. In the two-handed interaction mode, the non-dominant hand is used to grab the base part (e.g. plate with a hole), whereas the dominant hand manipulates the part to be assembled (e.g. a screw).

The original VADE system runs with a HMD and uses data gloves for the user interaction. A VADE installation at the *National Institute of Standards and Technology*[2] *(NIST)* uses a responsive workbench as visual output device.

Natural and Robust Interaction in Virtual Assembly Simulation [ZR01] The interaction techniques and methods proposed by Zachmann and Rettig are specially designed for natural and robust interaction in assembly simulation systems. In order to increase the intuitivity and the efficiency of such a system, they suggest using multimodal user input namely graphical menus, voice commands and hand gesture recognition.

They are using adapted two-dimensional graphical menus displayed as an overlay on the three-dimensional scene.

One of the key features is the development of a new grasping technique called *natural grasp*. This technique tries to transfer the natural interaction of a real hand with a real part into the interaction in the Virtual Environment. In order to do so, Zachmann and Rettig classify four different kinds of grasps:

1. Precision Grasp: This kind of grasp involves two fingers, usually the thumb and one other finger. As the name implies this is used for high precision tasks.

[1] Further information about the ProEngineer CAD software suite can be found under www.ptc.com
[2] Further information about the National Institute of Standards and Technologies can be found under www.nist.gov

2. Cigarette Grasp: Grasps of this kind involves two neighboring fingers and is most often used to hold long thin objects like a pencil or a cigarette.

3. 3-Point Pinch Grasp: Three fingers are involved in this grasp, whereas on of the used fingers is the thumb. This grasp is fairly solid and allows the rotation of the grabbed object without rotating the hand.

4. Power Grasp: For this kind of grasp the whole hand and especially the palm is used. Grabbed objects are stationary attached to the hand.

Their algorithm then uses a heuristic to recognize the kind of grasp intended by a user to apply to the virtual object. This heuristic depends on the state of the user's hand.

Task-Level Assembly in Virtual Environments [Jun03] One of the main concepts of the immersive assembly modeling framework developed by Jung is a task-level interface which maps high-level user commands to low-level assembly commands. The framework uses connection sensitive regions of parts to establish part connections. These regions are called *ports* and two ports can be connected by applying different *connection types*, like screwing or welding.

Jung introduces a taxonomy with a rich set of predefined ports and connection concepts which capture most of the common assembly tasks. The most high level ports in this taxonomy are the following concepts:

- Extrusion Ports: Ports of this kind model the connection of parts with extrusion geometries. The most common application of extrusion ports is for groove and tongue kind of assembly operations. Extrusion ports are further differentiated into male (e.g. shaft of a screw) and female ports (e.g. hole of a nut), whereas a valid mating must involve a male and a female port. In general, a valid connection between two extrusions ports provides one translational and one rotational degree of freedom.

- Plane Ports: Connections between planar object surfaces are modeled using this kind of port. Plane port connections provide two translational and one rotational degree of freedom. In order to connect two plane ports, they must be coplanar and their extent must overlap.

- Point Ports: They are used to model point like object connections that are allowing no translational and one rotational degree of freedom when two objects are connected. The connection of a steering wheel with the steering rod is an example for a connection by two point ports.

A comprehensive set of algorithms is used in order to transform high-level specifications into low-level commands. These commands include assembly and disassembly operations as well as adjustment operations of already assembled parts. Each of these three operations

needs different parameters, whereas all operations have required and optional parameters. The required parameters are needed for the execution of the operation and the optional parameters can be computed using predefined heuristics in case they are not given. The part connection function needs at least the two parts which should be connected. If no other information is given, the remaining not specified parameters are filled according to a heuristic (e.g. always connect the two closest compatible ports).

The system runs in a surround screen display, where natural language and hand gestures are used for the interaction.

Virtual Environment for Generating Objects (VLEGO) [KTK+97] The VLEGO application is a prototype of an immersive three-dimensional modeling system developed at the *Nara Institute of Science and Technology*[3] *(NAIST)*. It uses toy blocks similar to LEGO[4] bricks as modeling primitives and allows different basic one- and two-handed interaction techniques. Since the modeler is only capable of modeling with these kind of toy blocks, all parts can only be transformed according to strict translational and rotational constraints.

In general, the transformation of each toy block in the system is a four degrees of freedom transformation with an additional grid constraint (4DOFC). A block bound to this constraint can only be moved at discrete positions at the interval of 1 *cm*, which is the interval of the studs of the toy bricks, and in addition its orientation is restricted to 0, 90, 180, 270 rotational degrees around the z-axis (assuming a right handed coordinate system is used). As a result of these constraints, the bricks are aligned with respect to each other at all time during the modeling.

VLEGO provides two different modes for two-handed interaction, namely a *Separative Mode* and a *Cooperative Mode*.

In the separative mode each hand picks a toy block. The distance between the two selected toy blocks must be larger than a specific threshold δ. The system now considers those blocks as unrelated and assumes that a user wants to align each of the two picked blocks independently with a non picked block. In this case the transformation of the two active blocks is a restricted 4DOFC transformation.

If the distance between the two picked blocks is smaller than the threshold δ, the system changes the interaction mode from separative to cooperative mode. In cooperative mode, the system assumes that a user wants to assemble the two picked blocks together. The movement of the block which is picked first (called *base*) is not constrained, however the movement of the other picked block (named *work*) is constrained to a 4DOFC transformation. The constraint of the work part is dynamically determined based on the position

[3] For more information about the Nara Institute of Science and Technology visit: http://www.naist.jp/index_e.html

[4] More information about LEGO can be found under: www.lego.com

and the orientation of the base part.

4.2 Developed Concept

The developed VEAM system is an immersive assembly based modeling system and allows the design and evaluation of new customized products based on combinations of product models.

The modeling primitives used by the VEAM system are predefined parts whose shape equations are not changed during the modeling process. This is especially important with respect to the integration of the system into a manufacturing process chain, since the geometric data of all used parts is already known and the machine specific *computer numerical control*[5] *(CNC)* programs can be precomputed. In case the geometric properties of parts would be modified during the modeling process, an on-the-fly generation of specific CNC programs would be needed. However, this is still problematic and is not supported by the INT-MANUS [FSM+06] manufacturing process chain the VEAM was integrated into. The specification for the modeling primitives used by the VEAM system is described in section 4.3.

The VEAM system does not use the geometric representation of parts as a criteria for possible part connections, because every polygonal geometric model is an approximation of the real geometry and therefore does not represent the real part correctly. In case the model is only assembled for visualization purposes, this causes no problems. However, accuracy problems will occur as soon as the generated model is used in a manufacturing process which exclusively relies on these connection information. To reduce this accuracy problem, more detailed geometric models of the parts can be used. Nevertheless, this can lead to a violation of the real time rendering constraints, especially in case commodity hardware is used or the model consists of a large number of parts.

Since the VEAM system is used in an integrated manufacturing process chain and approximations of the part geometries are used to ensure real time rendering, the connectivity information of parts is not based on their geometry, but rather on constructs called *Handles*. Section 4.4 describes the concept and the usage of handles and introduces various handle types.

A key concept of the VEAM system is the product configuration grammar which is used to define all possible part connections. A description and specification of the grammar is given in chapter 4.5.

During the modeling process, the connectivity information of all interconnected parts must be stored, since this is a necessary information for various operations such as the

[5] A computerized numerical control is a controller that drives machine tools

part disassembly operation or the generation of the assembly graph. The VEAM system stores this information in a data structure called *ConnectionGraph*. The concept and the algorithms of the *ConnectionGraph* are described in detail in section 4.6.

The part assembly and disassembly algorithms of the VEAM system are solely based on the product configuration grammar and on the *Handle* concept. In most cases, assembly situations are ambiguous because the involved parts can be connected in several ways. The connection algorithms assist users to solve these ambiguous assembly situations by applying different heuristics. In section 4.7 the part assembly and disassembly algorithms are described in detail.

The VEAM system provides an import and export functionality for assembled models. This is particularly important for the integration of the system into an integrated manufacturing process chain and is a requirement for creating extensive models. The import and export functionality of the VEAM system is described in section 4.8.

The underlying concepts of the developed VEAM system differs in various aspects from the four related immersive modeling systems described in the section 4.1.

Opposed to the two systems described in [Jun03, KTK+97], the VEAM system does not use the geometric representation of parts as a criteria for possible part connections.

The Virtual Environment for Generating Objects [KTK+97] uses toy blocks similar to LEGO bricks as modeling primitives. On the basis of the regular structure of these blocks and their studs, the system a priori restricts possible block connections to discrete positions and orientations. Due to this restriction, the system is only capable of using these specific blocks as modeling primitives. Although similar blocks are used exemplarily as modeling entities in this thesis, the VEAM system is not restricted to any specific class of primitives.

The VEAM system provides an extensive serialization and deserialization functionality for assembled models. While the VADE [JJW+99] system uses the directly connected CAD system for this, both modeling systems [Jun03, KTK+97] do not provide any information about a possible model import and export functionality.

4.3 Modeling Primitives

The goal of the VEAM system is to generate new customized products based on combinations of mature product parts. Therefore, the primitives used by this modeler are predefined parts. In the following sections the term part is used as a synonym for primitive.

The developed part assembly algorithms (see section 4.7) of the VEAM system do not rely on the geometric representation of parts and therefore geometric approximations of

the parts can be used for the visualization. This is an important property to ensure the real time capability of the system, especially in case commodity hardware is used or the modeled product is composed of a large number of parts.

Due to the use of geometric approximations and the integration in the manufacturing process chain, no changes of the geometric representation of the primitives are made by the system.

The system is capable of handling an arbitrary number of primitives and is not limited to any specific class of parts. Parts can be added to the system dynamically at runtime.

Each primitive does not only have a geometric representation through a polygonal model, but also the following properties:

1. Connectivity information: The connectivity information of a part is encoded in handles (a description of handles is given in section 4.4) which are assigned to the part. Each part can have an arbitrary number of handles attached.

2. Material properties: A part can have different material properties. These include color information as well as the kind of material, such as iron or plastic, which is used to produce the physical part.

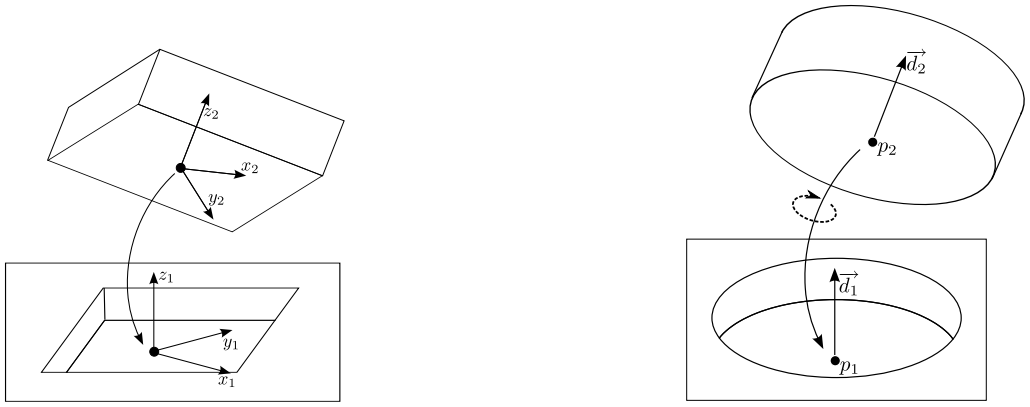
3. Production details: This describes the time and costs needed for the production of a part. This information is used to estimate the costs and production time of the modeled product. The final production time and costs will be calculated by the manufacturing process chain.

4.4 Handles

The connectivity information of a part is not based on its geometric representation or on specific object classes. The VEAM system uses a grammar-based model to encode the connection information of parts. The core of this grammar model is the concept of so-called *Handles*.

4.4.1 General Definition

A handle is an attribute of a part and specifies its connection possibilities and the exact geometric arrangements of potential connections with other parts. To describe these geometric arrangements, handles use geometric primitives, like points, vectors and coordinate systems in conjunction with various connection semantics. The used geometric primitives and connection semantics vary for all handle types. Each handle has its own coordinate



(a) *ExactMatch* (b) *AxisMatch*

Figure 4.1: In (a) a part connection using two *ExactMatch* handles is illustrated. For a part connection with two *ExactMatch* handles, no degree of freedom is left. In (b) a part connection using two *AxisMatch* handles is shown. Such a connection provides one rotational degree of freedom and therefore, the angle at which the cylindrical part is connected is not specified.

system, which is used for the placement of the geometric primitives. The coordinate system of a handle must be defined in the local reference frame of a part.

In general, handles are used to define the set of models possible to construct out of a set of given parts and support a user during the assembly process of two parts.

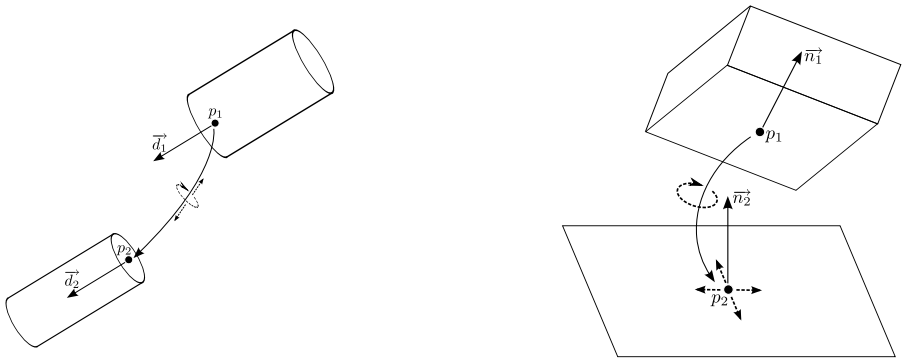
4.4.2 Handle Types

The system supports different connection semantics by providing different types of handles. Currently, the VEAM system provides four different handle types. These handle types do not cover all possible connection semantics, but are sufficient for most uses. In case additional connection semantics are needed, the system provides functionality to add new handle types.

ExactMatch

An *ExactMatch* handle corresponds to the connection semantics of two parts exactly fitting into each other. Parts connected with two *ExactMatch* handles do not provide any degree of freedom for the transformation of the connection.

The geometric representation of this handle type is a right-handed orthonormal coordinate system. The handle is defined by a point representing the origin of the coordinate system and two linearly independent vectors. Since a right-handed coordinate system is assumed, a distinct vector which is orthogonal to the two provided vectors can be computed. In figure 4.1a a connection of two parts using a pair of *ExactMatch* handles is illustrated.



(a) *AxisSlide* (b) *SurfaceMatch*

Figure 4.2: In (a) two parts are connected using two *AxisSlide* handles. This connection allows one translational and one rotational degree of freedom. Thus, the two cylinders can be fit into each other and can be rotated around the common axis during the preliminary connection. In (b) a part connection using two *SurfaceMatch* handles is shown. The relative location and orientation of the two parts are not specified by this connection.

AxisMatch

The geometric representation of an *AxisMatch* handle is a point and a vector. This handle type represents a connection of two parts at one common point, whereas both vectors must coincide. This connection induces no translational, but one rotational degree of freedom around the common axis of the two involved *AxisMatch* handles. In figure 4.1b the connection semantic of an *AxisMatch* handle is illustrated.

AxisSlide

Similar to the *AxisMatch* handle, the geometric representation of an *AxisSlide* handle consists of a point and a vector. However, the connection semantic is different. It models connections of two parts assembled in a groove and tongue fashion. This kind of connection provides one translational and one rotational degree of freedom. Therefore, both parts can be rotated around and translated along the common vector. A sample part connection using two *AxisSlide* handles is shown in figure 4.2a.

SurfaceMatch

A *SurfaceMatch* handle models the connection of two planar object surfaces. Connections based on two *SurfaceMatch* handles offer two translational and one rotational degree of freedom. Thus, both parts can be moved and rotated against each other during the handle connection phase. The geometric representation of a *SurfaceMatch* handle is a

planar rectangular surface. Figure 4.2b shows an exemplary connection of two parts using *SurfaceMatch* handles.

4.4.3 Connecting Handles

Two parts can be connected if there is at least one pair of matching handles amongst each part. For a matching handle pair, the types of the two involved handles must be identical and the product configuration grammar must define a matching relationship between them. A detailed description of the product configuration grammar is given in section 4.5.

In case several possibilities for a connection of two parts exists, each of these connections must have at least one corresponding pair of handles.

After the connection of two parts, a constrained transformation, determined by the types of the involved handles can be applied. In case a connection of two parts involves more than one pair of handles, the calculation of the degrees of freedom left for this constricted transformation is complex, since all involved handle pairs must be considered when calculating the possible transformation. An algorithm to address this problem is described in section 4.7.

4.5 Product Configuration Grammar

A grammar language, called product configuration grammar, is used to define all possible part connections. Potential part connections are not described explicitly in this language, because this would result in an immense set of possible part connections even for a relatively small set of parts. Therefore, a grammar is used to calculate possible connections at runtime.

The grammar provides two methods to describe matching handle relations:

1. An explicit specification of a pairwise matching relationship between two handles.
2. The definition of matching handle relationships based on tags.

The explicit specification of pairwise matching relationships of handles is only reasonable for a small number of parts with few handles assigned.

A more general approach is the description of matching handles based on tags and their compatibility. Each handle can have an arbitrary number of tags assigned. Possible handle connections are described in terms of collections of compatible handle tags. This tag based handle matching technique provides a flexible and easy to use functionality, which describes most of the occurring part connections in an assembly based modeling system.

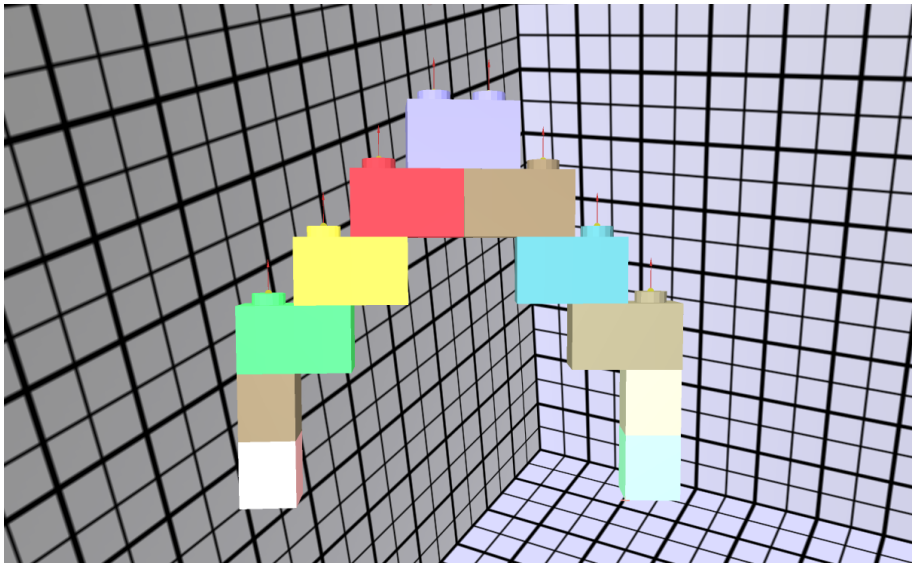


Figure 4.3: Model created on the basis of the product configuration grammar defined in listing 4.1.

In addition to the matching handle relation functionality, the grammar allows the definition of additional restrictions, such as the relation of compatible parts. This defines a compatibility relation on part level and is applied prior to the handle relations.

A convenient way to specify this product configuration grammar is to use an XML based description. The developed VEAM system provides an import and automatic validation functionality for these grammar files. A detailed description of the grammar parser and the validation module can be found in section 6.1.2.

A simple example of the XML based product configuration grammar description for two LEGO parts is shown in listing 4.1. The parts associated with the grammar are described in a `PartCollection`. This collection can contain an arbitrary number of `Part` objects, which correspond to the actual parts used in the VEAM system. All handles are defined in a `HandleCollection`. This collection can contain descriptions of all handle types known by the system and not only *AxisMatch* handles like in the shown listing 4.1. Tags are assigned to handles by adding them to a `HandleTagCollection`. Each handle can be added to multiple `HandleTagCollections`. A matching relationship between `HandleTagCollection` objects and therefore for all handles contained by these collections, is established by adding them to a `CompatibleHandleTagCollection`.

The model shown in figure 4.3 is modeled using the XML grammar file shown in listing 4.1.

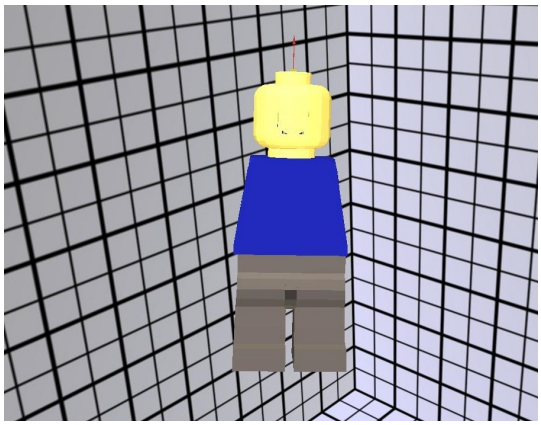
Listing 4.1: Sample Product Configuration Grammar

```
<ProductConfiguration id="conf1" name="Sample␣grammar">
```

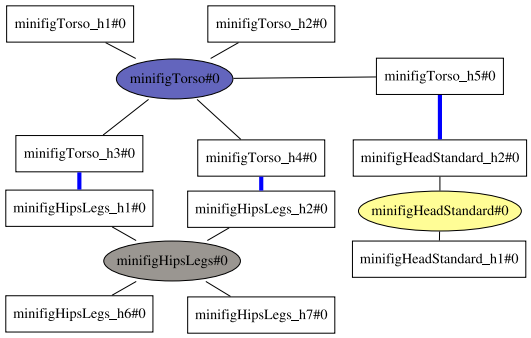
```
  <PartCollection>
    <Part fileName="parts/bricks/brick1x1.obj" id="brick1x1" desc="Lego 1x1 brick"/>
    <Part fileName="parts/bricks/brick1x2.obj" id="brick1x2" desc="Lego 1x2 brick"/>
  </PartCollection>

  <HandleCollection>
    <!-- 1x1 brick -->
    <AxisMatchHandle desc="AMH" id="brick1x1_h1" partId="brick1x1"
      xPos="0" yPos="0" zPos="0" dXDir="0.0" dYDir="0.0" dZDir="1.0" />
    <AxisMatchHandle desc="AMH" id="brick1x1_h2" partId="brick1x1"
      xPos="0" yPos="0" zPos="0.0" dXDir="0.0" dYDir="0.0" dZDir="1.0" />
    <!-- 1x2 brick -->
    <AxisMatchHandle desc="AMH" id="brick1x2_h1" partId="brick1x2"
      xPos="0" yPos="0" zPos="0" dXDir="0.0" dYDir="0.0" dZDir="1.0" />
    <AxisMatchHandle desc="AMH" id="brick1x2_h2" partId="brick1x2"
      xPos="0" yPos="0" zPos="0.0" dXDir="0.0" dYDir="0.0" dZDir="1.0" />
    <AxisMatchHandle desc="AMH" id="brick1x2_h3" partId="brick1x2"
      xPos="0" yPos="0" zPos="0" dXDir="0.0" dYDir="0.0" dZDir="1.0" />
    <AxisMatchHandle desc="AMH" id="brick1x2_h4" partId="brick1x2"
      xPos="0" yPos="0" zPos="0.0" dXDir="0.0" dYDir="0.0" dZDir="1.0" />
  </HandleCollection>

  <HandleTagCollections>
    <HandleTagCollection id="topBrickHandles">
      <HandleId id="brick1x1_h1" />
      <HandleId id="brick1x2_h1" />
      <HandleId id="brick1x2_h3" />
    </HandleTagCollection>
    <HandleTagCollection id="bottomBrickHandles">
      <HandleId id="brick1x1_h2" />
      <HandleId id="brick1x2_h2" />
      <HandleId id="brick1x2_h4" />
    </HandleTagCollection>
  </HandleTagCollections>

  <CompatibleHandleTagCollections>
    <CompatibleHandleTagCollection id="BrickConnection">
      <CompatibleHandleTag id="topBrickHandles" />
      <CompatibleHandleTag id="bottomBrickHandles" />
    </CompatibleHandleTagCollection>
  </CompatibleHandleTagCollections>

</ProductConfiguration>
```

4.6 ConnectionGraph

The VEAM system uses a data structure called *ConnectionGraph* to store all connection properties of a part group. A part group is a set of interconnected parts. Each part group has its own *ConnectionGraph*. The *ConnectionGraph* internally uses an undirected cyclic graph to store the connection information of assembly models. The vertices of the graph correspond to parts or handles and the edges represent a connection relationship between the adjacent vertices.



(a) Assembled lego mini figure



(b) *ConnectionGraph* of the assembled lego mini figure shown in (a). Part vertices are drawing as ellipses and handle vertices are drawn as rectangles.

Figure 4.4: Simple model and its corresponding *ConnectionGraph*.

Vertices corresponding to parts are never adjacent, because parts are not connected directly but by their handles. All handles of a part are represented as adjacent vertices of the part vertex. Edges between two handle vertices represent handle connections and therefore the connection of the parts of these handles.

Figure 4.4a shows a simple assembled model. The corresponding connection graph is shown in figure 4.4b. All edges of this *ConnectionGraph* connecting two handle vertices are drawn blue. Each of these vertices represent a connected handle pair.

One important property of the *ConnectionGraph* is the interconnection property. This property states that a path connecting all part vertices exists. In other words, the graph must be connected with respect to all part vertices. A *ConnectionGraph* that fulfills this property, ensures that all parts of the part group are interconnected and therefore build a valid assembly model.

The interconnection property is maintained for all part assembly operations, since the part and the handle vertices of the newly assembled part are added to the *ConnectionGraph* of the part group to which the part is assembled and an edge is inserted in the *ConnectionGraph* for each connected handle pair.

Violations of the interconnection property can occur in case a part is disassembled from a part group, because its part and handle vertices together with all outgoing edges are removed from the *ConnectionGraph*. This can result in a breakup of the *ConnectionGraph* into several connected components, whereas each of these components then corresponds to an individual part group.

Algorithm 1 Pseudocode to calculate the connected components of an undirected graph using a depth-first search traversal.

```
CONNECTED-COMPONENTS(G)
 1  //Loop over all vertices of graph G
 2  for each vertex u ∈ V[G]
 3        do
 4           //Mark vertex u as not visited by coloring it white
 5           color[u] ←WHITE
 6           //Set component-number of vertex u to 0.
 7           //All vertices initially belong to the same component
 8           component[u] ← 0
 9  //Loop over all vertices of graph G
10  for each vertex u ∈ V[G]
11        do
12           //Check if vertex u is not yet visited
13           if color[u] = WHITE
14           //Depth-first search traversal starting from vertex u
15              then CONNECTED-COMPONENTS-VISIT(u)
16                   //Increment the global component-number
17                   componentNum + +
CONNECTED-COMPONENTS-VISIT(u, componentNum)
 1  //During the traversal all vertices are colored gray
 2  color[u] ←GRAY
 3  //Set the component-number of u
 4  component[u] ← componentNum
 5  //Loop over all adjacent vertices of u
 6  for each v ∈ Adj[u]
 7        do
 8           //Check if vertex v is not yet visited
 9           if color[v] = WHITE
10           //Set the parent vertex of v to u
11              then p[v] ← u
12                   //Recursively visit vertex v
13                   CONNECTED-COMPONENTS-VISIT(v, componentNum)
14  //Mark vertex u as visited
15  color[u] ← BLACK
```

The algorithm to calculate the connected components of the *ConnectionGraph* is given in Algorithm 1. For the traversal of the vertices of the graph, either a depth-first or a breadth-first search can be used. This algorithm uses a depth-first search traversal. At the beginning, the component-number field of each vertex and a global component-number are initialized with zero. The traversal of the vertices then starts from a node u where the component-number of all visited vertices are set to the actual global component-number. After the initial traversal, the global component-number is increased and the not yet visited vertices are traversed and their component-number is again set to the actual global component-number. This is repeated until all vertices of the graph are visited. The

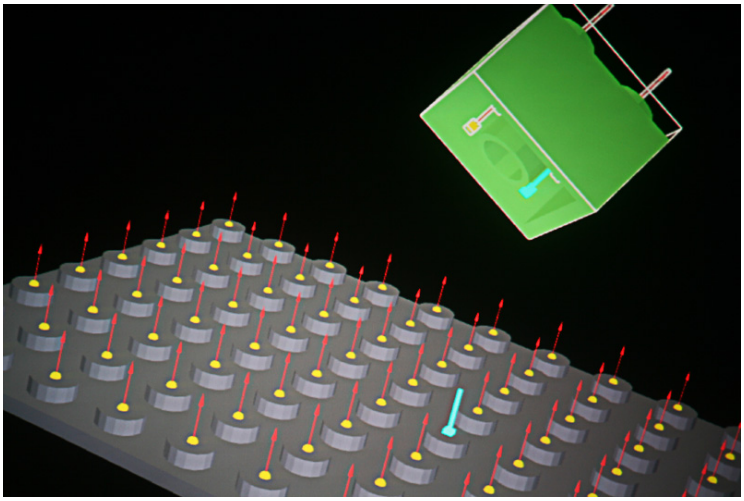


Figure 4.5: Ambiguity during the part connection process. The green brick can be connected with the gray plate in a variety of ways. The part assembly algorithm uses different heuristics to calculate the closest compatible handle pair for ambiguous assembly situations. The closest compatible handle pair is highlighted in order to provide a visual feedback for the user.

component-number of each vertex can now be used to identify the connected component to which the vertex belongs.

An in-depth description of graph algorithms can be found in [CLRS01].

4.7 Assembly Modeling Algorithms

This section describes the developed part assembly and disassembly algorithms of the VEAM system. The part assembly algorithm encapsulates the functionality for assembling parts and assists a user to solve ambiguous assembly situations. The part disassembly algorithm manages the disassembly of a part from a part group and deals with potential breakups of the part group caused by the removal of the part.

4.7.1 Part Assembly

Assembling two parts requires at least one pair of matching handles. Whether two handles are matching is defined by the product configuration grammar described in section 4.5.

However, in most cases the matching process of handles is ambiguous, because parts often have several handles to which another part is compatible. In figure 4.5 the small 2x2 brick can be connected to the large plate in a variety of ways, because each stud of the plate and each fitting hole of the brick are instrumented with compatible *AxisMatch* handles (see [FWd+08]).

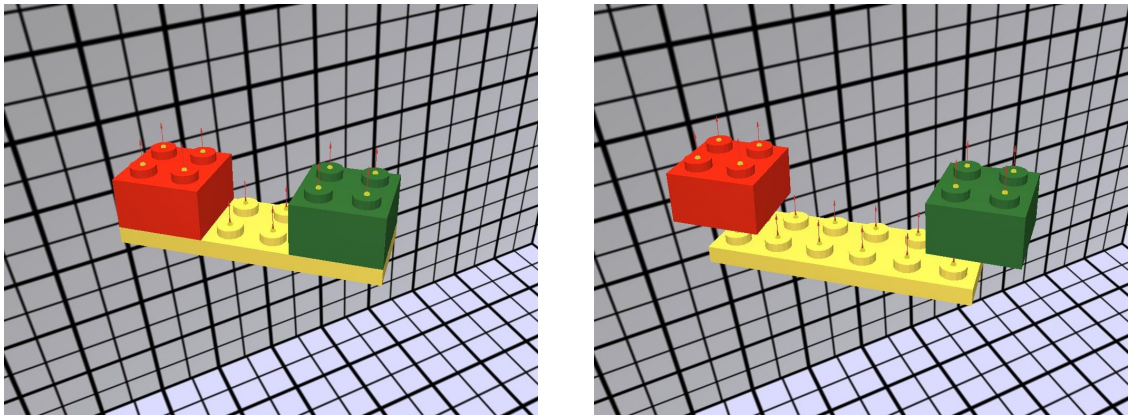
Algorithm 2 Pseudocode of the part assembly algorithm.

```
AssemblePart(P)
 1  //Calculate list of potential counterparts
 2  PotCParts ← Calculate-Potential-Counterparts()
 3  //Select closest handle pair according to the used distance metric
 4  HPair ← Select-Closest-Handle-Pair(PotCParts)
 5  //Get the counterpart of the handle pair HPair
 6  CPart ← Get-Counter-Part(HPair)
 7  //Check if distance of handle pair HPair is smaller than threshold δ
 8  if δ > Distance(HPair)
 9     then
10          //Snap part P to part C according to the handle pair HPair
11          Snap(P, C, HPair)
12          //Check if the involved handle pair allows a restricted transformation
13          DofTrans ← DOF-Trans-Mode(HPair)
14          if DofTrans ≠ nil
15            then
16                 //The user transforms part P according to DofTrans
17                 while Active(DofTrans)
18                      do
19                          //Restrict the user defined transformation according to DofTrans
20                          Trans ← Calculate-Transformation(DofTrans)
21                          //Apply restrict transformation Trans
22                          Apply-Transformation(Trans,P)
23                 //Check if the system can find a valid assembly position of part P
24                 if Find-Valid-Position(P)
25                   then
26                          //Assemble part P and part C
27                          Connect-Parts(P,C)
28            else
29                 //Assemble part P and part C
30                 Connect-Parts(P,C)
```

The part assembly algorithm developed in this thesis does not only assists a user in ambiguous assembly situations, but more important is capable of managing part connections that involve more than one handle pair. This is a complex task, because the calculation of the degrees of freedom left for the transformation of the parts depends on all involved handle types. The pseudocode of the assembly algorithm is given in Algorithm 2.

In order to perform an assembly operation, a user moves part P through the workspace and brings it close to a counterpart C, to which it should be connected. The algorithm calculates the bounding spheres for each part in the workspace. All objects, whose bounding sphere intersect with the bounding sphere of part P, are stored in a list of possible counterparts.

The list of potential counterparts is now searched for a compatible part group. This is



(a) Part group before the disassembly operation.

(b) Part group after the disassembly operation.

Figure 4.6: The disassembly of the yellow bar (a) results in a breakup of the part group (b), since it was the only connection between the red and green brick.

done by iterating over all handles of all possible counterparts and according to a specific heuristic a matching handle pair between part P and counterpart C is selected. Different heuristics, such as minimizing the euclidean or the angular distance, are used to calculate the closest handle pair.

A snapping mechanism, depending on the types of the involved handle pair, is used to align the parts P and C in a preliminary position and orientation.

Depending on the degrees of freedom of the handle connection, the algorithm allows a restricted transformation of the parts.

After a user has positioned part P, using the restricted transformation, the algorithm checks the validity of the connection. In most cases the position is not valid, because a user is not able to position part P with the needed precision. The system then searches the neighborhood for valid positions. The range of the neighborhood search can defined by the user. If the system detects a valid position, the part will be pulled tight to this position, otherwise the two parts will not be connected.

In addition to the involved handles, a connection of two parts often induces secondary handle connections. Thus, the last step of the algorithm is to check for these secondary handle connections and add them to the *ConnectionGraph*.

4.7.2 Part Disassembly

Parts can not only be assembled, but also disassembled. The part disassemble operation involves a removal of handle connections and can therefore result in a breakup of the part group containing the part to be disassembled. This happens if the disassembled part is the only connection between subgroups of interconnected parts. In the model shown in

Algorithm 3 Pseudocode of the part disassembly algorithm.

```
DisassemblePart(P)
 1  //Get the part group of the part to be disassembled
 2  PGroup ← Get-PartGroup(P)
 3  //Check if PGroup only contains one part
 4  if Num-Parts(PGroup) = 1
 5     then return
 6  //Get the ConnectionGraph of part group PGroup
 7  ConGraph ← Get-ConGraph(PGroup)
 8  //Remove vertices corresponding to part P from ConnectionGraph ConGraph
 9  Remove-Node(ConGraph,P)
10  //Remove part P from the part group PGroup
11  Remove-Part(PGroup,P)
12  //Create a new part group containing part P
13  Create-PartGroup(P)
14  //Calculate the connected components of the ConnectionGraph ConGraph
15  ConComps ← Get-Connected-Components(ConGraph)
16  //Loop over all connected components of ConGraph
17  while ConComps ≠ nil
18        do
19           comp ← Get-Component(ConComps)
20           //Check if the part group corresponding to the component comp
21           //is not equal to part group PGroup
22           if comp ≠ PGroup
23              then
24                   //Create a new part group containing all parts of the component comp
25                   Create-PartGroup(comp)
26
```

figure 4.6, the base plate is the only connection between the two bricks. As soon as the base plate is disassembled and removed from the part group, the two bricks do not belong to the same part group, because they are not interconnected anymore.

The interconnection property of a part group can be verified using the *ConnectionGraph* (see section 4.6). Since all handle connections are represented as edges, all vertices in the *ConnectionGraph* must be reachable from each other, in order to fulfill the interconnection property. The pseudocode of the part disassembly algorithm, which is capable of maintaining the interconnection property of a part group at all times, is given in Algorithm 3.

The first step of the algorithm is the selection of the part P which should be disassembled. Each part knows its part group *PGroup*. If the part group does not contain other parts than P, nothing has to be done, since the part is not connected to any other part.

In case the part group does contain other parts, the part vertex and all handle vertices corresponding to part P are removed from the *ConnectionGraph* and the geometric object

of P is also removed from the part group *PGroup*. A new part group only containing part P is created. The removal of the vertices from the *ConnectionGraph* can result in a violation of the interconnection property of the graph and therefore a breakup of the *ConnectionGraph* can occur.

In order to detect a possible violation of the interconnection property and to restore it, the connected components of the *ConnectionGraph* are calculated. If the *ConnectionGraph* consists of more than one connected component, the part group must be split, whereas each component emerges in a new part group.

4.8 Model Import and Export

The VEAM system supports the import and export of assembled models. This is not only important for the integration of the system into a manufacturing process chain, but is also a valuable functionality for creating extensive models.

In general, the connection information of the assembled model and the specific assembly locations of all individual parts are the minimal information needed for the model import and export. No geometric information must be stored or created, since the VEAM system uses predefined parts, whose geometric representations are not changed during the modeling process.

The basis for the model import and export is the *ConnectionGraph*, since it encodes the connection information of all parts of a model. The part assembly location, which specifies the position and orientation of a part, is defined by the user during the modeling process.

The VEAM system does not provide a universal model export module, because different manufacturing process chains require the modeling data in different file formats. Instead the system offers an interface providing functionality to access all needed information to convert the assembly data into the specific required file format. Currently, an export module for the file format used by the INT-MANUS [FSM+06] manufacturing process chain exists.

In order to use the VEAM system for the creation of extensive models, the system provides functionality to save and load assembled models. For this reason, the *ConnectionGraph* is completely serialized and for each part vertex an additional 4x4 matrix representing the assembly location of the part is stored. This information in combination with the product configuration grammar, which contains information of the geometric representation of a part, is sufficient to fully reconstruct an assembled model.

5 VEAM Interaction Techniques

The three-dimensional user interface of the VEAM system and any other Virtual Environment application needs well designed control-display mappings for the conversion of input actions. Control-display mappings refer to the conversion of information provided by the physical input devices into system specific actions.

In general, input actions are human activities to enter a command to a computer system, like pressing a key or moving a mouse. On two-dimensional desktop based systems, the possibility to design control-display mappings are rather limited, due to the standard input and output devices and the widely used Windows, Icons, Menus and Pointers (WIMP) interaction metaphor. This combination provides a well-established and de facto standard mapping of hand actions to functionality. Thus, the way a user applies her hand is predetermined.

Due to the variety of input and output devices of a Virtual Environment, a much larger set of possible control-display mappings exists than for desktop based environment. This is important for the design of direct interaction techniques with virtual objects, which is one of the most common operations in an immersive modeling system. Furthermore, efficient control-display mappings for system control techniques, like three-dimensional collocated widgets, can be designed.

This chapter is divided into three main sections. In the first section, important criteria for the selection of physical input devices are introduced, followed by a description of the input devices actually used.

The second section describes the selection and manipulation techniques used by the VEAM system. Amongst other techniques, this includes the two most often used interaction techniques, namely the part assembly and disassembly operations.

To provide flexible and efficient system control techniques, an application independent three-dimensional widget toolkit has been developed. This toolkit contains a variety of adapted two-dimensional widgets as well as a set of collocated three-dimensional widgets. The toolkit along with its requirements and the integration into the VEAM system is described in the third section.

5.1 Input Devices

For the selection of appropriate physical input devices for a particular three-dimensional application, many factors have to be considered.

Amongst others, the most important criteria are:

1. Device ergonomics: The ergonomic aspect is a major factor for choosing an input device, especially for long-lasting applications. Input devices should put as little strain as possible on the body of a user. If this is not the case, a decrease of efficiency can occur, because it is difficult for a user to perform certain tasks. In general, input devices should be lightweight and easy to use and provide a significant information transfer to the application.

2. Type of input modes: Each three-dimensional application needs input devices with different types of input modes. Common three-dimensional desktop based modeling systems are using a keyboard, a mouse or a graphical tablet as input devices. However, these devices are not suited for an immersive modeling system, because they are cumbersome to use in a Virtual Environment and do not provide all of the needed input types, such as the six degrees of freedom tracking of a user's head or hand. In contrast, a three-dimensional desktop ego shooter game does not need the ability of a sophisticated six DOF user tracking, because their interaction possibilities are tailored for use with a keyboard and a mouse.

3. Control-display mapping strategies: In general, an input device can be used to carry out a variety of interaction techniques by using different control-display mappings. However, these mappings are a crucial factor for the usability of an input device with a specific interaction technique. In most cases, an input device provides natural mappings only for a limited number of interaction techniques in an application and poor mappings for all other techniques. Thus, it is important to classify the interaction tasks of an application and check the needed control-display mappings of all input devices with respect to these interaction techniques.

In most cases the selection of physical input devices for a specific application is a tradeoff. Some input devices are developed for a general purpose and can be used to implement a variety of different interaction techniques. However, they will not provide the best control-display mapping for all of them. Specialized input devices provide a higher usability for specific tasks than a general purpose input device, but are rather limited to these tasks. A comprehensive discussion about the selection of appropriate input devices can be found in [BKJP05].

The physical input devices used by the VEAM system are different stylus-type input devices. Each of the devices is equipped with passive retroreflective markers that are used for the six DOF tracking of the device. They all provide a number of buttons that can be

(a) *Wiimote* (b) Wireless presenter

Figure 5.1: Two of the used stylus-type input devices with attached retroreflective markers.

used for different interaction techniques. In figure 5.1 two of these stylus-type devices are shown.

The *Wiimote* device (figure 5.1a) is the input device of the *Wii*[1] game console and provides several buttons, a speaker and a vibration unit. This device has an ergonomic design and uses a bluetooth connection for the data transfer.

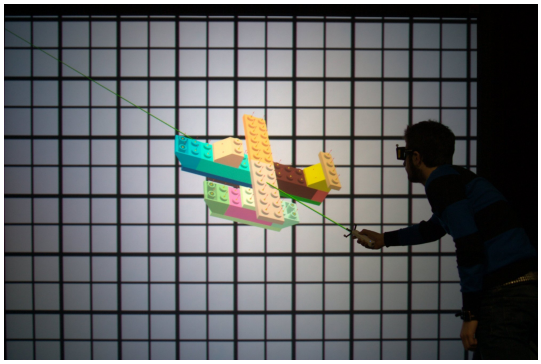
The other stylus-type devices used, such as the device shown in figure 5.1b, are so-called wireless presenters and are commonly used as remote controls for presentations. Compared to the *Wiimote*, these devices are less powerful, because they do not provide a speaker nor a vibration unit and they commonly have fewer buttons. In addition, interference problems occur when multiple devices of the same kind are used simultaneously, because most devices do not use a standard data transfer protocol, like the *Wiimote* does.

5.2 Selection and Manipulation

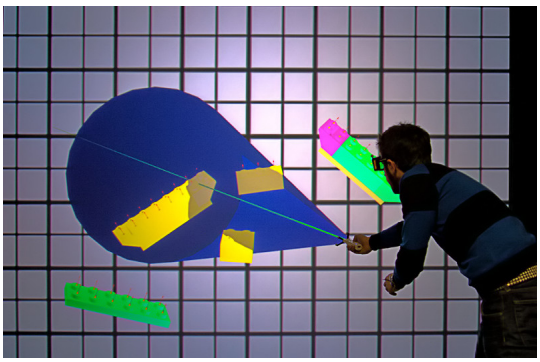
The selection and manipulation of virtual objects in a Virtual Environment is one of the most fundamental tasks and is often a precondition for other techniques like system control.

First the different object selection techniques of the VEAM system are described, followed by a characterization of the used object translation and rotation techniques. Finally, the assembly and disassembly tools, that are making use of the introduced selection and transformation techniques, are described in detail.

[1]Further information about the *Wii* console can be found under www.wii.com



(a) The airplane is selected using the pick ray selection technique.



(b) All part groups that are intersected by the conical volume are selected. To provide a visual feedback, all selected part groups are drawn yellow.

Figure 5.2: Two photos showing the selection tools of the VEAM system. In (a) a single part group is selected by the pick ray technique. In (b) multiple part groups are selected using the conical group selection technique.

5.2.1 Selection Tools

Selection tools are used to select virtual objects in a scene. The system provides a tool to select single objects and a tool for the selection of object groups.

Pick Ray Selection

When using the pick ray selection tool, a virtual ray is attached to the input device and this ray is used for the selection of objects. To reduce the cognitive load of this selection technique, the visualization of the virtual ray is emitted directly from the input devices.

In case a user intersects the virtual ray with an object in the scene, the user has to confirm this selection by pressing a button of the input device. If more than one part is intersected by the ray, only the closest object will be selected. This is a powerful selection mechanism, because it is a natural interaction technique and requires no or little training. One disadvantage of this technique is the difficulty of the selection of distant or small objects, because a high angular precision is needed for pointing at those. However, for the selection of parts in the VEAM system this is no limitation, due to the scaling possibility of the workspace and therefore the scaling of all parts contained by this workspace.

This pick ray selection technique is a direct implementation of the general ray casting selection technique described in section 3.1.2. Figure 5.2a shows a selection of a part using the pick ray selection technique.

Conical Group Selection

In contrast to the pick ray selection, the so-called conical group selection can be used to select multiple parts at once.

In general, the virtual pick ray is replaced with a conical selection volume with its apex at the input device. All objects lying inside of this conical volume are selected.

The application of the conical group selection consists of three consecutive steps. First a user presses a button to indicate a conical selection. The second step is the modification of the selection volume. This is done by modifying the apex angle of the conic selection volume. At the beginning the initial pointing direction vector of the input device is stored. The user can now rotate the input device around an arbitrary axis and the angular difference between the actual pointing direction and the initial direction of the input device is used to alter the apex angle of the conic selection volume. After this adjustment of the size of the selection volume, the user confirms the selection by releasing the button.

This group selection technique can be seen as a modification of the flashlight technique introduced in section 3.1.2. An example of the selection of multiple parts using the conical group selection technique is shown in figure 5.2b.

5.2.2 Transform Tools

Transform tools are used to translate and rotate selected parts. The system supports different dragging and rotation tools. All transformation tools can be used either with one or with multiple selected objects. For the sake of clarity, all techniques are described for a single selected object.

Direct Dragging

The direct drag technique is an isomorphic manipulation technique (please compare the paragraph about isomorphism in section 3.1.1 for a description of isomorphic and nonisomorphic interaction). The selected object is attached at the intersection point of the pick ray with the object. The pivot point (rotation point) is the location of the input device in the virtual space. This is useful, in case the dragged part should be rotated around a user within a constant distance. However, the translation of objects and in particular of far away objects with this technique is inefficient, due to the isomorphic mapping of the hand movement to the movement of the virtual object. Thus a translation can result in a series of object re-selections with subsequent translations.

The virtual pick ray is shown at all times during the manipulation, since it always points directly at the object.

Scaled Dragging

In contrast to the direct dragging technique, the scaled dragging tool uses a nonisomorphic mapping of the movement of the input device, with the aim of providing a technique for manipulating all objects in a scene, independent from their distance to a user.

The nonisomorphic mapping uses a linear scale factor, opposed to the non linear mapping function of the Go-Go technique described in section 3.1.2. The linear scale factor is recalculated for every manipulation. This is done by calculating the distance, d_o, between the selected object and a user (in most cases the head of a user) as well as the distance, d_h, between the hand (position of the input device) and a user. The scaling factor is now given by $s = \frac{d_0}{d_h}$.

Using the scaled dragging technique, all objects in a virtual scene, independent from their distance to a user, can be efficiently translated in the scene.

The pick ray is always pointing at the selected object, because only the translation component is affected by the linear scale factor. Therefore, the visual representation of the pick ray is shown during the dragging.

In-place Rotation

The described dragging techniques are suited for the translation and for the rotation of objects around the axis defined by the pick ray. However, rotations around other axes are cumbersome to achieve with these techniques, in particular for far away objects. Since the pivot point of the direct grab and also of the scaled grab technique is set to the center of the input device; distant objects must first be brought in close-up range of the user. Then the rotation can be applied, followed by the translation of the object back to its original position.

In order to overcome this problem and to provide an intuitive technique for object rotations, the pivot point is not set to the location of the input device, but rather to the center of the selected object. Thus, all rotations of an object are in the local reference frame of the object. As a result, the world position of the center of the object is not affected by these rotations.

Additionally, the mapping of the rotation of the input device can be done in an isomorphic and in a nonisomorphic way. In some cases it is useful to amplify the rotations of the input device, so that a user can control a larger range of rotations with a small rotation of the input device. However, for some users nonisomorphic rotational mappings are not intuitive and therefore the developed in-place rotation tool supports both isomorphic and nonisomorphic mappings.

The visualisation of the pick ray is disabled during the complete execution of the in-place rotation technique, because it is not necessarily pointing to the rotated object any more.

5.2.3 Assembly Tools

The most often used interaction techniques in the VEAM system are the part assembly and the part disassembly technique. They are used during the complete modeling process and make use of the previously described selection and transformation techniques.

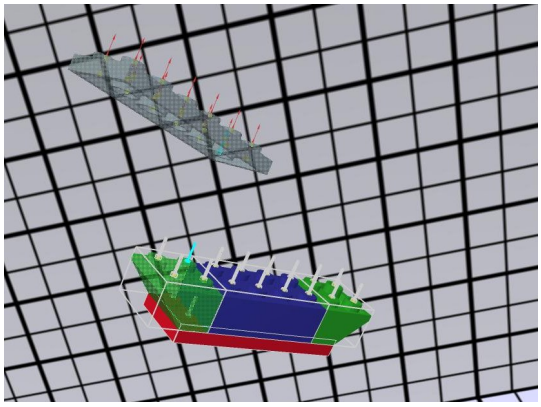
Part Assembly

This tool is used for the part assembly operation according to the part assembly algorithm described in section 4.7.1.

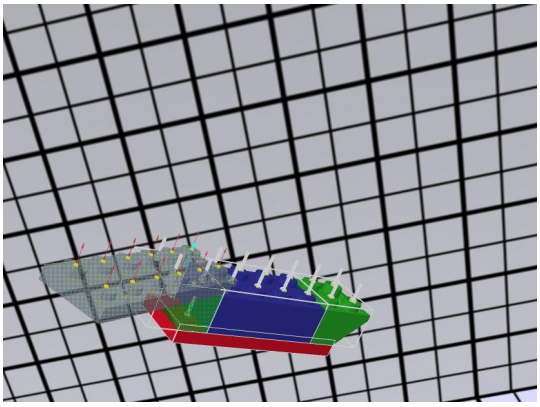
For the part assembly operation, first of all a part must be selected. This can be either done with the pick ray or the conic group selection technique. After the selection of a part, all transformation techniques can be applied to this part. In most cases, this is a transformation of the part in order to bring it close to its final assembly location. During these translation and rotation operations, the system checks if the part comes close to any other part in the workspace. In this case, the closest compatible handle pair of these two parts is calculated. If a compatible handle pair is found, both parts are drawn in a transparent mode and the two handles are highlighted. The marking of the compatible handle pair provides a visual cue about possible assembly configurations. Two highlighted *AxisMatch* handles are shown in figure 5.3a.

In case a user brings two parts close together and the distance between the two compatible handles is smaller than a specific threshold δ, the system assumes a user is indicating an implicit connection request. Thus, the two parts are snapped together and a preliminary connection between these two parts is established. Figure 5.3b shows the snapping of two parts and the establishment of a preliminary connection. This interim connection can already form a valid assembly configuration, but does not have to, because depending on the involved handle types, the connection can be altered according to a constrained transformation. These transformation constraints are defined by the involved handle types and are described in section 4.4.

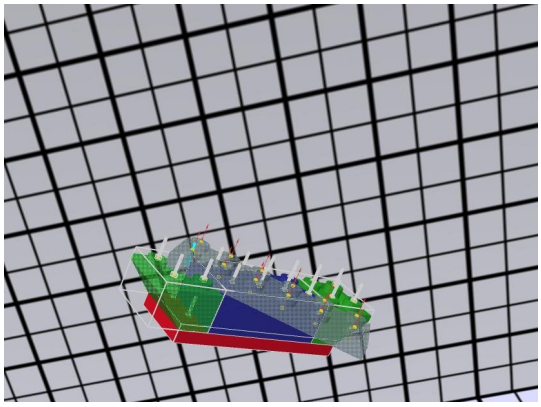
In case the preliminary connection was established by two *AxisMatch* handles, it can be altered by a rotation around the common axis defined by the vectors of the two *AxisMatch* handles. During this constrained transformation, the rotation of the input device around the common axis of the two handles is directly mapped to the rotation of the part which should be assembled. The constrained transformation of two *AxisMatch* handles is shown in figure 5.3c.



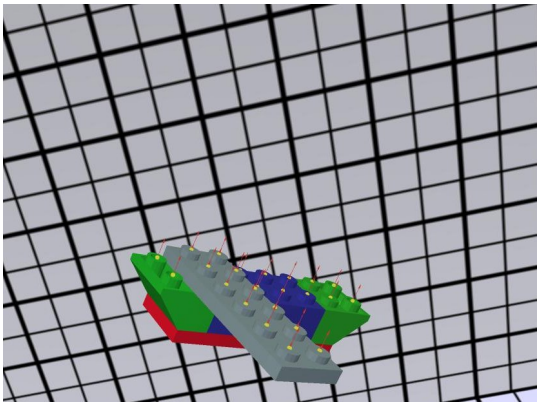
(a) Highlighting of the closest compatible handle pair. The involved parts are drawn in a transparent mode.



(b) The involved parts are snapped and a preliminary connection is established.



(c) Execution of a constricted transformation of the preliminary connection according to the involved handle types.



(d) The final assembled model.

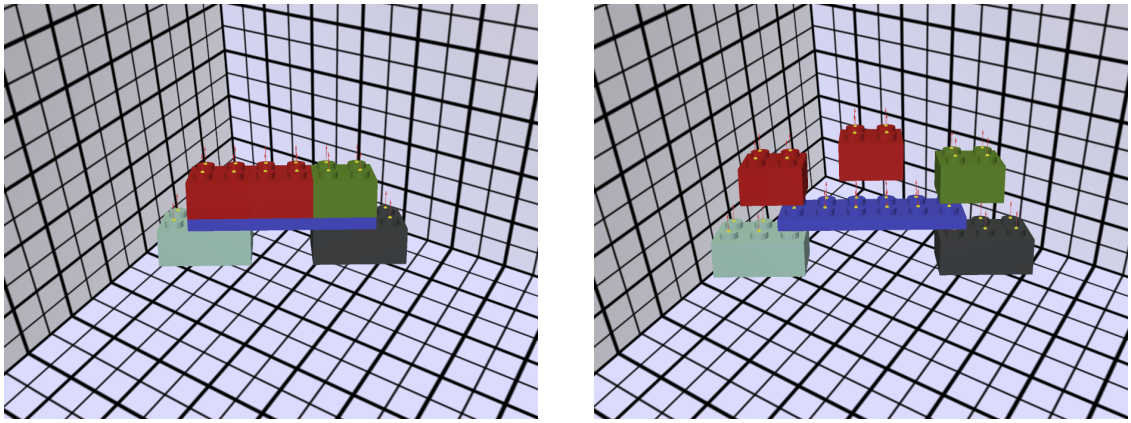
Figure 5.3: Image sequence illustrating the different steps of the part assembly operation in the VEAM system.

After a user positioned the part using the restricted transformation, the validity of this position needs to be checked. This is done by the evaluation of the product configuration grammar with all involved handle pairs. In case the position of the part is not valid, the system searches the near neighborhood for valid positions. If a valid position is found, the part will be pulled tight to this position. This automatic correction of the final assembly location is shown in figure 5.3d.

The system highlights the assembled part, using a linear color blending, as a visual feedback mechanism to indicate a successful part connection.

Part Disassembly

This tool is used to disassemble a part from a model on the basis of the algorithm described in section 4.7.2.



(a) The assembled model.

(b) The blue bar was selected to generate the explosion drawing.

Figure 5.4: Illustration of the explosion drawing of a simple model.

To perform a disassembly operation, a single part of a model must be selected. The selected part is then unhinged from the model and a possible breakup of the model is calculated. Due to the possible breakup of the model, all unhinged parts, this includes the initial selected part as well as the parts involved in the breakup of the model, are highlighted in order to provide a visual feedback to the user.

All disassembled parts are immediately treated as new models by the system and therefore all selection, transformation and assembly tools can be applied to them.

Explosion Drawing

The identification of directly connected parts especially in large models can be difficult. The *ConnectionGraph* contains this information, but its direct inspection is not intuitive. Thus, a graphical illustration similar to an explosion drawing is used for visual analysis of part connections.

A user selects a single part of a model and all directly connected parts are translated recursively to generate the explosion like model. Figure 5.4 shows a simple model and the explosion drawing generated in case the blue bar is selected.

5.2.4 Two-User Tools

The system does not only provide single user tools, but also tools for a collaborative interaction of two users. In general, the two-user tools also make use of the selection and manipulation techniques previously described, but apply additional locking constraints. E.g. all transformation tools, like the direct and scaled grab tool are locking the selected parts in order to prevent concurrent transformations.

More information and scenarios about the two-user functionality of the VEAM system can be found in [FWd+08].

Collaborative part assembly

Using the collaborative part assembly tool, two users can work on a single part assembly operation cooperatively. The usage of this tool is straight forward. Both users can simultaneously select and grab two different parts, move them through the scene and assemble them in the desired way.

Part hand-over

In a collaborative modeling environment it can be useful to provide a direct hand-over functionality of parts from one user to another. This is advantageous in case the part creation widget is not easily accessible. One user can ask the other to create the desired part and hand it over. In addition, this technique can be useful in a scenario where various modeling possibilities are discussed and one user needs the control over a part which is currently locked by another user.

The hand-over process is initiated by positioning the part towards the hand of the other user. In addition to this, a button of the input device needs to be pressed to issue the transfer of the control of the part. Opposed to the standard part selection techniques, like the pick ray or the conical group selection technique, the user who wants the control over a part does not have to perform any ray intersections. Thus, the part hand-over method is an easy to use collaborative selection technique.

5.3 System Control

A system control task is a user task to send commands to an application in order to change the interaction mode or the state of a system. Every non-trivial application must provide system control interfaces for these tasks. Almost all desktop based application are using the WIMP interaction metaphor for system control tasks. Due to the variety of different input and output devices of Virtual Environments, no standard interaction metaphor for system control techniques in Virtual Environment applications exists.

The VEAM system uses graphical menus and three-dimensional collocated widgets for system control tasks. In the following sections, the requirements of the developed widget toolkit with respect to the VEAM system are introduced in detail, followed by a description of the used three-dimensional widgets.

5.3.1 Graphical Menus

This section describes and analyses the requirements and functionality of the developed graphical menu system and describes its integration in the VEAM system.

Menu embedding

Some applications are using special devices, like a PDA or a Tablet-PC, for the display and the interaction with a graphical menu. The development of graphical user interfaces for such systems is convenient, because many mature widget toolkits exist for these devices. However, a major drawback of the use of such devices for system control tasks in a Virtual Environment is the loss of immersion caused by the usage of these devices. This immersion loss results from the necessity of refocusing before and after the interaction with the device.

To prevent this distraction of the immersion, the developed menu system is completely integrated in the Virtual Environment and does not need any additional devices besides the existing stylus-type input devices.

Dimensionality of the menus

A fundamental property of a graphical menu system is its dimensionality. An example for a one-dimensional menu is a circular menu, which is commonly used for selection tasks. Two-dimensional menus are commonly adaptions of WIMP based desktop menus, like pop-up menus or floating menus. These menus are powerful, because they do not only provide functionality for the selection of items, but can also be used to manipulate parameters and handle symbolic input. The concepts of both one- and two-dimensional menus are briefly described in section 3.2.2.

In [GB04] Gerber et al. compare the selection efficiency of a one-dimensional circular selection menu in contrast to an adapted two-dimensional menu. They conducted a user study, in which users had to select a specific item out of an unordered collection of items. The ring menu contained ten items and the adapted two-dimensional menu was holding twenty five items. Their user tests have shown that both menus provide the same selection efficiency, whereas the two-dimensional menu contained more than twice the amount of items.

Another advantage of the usage of adapted two-dimensional menus in a Virtual Environment applications is their familiarity to users. This is particular important for novice users, because they do not need any or little training when this kind of menus is used in an immersive application (see the subsection about *Graphical Menus* in section 3.2.2 for further details).

Due to the high selection efficiency, the capability of handling symbolic input and their familiarity, the VEAM system partially uses adopted two-dimensional menus for system control tasks.

Widget Layout

The layout mechanism of a widget toolkit is important, because it affects the structure, the comprehensibility and the overall performance of a menu.

Many factors must be considered when determining the efficiency of a widget layout. Amongst others, the major factors are the experience of the user and the familiarity with the input device, the size of the widgets and their distance. In 1954 Fitts [Fit54] discovered a formal relationship that predicts the time needed for a rapid one-dimensional movement to a target area. Fitts' law is a function of the distance and the size of the target. The pristine formulation of Fitts' law was only valid for one-dimensional tasks. However, MacKenzie and Buxton [MB92] extended Fitts' law to general two-dimensional target acquisition tasks. This allows the application of the Fitts' law in two-dimensional graphical menus. Fitts' law can be written as:

$$T = a + b \log_2 \left(\frac{D}{W} + 1 \right), \tag{5.1}$$

whereas T denotes the time needed for the completion of the movement. The start and stop time of the used input device is labeled as a and the speed of the device is represented by b. Both a and b are device dependent and can be estimated by a linear regression of measured data. D is the distance from the starting point to the center of the target area. The allowed error tolerance of the final position along the axis of motion is called W. The equation models a tradeoff between speed and accuracy in rapid movements.

The following rules can be deduced from Fitts' law:

- Widgets that are frequently used should have a large target area.
- Widgets or actions that are executed often, should be arranged close to the average cursor position.
- The top, bottom, left and right sides of the screen are highly targetable, because they provide an infinite boundary. This results from the unnatural boundary created by the edges of the screen. All targets lying on these edges are infinitely targetable, because they are impossible to go past. However, in case of a virtual screen, like the one used in the VEAM system, the sides of the screen do not provide any boundary, since the pointer can easily be moved beyond them.

Algorithm 4 Pseudocode of the widget layout algorithm of a window.

```
DO-LAYOUT(W)
 1  //Get a list of all children widgets of W
 2  children ← GET-CHILDREN-WIDGETS(W)
 3  accumSize ← 0
 4  //Loop over all widgets of the list children
 5  for each child ∈ children
 6     then
 7          //Get the size of widget child
 8          size ← GET-WIDGET-SIZE(child)
 9          //Accumulate the size of all children widgets
10          accumSize + =size
11  //Calculate the sizes that should be assigned to the children widgets
12  DETERMINE-WIDGET-SIZES()
13  //Calculate the layout of the widget W
14  CALCULATE-WIDGET-LAYOUT(W)
15  //Loop over all widgets of the list children
16  for each child ∈ children
17  //Call DO-LAYOUT() recursively for the child widget
18     then DO-LAYOUT(child)
```

On the basis of these assumptions, the menu system uses a recursive block layout system for all widgets. Two container widgets, corresponding to a vertical respectively horizontal container exists. Each widget, including the container widgets, can be put recursively into other widget.

The layout algorithm uses a top down approach, where the top level widget is responsible for the layout of all children widgets. The algorithm is based on two recursive passes. In the first pass each widget is queried for its desired size. In case the queried widget is a container widget, it will recursively query and accumulate the sizes of the children widgets.

After the completion of this step, the widgets calculate the sizes that should be assigned to itself and to each of its children widgets. This computation depends on the widget type and on the total widget size available. The second recursive pass sets the boundaries of all widgets according to the previously determined size. The pseudocode of the algorithm is given in Algorithm 4. This recursive layout mechanism is a simple, yet flexible way to create arbitrary box layouts.

Context menu

Context menus are particularly suited for menus containing a limited number of elements and do not clutter the environment like excessive global menus. The content of a context menu refers to the actual user context, which is defined by the interaction type and the selected object. A traditional context menu is displayed as a pop-up menu and is only

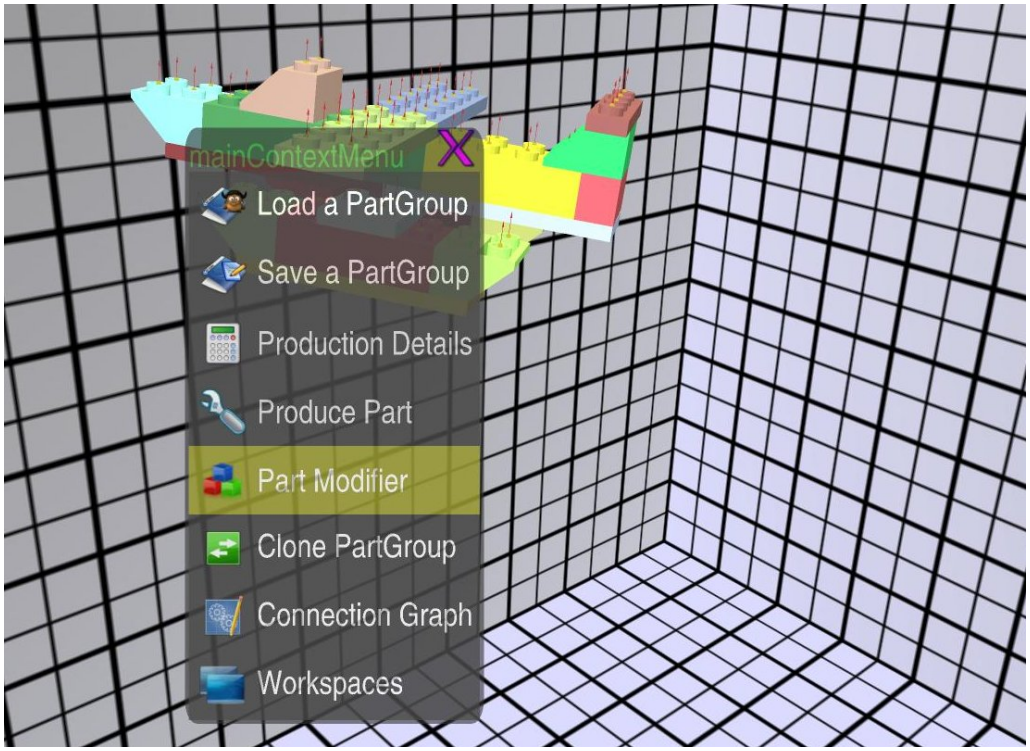


Figure 5.5: A semitransparent context menu of the VEAM system for the selection of a part group.

used to trigger actions and does not provide any features for parameter manipulation.

The menu used by the VEAM system is a combination of a context menu and a floating menu. Not only does it have the full functionality of a context menu, by providing context dependent features, but it also offers techniques for parameter manipulation and the handling of symbolic input.

Another requirement of the menu is the possibility of a hierarchical organization of the menu entries. The menu always displays a lower hierarchy level at the side of the menu corresponding to the parent hierarchy level. In case the menu representing a specific level of the hierarchy is closed, the system automatically closes all visible lower level menus. This kind of hierarchical structure allows an efficient navigation through the different levels of the menu hierarchy. The downside of this kind of hierarchical organization is a possible increase of occlusion caused by visible menus of a lower level in the hierarchy. Hence the depth of the hierarchy should not exceed a certain limit. In most cases a maximal depth of three is sufficient. In general, this kind of hierarchical organization is always a tradeoff between an efficient navigation through the levels of the hierarchy and a potential occlusion caused by them.

The menu has a window-frame and can be used as a normal floating menu. E.g. the menu can be moved through the environment and can also be utilized as a global menu. Figure 5.5 illustrates the menu within the context of the part selection.

Interaction style

As described in section 5.1 a user interacts with the modeling environment with stylus-type input devices and uses a ray casting interaction technique for indicating object selections.

Each of the input devices used must have at least two different buttons. The first button is used for the selection and manipulation of the virtual object with the actual activated tool and the second button invokes the context menu defined by the active tool and the selected object. The menu itself is a virtual object in the environment, therefore the first button can be used to rotate and translate the menu, by selecting its window-frame. All interaction tasks with the elements of the menu are done with the first button. The menu is automatically closed in case the second button is pressed again or the context menu is invoked for another context.

This button handling allows the simultaneous usage of the menu and the application of different manipulation techniques, like the part assembly operation.

Occlusion problems

The placement of new virtual objects in an existing three-dimensional scene can cause different occlusion effects for the participating users. On the one hand, the newly added object can be occluded or intersected by existing objects in the scene, or on the other hand, the new object may hide a part of the scene. In both cases either the visibility of the newly added object or of the existing objects is affected.

A natural requirement of every graphical menu is its visibility during the interaction, but at the same time a menu in a Virtual Environment should not cause ”too” much occlusion in the environment. Hence no ultimate position of a graphical menu, in particular of a floating menu, exists.

This system uses two techniques, namely transparency and a rule based placement mechanism to calculate an acceptable position of the menu.

Transparency The VEAM system uses a context based menu, which is tightly coupled to the combination of the active tool and the selected object. Thus, it is reasonable to position the menu in close range to the selected object. This kind of positioning allows a user to keep focused on the location of the interaction. However, it is most likely that the position of the menu totally or at least partially occludes the selected object. For context menus that are only used to trigger one command and as a result the menu will disappear, this temporarily occlusion causes no severe problems.

The used context based menu provides more sophisticated functionality than the simple dispatch of commands. Therefore, the menu is often used to trigger several successive

commands and it is not hidden after one single interaction. To reduce this occlusion problem, the menu is drawn in a semitransparent mode, which allows a sufficient view of objects behind the menu for most operations.

Placement An important characteristic of a graphical menu system in a Virtual Environment is its placement mechanism. In section 3.2.2 several menu placement heuristics, namely World-referenced, Object-referenced, Body-referenced and Device-referenced placement methods are described. Each of these four techniques are static placement techniques, because they only consider fixed relative positions respectively orientations and do not take into account any other objects in the Virtual Environment.

Potential problems of static menu placement techniques are occlusion or intersections of the menu by other objects in the scene or the reduced usability of the menu, because the distance from a user to the menu is too large. This problem often occurs in case an object-referenced menu placement heuristic is used and the objects are far away from a user.

In order to avoid these problems or at least to minimize them, the menu system uses a rule based positioning mechanism. This rule based placement policy tries to ensure the visibility and the usability of the graphical menu for all initial positions.

The first step of the placement algorithm is the calculation of a reference point, R. This point is the intersection of the virtual pick ray of the user with the scene. All users are physically located in an invisible cube, therefore every pick ray is either intersecting the virtual objects in the scene or the hidden background cube. After this, a reference line from the head position, H, of the user to the reference point is computed. This line is utilised as a hanging reference line for positioning the menu, since it provides an intuitive optical alignment for the user. The next step is the calculation of the exact position of the menu along the reference line. This is done in various steps. First the menu is positioned at a predefined distance, l_{max}, along the reference line. This predefined maximal distance ensures that the menu is displayed at a suitable distance which allows a comfortable use of the menu. However, in this position, the menu can be occluded or intersected by objects in the scene. This potential occlusion or intersection is calculated by emitting rays from the menu surface, in a specific pattern, in the direction of the head of the user. In case any of these rays intersect an object in the scene, the menu is occluded or intersected. This interference is resolved by translating the menu along the reference line in the direction pointing from R to H until the menu is not occluded any more or a predefined minimal distance, l_{min}, of the menu to the user is undercut. The minimal distance is used to ensure the usability of the menu, even though there are objects located directly in front of the user. Normally, these objects are very small and do not distract the view of the user too much; otherwise the user would not have invoked the menu from that position. In case such a translation of the menu was applied, its scaling is adapted, because the distance to

Algorithm 5 Pseudocode of the menu placement algorithm.

```
MENU-PLACEMENT(M)
 1  //Get the pointing direction of the toolholder
 2  Dir ← GET-TOOLHOLDER-DIR()
 3  //Calculate the reference point R
 4  R ← CALC-INTERSECTION(Dir)
 5  //Get the position of the user's head
 6  H ← GET-HEAD-POSITION()
 7  //Place the menu M along the reference line at the distance l_max
 8  PLACE-MENU(M,R,H,l_max)
 9  //Check if the menu M is intersected or occluded by any objects in the scene
10  if CHECK-INTERSECTION(M)
11     then
12          //Calculate the translation vector along the reference line in order to
13          //resolve any intersection or a possible occlusion of the menu M
14          Trans ← CALC-RESOLVE-TRANS()
15          //Apply the transformation Trans
16          APPLY-TRANS(M,Trans)
17          //Check if the distance from menu M to the head position H of the user
18          //is smaller than l_min
19          if GET-MENU-DIST(M) < l_min
20          //Place the menu M along the reference line at the distance l_min
21             then PLACE-MENU(M,R,H,l_min)
22          //Adjust the scaling of menu M
23          ADJUST-SCALE-FACTOR(M)
24  //Get the orientation of the user's head
25  Rot ← GET-HEAD-ROT()
26  //Rotate the menu M so that the user is looking vertically at the menu
27  APPLY-ROT(M,Rot)
```

the user is smaller than the default distance. In the last step, the orientation of the menu is adjusted in a way, that the user is looking vertically at the menu. The pseudocode of the menu placement policy is given in Algorithm 5.

Widget Types

For the graphical menu used in VEAM system, a variety of different widgets is needed. Despite widgets for triggering commands, widgets for parameter manipulation and widgets for handling symbolic input are required.

All widgets provide a theme interface which can be used to adapt their look and feel. This is useful for users, because they can partially adjust the visualization of the menus to their needs. Another benefit of the usage of widget themes, is the uniform appearance of all widgets.

To ensure a reliable and error free selection and activation of a menu entry, each widget

provides a user feedback mechanism. A general highlighting mechanism of all widgets is the highlighting of the background when a user selects the widget. Depending on the specific widget type, each widget provides additional visual feedback. E.g. the slider uses an opaque color for indicating the actual value.

The following paragraphs describe some of the most important and frequently used widgets implemented in the widget toolkit.

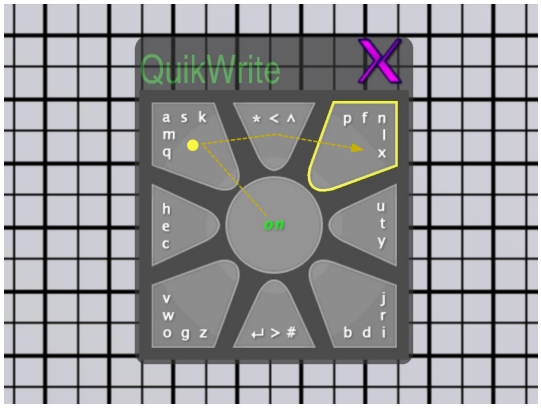
PushButton This widget provides functionality for issuing commands. It is one of the most frequently used widgets in any user interface. Its visual representation can consist of an icon, a text label or both.

Checkbox The checkbox widget is an option button which can be either checked or unchecked. It is typically used to implement features that can be enabled or disabled without affecting other checkboxes. The visual representation of a checkbox is a meaningful icon along with a text label. Both the icon and the text label have two states representing the actual status of the checkbox.

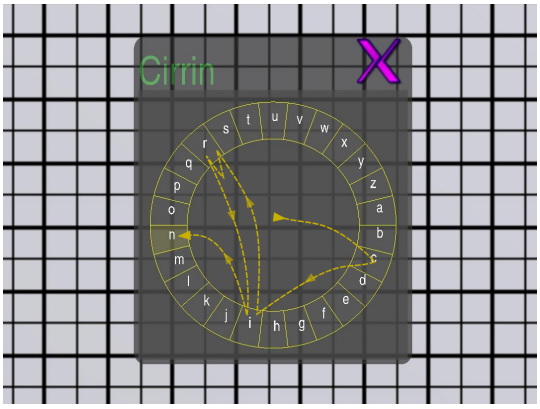
RadioButton The radio button is also an option button than can be switched on or off. It is used to allow a user to choose one option out of a set of options. In each group of radio buttons only one radio button can be activated at any time. A radio button is visually represented by an icon and a text label. Again, the icon and the text label can have two states indicating the status of the button.

Slider The slider widget is used for controlling a bounded integer value. A user can drag the slider horizontally or vertically, depending on the orientation of the slider. The position of the slider handle is used to calculate the actual integer value within the valid range. Its valid range is visualized using a default background color and the actual value is indicated by an opaque marking.

ScrollArea The scroll area widget provides a scrolling view on other widgets. It is used to display widgets in a frame. In case the children widgets are exceeding the size of the scroll area widget, the view displays a scroll bar, which can be used to view the entire area of the child widgets. The scroll area is not distinguishable from a vertical or horizontal container, unless the child widgets are exceeding the size of the scroll bar. If this is the case, a scroll bar is automatically displayed. The scroll bar consists of a background color and a opaque color representing the scrolling handle.



(a) The reference region is marked with a yellow circle. The actual selected character of stroke is a 'k', because the last selected region is two regions right of the reference region and the center character of the reference region is an 'a'. The textures representing the nine regions are taken from [Dre07].



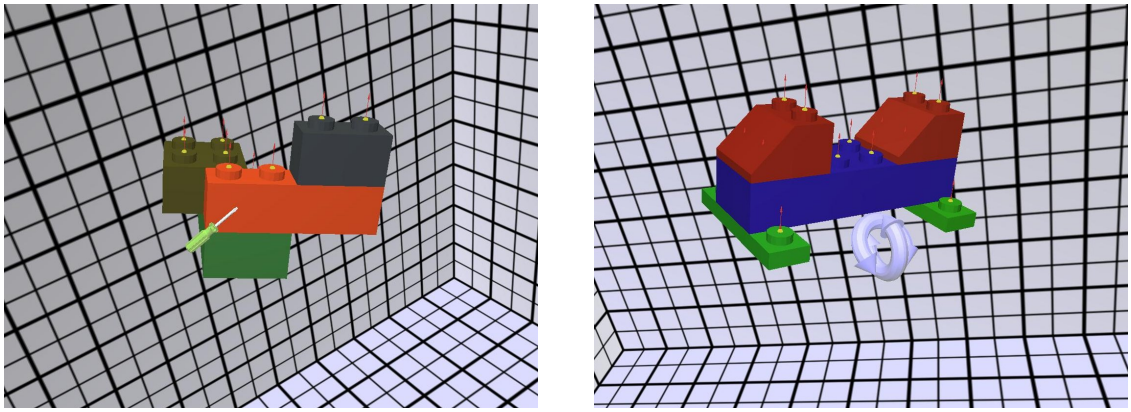
(b) The marked stroke describes to the word "cirrin". Each of the regions correspond to the shown character.

Figure 5.6: Illustration of the *Quikwrite* (a) and the *Cirrin* (b) widget of the developed widget toolkit for handling symbolic input.

FileDialog The file dialog widget provides the possibility to traverse the file system and to select files or directories. The file dialog supports two different modes, one mode to save and one mode to load files. The files and subdirectories of the actual selected directory are shown in a scroll area whereas each entry is represented by a push button. The developed file dialog which used in the VEAM system to save and load models is shown if figure 3.5.

Quikwrite Handling symbolic input is a tedious task in a Virtual Environment application mainly because no keyboard is available. However, many techniques, such as *Cirrin* [MA98] (see figure 5.6b) or *T-Cube* [VN94], have been developed for handling symbolic input in mobile devices (most of them also do not provide a full keyboard). An interesting input technique called *Quikwrite* [Per98] was developed by Perlin[2] in 1998. The basic concept is a 3x3 grid, where characters or symbols are located in the eight outer regions and a home region is in the center. All characters can be drawn by one single stroke: Every line is starting at the center region, going to multiple adjacent regions containing the characters and returning to the center. The first selected outer region can be seen as the reference region of the stroke. All symbols contained by this region can be chosen by selecting the different adjacent regions. The actual selection of a symbol is confirmed by selecting the center region. Multiple strokes can be done in a row, since all strokes are beginning and ending in the center region. In figure 5.6a the implemented *Quikwrite* widget with some explanatory annotations is shown.

[2]More information about Quikwrite along with an interactive applet can be found under: www.mrl.nyu.edu/~perlin



(a) Part disassemble widget

(b) Part group in-place rotation widget

Figure 5.7: Illustration of two 3D widgets used for system control tasks. In (a), the part disassemble widget corresponding to the red part is shown. The 3D widget representing the in-place rotation of a part group is shown in (b).

5.3.2 3D Widgets

Adapted two-dimensional widgets are efficient techniques for system control tasks mostly because every user knows them from desktop based applications. These menus are tailored for the usage within the WIMP metaphor, which is suited for desktop based interaction with common input devices like a keyboard and a mouse. However, in a Virtual Environment a larger set of possible control-display mappings exists than in a non-immersive desktop based environment.

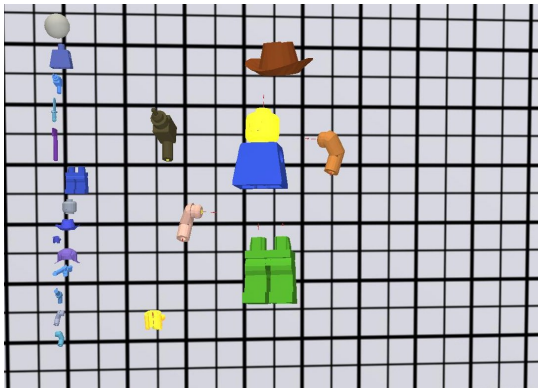
One fundamental advantage of control-display mappings of Virtual Environment applications compared to two-dimensional applications is the availability of the extra degree of freedom. Menus that exploit this potential are called 3D menus and are a combination of geometry and behavior. A description of 3D menus is given in section 3.2.2.

The VEAM system uses 3D widgets for system control tasks, by moving functionality directly onto objects, allowing the exploitation of the full degrees of freedom of the three-dimensional workspace.

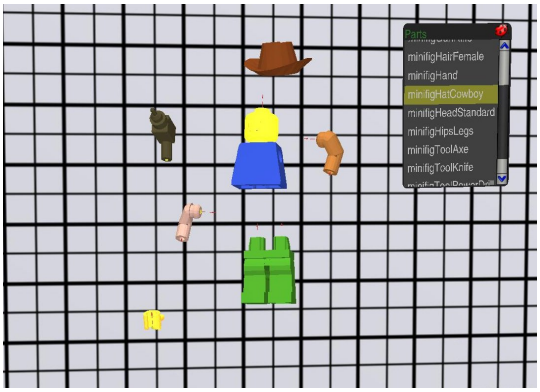
Disassemble Widget

In most cases the selection of an already assembled part implies a starting action of an assembly operation. However, this is not the only possible interpretation of this selection task. A user can just as well intend to unhinge the selected part from its actual part group.

A common workflow of this action would consist of the selection of the disassembly tool from a menu followed by the successive selection of the part which should be disassembled.



(a) A three-dimensional widget (on the left hand side) containing miniature previews of the parts. These miniature models can be directly selected in order to add them to the workspace.



(b) An adapted two-dimensional scroll area widget (on the right hand side) containing textural part descriptions for the selection of additional parts.

Figure 5.8: Illustration of two different widget types for the selection and the adding of new parts to the workspace used in the VEAM system.

Instead of these two distinct selection operations, which in turn can require other operations such as the display and concealment of a tool selection widget, a user can directly select the disassemble widget on top of a part and the part will be immediately unhinged and is automatically attached to the input device of the user for further manipulation. In doing so the selection and execution of a command forms one fluent action sequence. An example of the collocated 3D disassemble widget is shown in figure 5.7a.

Rotate Widget

A common operation on part groups, especially on distant part groups, is the in-place rotation. The VEAM system also provides a three-dimensional widget allowing the selection of the part group to be rotated with a simultaneous activation of the in-place rotation tool. The 3D rotation widget is shown in figure 5.7b.

Part Selection Widget

The disassemble and the in-place rotation widget are object collocated 3D widgets that are working on parts or respectively on part groups. Opposed to this, the 3D part selection widget is not collocated but can rather be moved through the modeling environment. It is used to select and add new parts to the workspace.

All known parts of the system, which are defined by the product configuration grammar, are shown as a miniature model of their real geometric models. Per default, all miniature models are aligned in a vertical row. A part can be selected with a simple pick ray selection, whereas the actually selected part is enlarged and rotated, in order to provide

a visual feedback of the selection. If the part selection is confirmed, this part is added to the workspace and is directly attached to the input device for further manipulation.

Figure 5.8 shows the part selection using the 3D part selection widget and an adapted two-dimensional scroll area containing the textural description of the parts.

6 Software Design

This chapter describes the implementation of the VEAM system introduced in chapter 4 and the implementation of the three-dimensional widget toolkit described in chapter 5.

Implementation specific terms and definitions, like objects, classes and functions are written in `typewriter font`.

6.1 Assembly based Modeling Framework

The following sections describe the implementation of the VEAM system. First the overall design principles of the system are introduced, followed by a more detailed description of the main modules of the system. For the sake of clarity, all used UML diagrams were reduced to the essential details compared to the real implementation.

6.1.1 Design Principles

The most important design principle of the modeling VEAM system is its strict object oriented and modular architecture. Each module has a clearly defined interface, which can be used by other modules to access its functionality. Since the complete functionality of a module is encapsulated and therefore hidden from other modules, a module can be easily extended or completely exchanged by another module which provides the same interface.

Another reason for this high degree of encapsulation of each module, is the requirement of the system to be integrated into a real manufacturing process chain.

The system uses the Virtual Reality framework AVANGO™ which is described in section 2.3 and is written with the *C++* programming language. In addition to its *C++* API all module interfaces provide a binding to the interpreted scripting language *Scheme*.

Figure 6.1 shows a simplified dataflow oriented diagram of the system architecture. The minimal input of the system consists of the product configuration grammar and a profile which describes the environment where the system is running. Currently, the VEAM system can be executed in the immersive *TwoView* display system. Since AVANGO™ is used for the display abstraction, the system can easily be extended to run with additional environments, like a responsive workbench.

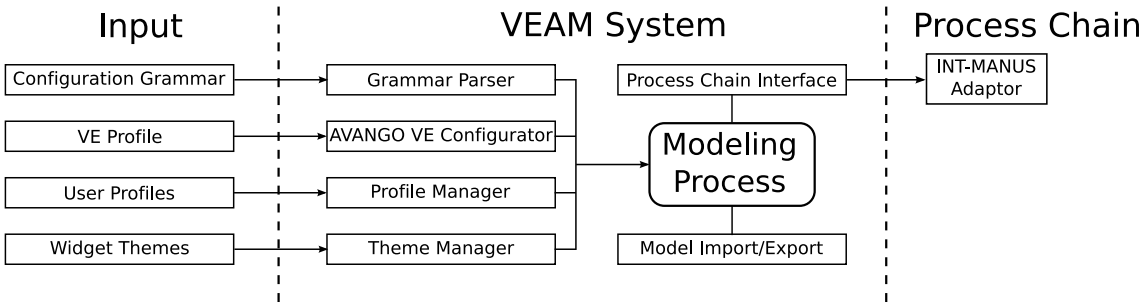


Figure 6.1: Simplified dataflow oriented diagram of the central modules of the VEAM architecture. The minimal input of the VEAM system consists of the product configuration grammar describing the modeling primitives and a profile specifying the used Virtual Environment. The VEAM system provides a model import and export module and an interface to access the functionality needed for the integration of the system into a manufacturing process chain.

In case different user roles are needed, an additional configuration script which defines the specific tools for each user, is needed. If no such script is provided, all registered users are equipped with the same set of tools and therefore have equal modeling possibilities.

The developed widget toolkit which is used to implement the system control functionality of the VEAM system has a built-in theme support. Therefore, a user based theme can be used to personalize the look and feel of the three-dimensional user interface of the VEAM system.

The needed functionality for the integration of the VEAM system into an existing manufacturing process chain, like the INT-MANUS process chain described in [FSM+06], is encapsulated in a single module. Since every manufacturing process chain has its own interfaces, no universal export module can be provided by the VEAM system. Instead environments specific adapter objects are used to convert the modeling data into the format needed by the particular manufacturing process chain.

6.1.2 Product Configuration Grammar Parser

The VEAM system uses the production configuration grammar (see section 4.5) to define all possible part connections. The system uses an interface to encapsulate the functionality to add parts and their product configuration grammar. Due to this abstraction, the system is not restricted to a single way how these information are passed to the system.

A convenient way to specify the product configuration grammar is to use an XML based description of the grammar (see listing 4.1 for an example). These XML files can be edited directly by a user or can be utilized as a data exchange format between the VEAM system and another application generating these XML based grammar files.

To handle these XML product configuration grammar files, the VEAM system provides a

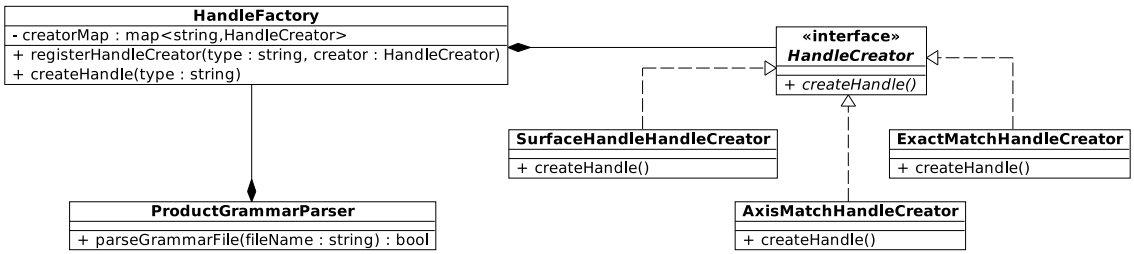


Figure 6.2: UML class diagram illustrating the *Factory Pattern* used for the creation of handles. All handle creator objects are implementing the `HandleCreator` interface and are registered at the `HandleFactory` object with a unique id, which is later used by the parser to identify the specific creator object.

parser module. The input of the parser is an XML grammar file and every file is validated against an XML schema[1], which contains a formal description of the product configuration grammar, because the parser can not rely on the correctness of the given grammar file. In case the validation fails, the grammar will not be parsed and therefore not added to the system. This mechanism ensures a valid state of the internal grammar module of the VEAM system at all times.

The grammar parser is implemented as a state machine and uses the *xerces*[2] library for reading and processing XML data and to generate the *Document Object Model (DOM)* tree. The document object model is a standardized, platform and language independent description how to access and manipulate XML documents. The nodes of a DOM tree represent the text and the attributes of an XML document. In addition, the tree structure provides a convenient way for data processing.

The parser is implemented as a state machine, which switches its states depending on the parsed DOM nodes. A central component of the parser is the mechanism to dynamically create handles based on their identifiers in the XML file.

The VEAM system currently supports four different handle types (see section 4.4 for their description), each defining a different part connection semantic. These handle types do not cover all possible connection semantics, but are sufficient for most uses. In case additional connection semantics are needed, new handle types must be added to the system. Therefore, the parser needs an easy to use and flexible mechanism to deal with new handle types. This is accomplished by using a variation of the *Abstract Factory Pattern* introduced by Gamma et al. [GHJV95]. In general, this design pattern provides an interface to create related objects at runtime without specifying their concrete class type. This is typically done by two class hierarchies. One hierarchy represents the products (in this case the spe-

[1] XML schema is used as a formal description of XML vocabularies and consists of a set of rules which an XML document must fulfill. More information about XML schema and its formal definition can be found under http://www.w3.org/XML/Schema

[2] *Xerces* is an open source XML parser developed by the Apache Software Foundation. Further information are available at: http://xerces.apache.org

cific handle types) to be created and the other hierarchy contains creator objects. Every product is not created directly, but rather by using the corresponding creator object.

The parser uses this pattern to dynamically create handles out of the parsed file. Figure 6.2 shows a UML class diagram illustrating the application of the pattern in this context. All specific creator objects, classes that are implementing the `HandleCreator` interface, are registered at a factory object called `HandleFactory` together with an unique identifier, which is later used to identify the creator object. E.g. the `AxisMatchHandleCreator` object can create *AxisMatch* handles and may be registered at the `HandleFactory` together with the id "AxisMatch".

If the parser is in the handle creation state, because the last processed node was indicating the beginning of a handle collection, the parser extracts the identifiers of the following nodes and uses the handle factory object to create and correctly initializes the appropriate handle types. To add a new handle type using this technique, only the appropriate creator object together with the identifier must be registered at the factory.

6.1.3 Global Configuration Module

This module encapsulates and coordinates the distribution of all configuration relevant actions across the modeling system.

In order to provide an uniform way to access the functionality of the module, its central object called `GlobalConfigurationManager` is implemented according to the *Singleton Pattern* [GHJV95]. In general, the *Singleton Pattern* ensures that an object has only one instance and provides a global access point to it. This is done by using a static function to create the singleton class. This function creates a new instance of the singleton class if and only if no other instance of the class already exists. In case an instance of the class already exists, the static function just returns this instance. To make sure the class cannot be instantiated in any other way, its constructor is marked as private.

Each C++ object of the system as well as any object created at runtime using the *Scheme* language can obtain a pointer to the `GlobalConfigurationManager` using the static method `getInstance()`.

This object is responsible for the following configuration tasks:

1. User Management: Each user is represented in the system as a single object. Amongst others it contains the id and the set of tools of a user. The interface of the configuration module provides functionality to add and remove users from the system.

2. Tool Management: This interface provides functions to activate and to deactivate tools. The complexity of this operation is completely encapsulated in the configuration module. To alter the activation status of a specific tool, only the user id, the

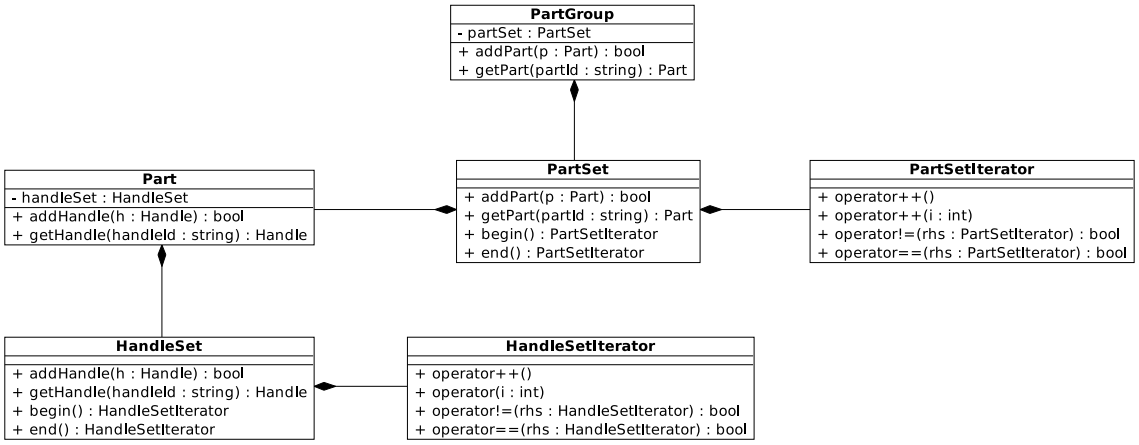


Figure 6.3: UML class diagram illustrating the context of the `Part` aggregation object called `PartSet` and the `Handle` aggregation object called `HandleSet`. Both aggregation objects provide an external iterator (`PartSetIterator` respectively `HandleSetIterator`) to access their content.

tool id and a boolean value indicating the tool state is needed. The configuration module then processes this request and validates it against all known tool change constraints.

3. Part Management: The functionality to add new parts together with their product configuration grammar at runtime to the VEAM system is encapsulated by this interface.

4. Widget Theme Management: All widgets used for system control tasks by the VEAM system support a configuration using theme objects. This interface provides functions to load and activate different widget themes at runtime.

6.1.4 Aggregation Objects

The VEAM system uses aggregation objects to encapsulate the functionality to manage the storage and manipulation of different types of objects. In general, the system needs to manage workspaces, part groups, parts and handles. A UML class diagram illustrating the context of these aggregate objects is shown in figure 6.3.

Each of these aggregation objects provide functionality to add and remove items together with an iterator interface to access their content. Since the VEAM system supports collaborative two-user modeling, concurrent access of these aggregation objects can occur at runtime. The concept of internal iterators is not capable of handling this, therefore external iterators are used to access the aggregation objects. The fundamental difference of the two iterator concepts is the question who is in control of the iteration process. In case the client that uses the iterator controls the iteration process, one speaks of an external iterator. If the iterator itself controls the iteration process, it is referred to as

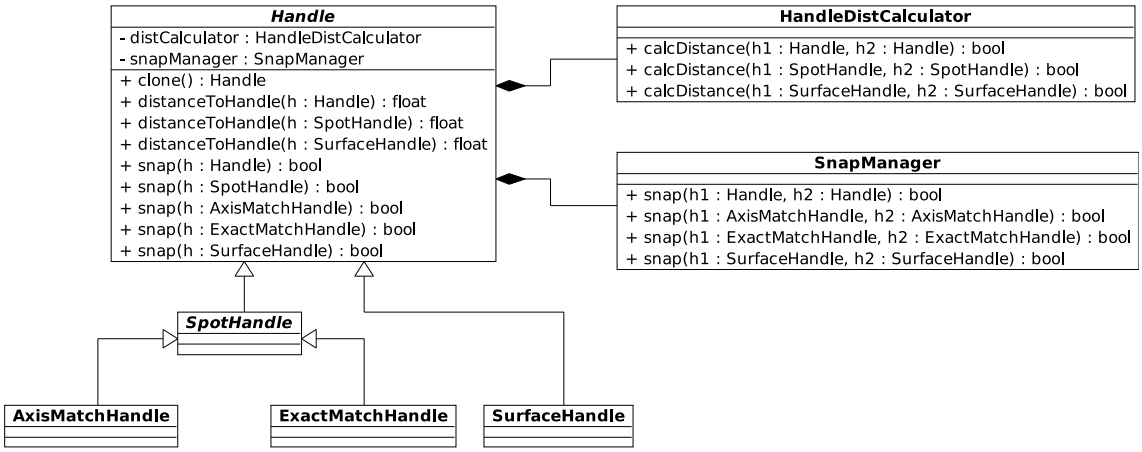


Figure 6.4: UML class diagram of the classes used for the handle distance and part snapping calculation. The complete logic of these two operations is encapsulated in the `HandleDistCalculator` and the `SnapManager` class and is not spread over the system. This reduces the coupling between the objects and provides a central point for potential extensions.

an internal iterator. A detailed description about the *Iterator Pattern* can be found in [GHJV95].

6.1.5 Assembly Module

The assembly module encapsulates the part assembly and disassembly algorithms described in section 4.7.

Amongst others, this module contains classes that are responsible to calculate the distance between handles, the snapping of part groups and the calculation of the interaction mode after the preliminary connection of two part groups was established.

Again the design of this module is aiming towards a high degree of flexibility and extensibility. One way to achieve this is to centralize the logic and the computation in a single or in few clearly defined objects instead of spreading the logic and computation among all involved objects.

This module uses different methods to achieve the centralization and encapsulation of the application logic. An example is the distance calculation between handles as well as the part snapping mechanism. Both operations depend on the type of the involved handles. E.g. the calculation of the distance between two *AxisMatch* handles differs significantly from the distance calculation of two *SurfaceMatch* handles. All handle types inherit the abstract `Handle` class and the system treats all handles as pointers to this base class, regardless of their actual runtime types. As mentioned above, the handle distance and the part snapping calculations depend on the runtime type of the arguments and therefore a

mechanism to dynamically determine the runtime type of objects is needed. This can be achieved by using a double dispatch mechanism. This mechanism dispatches a function call to different concrete functions depending on the runtime types of the involved objects in the function call. This technique is closely related to the *Visitor Pattern* [GHJV95]. The double dispatch mechanism is widely used throughout the system and not only for the handle distance and part snapping calculations, because it allows the separation of an algorithm from an object structure.

A UML class diagram showing the involved classes for the handle distance and snapping calculations is shown in figure 6.4. The `HandleDistCalculator` class is responsible of calculating the distance between all handle types known by the system. The `SnapManager` class calculates the snapping of two parts according to the involved handle types. The encapsulation of the logic in these two classes allows an easy integration of new handle types, since only these two classes and not all involved classes must be altered.

6.1.6 Tool Management Module

This module is responsible for the management of the tools of all registered users. Each user has her own set of tools, whereas multiple tools can be active at the same time. In order to do this, multiple toolholders must be used, because AVANGO™ supports only one active tool per toolholder. Due to this limitation, an object called `ToolCombo`, which consists of a tool and a toolholder is used by the system to manage all tool related functionality. For simplicity reasons, the term ”tool” is used in the following as a synonym for `ToolCombo`.

In some cases it is not reasonable that certain tools are active at the same time, if they use the same trigger mechanism. Since a user can manually configure the trigger of each tool using a configuration script this problem can even occur if the input device provides multiple buttons. To overcome this problem, the VEAM system uses a set of tool change constraints, which are applied before the state of a tool is changed. The system distinguishes between two different kinds of constraints:

1. Local Exclude Constraint: Using this kind of constraint only the active tools of the actual user will be checked. It is used to ensure that specific tools of one user are not active at the same time. An example for this is the simultaneous activation of a drag tool and a delete tool using the same trigger mechanism.

2. Global Exclude Constraint: This kind of constraint is checked against the active tools of all registered users. It used to guarantee that specific tools of all user are not active concurrently. This mechanism can be useful to avoid the situation where two users have an active global drag tool at the same time.

The module can be easily extended with other kind of constraints, because all constraints

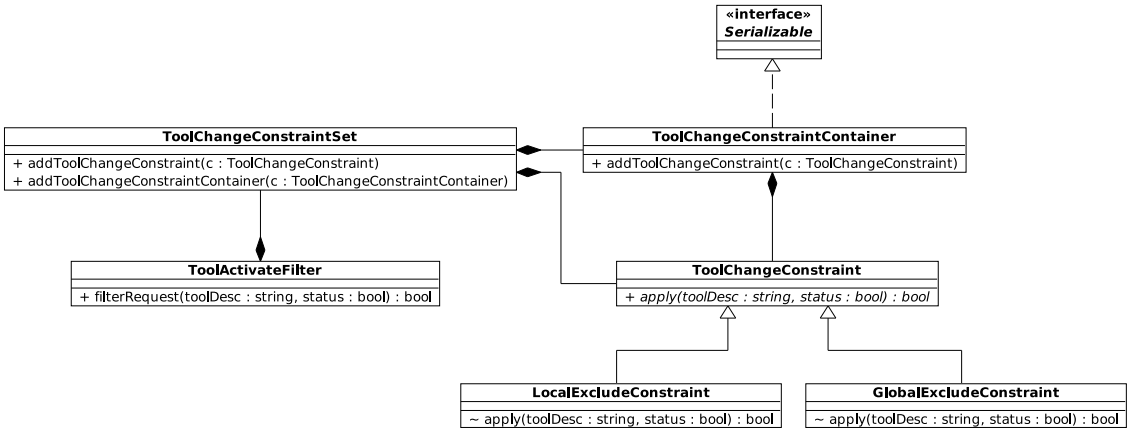


Figure 6.5: UML class diagram of the tool management module illustrating the usage of the *Template Pattern* for the application of the different concrete tool change constraints (represented by the *LocalExcludeConstraint* and *GlobalExcludeConstraint* class).

are inherited from the `ToolChangeConstraint` interface and the algorithms are only working with pointers to this interface. Figure 6.5 shows a UML class diagram containing the classes of this module.

Since the available tools can vary for different environments and they are configured by a script which is loaded at the start up of the program, a flexible mechanism to provide tool change constraints is needed.

All constraints are aggregated into a container class called `ToolChangeConstraintContainer`. This aggregation object provides functionality to serialize and deserialize its content into XML based archives. This archive in turn can be used as input for the system. In addition to this functionality of specifying tool change constraints in an XML file that can be dynamically added to the system, all constraints provide a *Scheme* binding that can be used to create and add constraints at runtime.

All user requests, this can either be an activation or a deactivation request to change the state of a tool, are first passed to a filter object before the system executes the requests. At the moment, the tool change constraints are only processed for tool activation requests. The filter object responsible for validating the tool change requests against all known tool change constraints is called `ToolActivateFilter.`

The behavior of this module is implemented according to the *Template Pattern* [GHJV95]. In general the *Template Pattern* encapsulates the structure of an algorithm (validating a tool activation or deactivation request) in a class (the activation filter object) and specific steps of the algorithm (application of the tool constraints) are defined as abstract template functions. This allows the refinement of the abstract template function, in this case the application of a tool change constraint, without the need to change the algorithm at all.

6.1.7 Process Chain Integration

This module is responsible for the integration of the VEAM system into an existing manufacturing process chain. The interface of this module provides functionality to access all information needed for the production of a model. Amongst others, these are the list of needed parts, their material information and the assembly graph.

No functionality for a generic integration of the system into different manufacturing process chain can be given, because different production chains might have distinct interfaces.

One way to allow the communication of modules with different interfaces is to use the *Adapter Pattern* [GHJV95]. The adapter object is responsible for handling the logic needed for the transformation of data from one format into another format that is suitable for the other interface. Thus, environment specific adapter modules are used for the integration of the VEAM system into a particular manufacturing process chain.

Currently an adapter module for the integration of the system into the INT-MANUS [FSM+06] manufacturing process chain is developed and implemented. In addition to the direct integration of the VEAM system into the manufacturing process chain, this module provides a network communication layer using the *Simple Object Access Protocol*[3] *(SOAP)* in order to offer a remote integration possibility. This allows the modeling to take place in a Virtual Environment installation regionally separated from the production environment.

6.2 VEAM Widget Toolkit

This section describes the implementation of the developed application independent widget toolkit. This toolkit is used to implement the system control functionality of the VEAM system.

At first, the requirements of the toolkit along with its general design principles are introduced, followed by a detailed description of the most important components. Again, all UML diagrams used in this section were reduced to the essential details for the sake of clarity.

6.2.1 Design Principles

The objective was to develop an object oriented, application independent three-dimensional widget toolkit for Virtual Environment applications using the AVANGO™ framework.

The widget toolkit is completely written with the *C++* programming language and in addition to its *C++* API it provides a binding to the interpreted *Scheme* scripting lan-

[3] Further information about the SOAP protocol can be found under: www.w3.org/TR/soap

guage. It does not claim to be a rich widget toolkit, but is rather an easy way to combine a Virtual Reality application with widgets for system control tasks. Using the provided *Scheme* scripting functionality, little code is needed to create simple three-dimensional user interfaces for Virtual Environment applications.

The architecture of the toolkit was especially designed with regard to extensibility. All functionality of the widgets is strictly encapsulated and can be accessed by clearly defined interfaces. This allows the creation of new widgets by combining and reusing existing widgets without the need of code duplication. In addition, custom widget implementations can be made, by overriding the default implementation of widgets in order to adapt the functionality to specific application needs.

The complete widget toolkit is implemented as a *white-box* framework with respect to transparency. In general, a *white-box* framework provides many abstract classes, that can be used to extend the framework with user defined functionality by deriving from these classes. A detailed characterization of *white-box* frameworks together with their advantages and disadvantages can be found in [Szy02].

Communication technique

Some design concepts and techniques used by this toolkit are inspired by the QT[4] widget toolkit. The QT widget toolkit is a comprehensive cross platform development framework for two-dimensional graphical user interfaces.

A flexible inter object communication mechanism is a crucial feature of every widget toolkit. The communication mechanism is used to trigger specific actions in case a widget or more general, an object of the toolkit changes its status. E.g. the activation of the close button should automatically trigger the closing function of the window. Most toolkits provide a callback mechanism for the inter object communication. A callback is a pointer to an executable code block, which can be passed as an argument to another function. Typically, a button has a callback function, that is called every time the button is pressed. However, callbacks have some serious shortcomings. They are not type safe, which means there is no way to guarantee, the callback will be called with the appropriate arguments. Another far more severe disadvantage of callbacks with respect to the software design is the strong coupling of the callback function and the processing function, since every processing function must specifically know which callback to call.

An alternative way to manage the communication between objects is the so called signals and slots technique. This communication technique was first introduced by the QT widget toolkit.

In general, a signal is emitted in case a specified event occurs. Signals can be connected

[4]More information about QT widget toolkit can be found under: www.trolltech.com

to an arbitrary number of slots. Slots can be seen as callback receivers and are called every time a connected signal is emitted. All signals and slots connections are managed and keep track of all established connections. Therefore, signals and slots connections can be dynamically changed at runtime. Another advantage of this technique is the loose coupling of the involved objects. The object that is emitting the signal does not have to know anything about objects whose slots are connected to this signal. The signals and slots communication technique can be seen as generalization of the *Observer Pattern* [GHJV95].

The widget toolkit uses the managed signals and slots implementation provided by the *boost*[5] libraries.

Theme support

All widgets in this toolkit support the usage of themes. A theme describes the look and feel of a widget and can be changed at runtime. Thus, users can adjust the user interface of the VEAM system to their needs without changing the application code at all. Amongst others properties, the theme contains information about the margins and paddings of the widgets, the text size, various color related information and default icons for some widgets types such as a checkbox or a radio button.

A theme is specified in an XML format and can be dynamically changed at runtime by the so called `ThemeManager`. The function `refreshDefaultTheme()` defined in the abstract `Widget` class queries the `ThemeManager` for the current active theme and updates the cached themeable properties of the widget. Since there is only one `ThemeManager` at all times and it must by accessible from every component in the system, it is implemented according to the *Singleton Pattern*.

Widget status

In some cases it is useful to disable a widget. This is done by the `setDisabled(bool dis)` function defined in the abstract `Widget` class. A disabled widget does not handle any interaction events like selection or activation of the widget. In case the widget is a container widget, it implicitly enables or disables all of its children widgets.

To provide a visual feedback for the users about the activation status of a widget, a disabled widget is drawn in a special disabled mode. In case of a push button, the label and the icon are grayed out. The exact drawing style of a disabled widget is specified by the current widget theme.

[5] *Boost* is a collection of peer-reviewed libraries that extend the functionality of *C++*. For more information about the *boost* libraries look at: www.boost.org

Fading

All widgets provide the possibility to adjust its alpha value. The `multAlphaFac(float fac)` function in the abstract `Widget` class multiplies the actual alpha value of the widget with the given factor. In case the widget is a container widget, this computation implicitly affects the alpha values of all its children widgets. This functionality allows the implementation of visual gradual fading mechanism for widgets. The fading mechanism is used while displaying and hiding the context menu. This reduces the visual distraction of the user, since the menu will eventually occlude parts of the scene.

6.2.2 Plane Widget Hierarchy

One of the two major categories of widgets provided by the toolkit are adapted two-dimensional widgets (see figure 3.5 and 5.5).

The base class of all plane widgets is called `PlaneWidget`. The system treats all plane widgets at runtime as instances of this class, independent from their actual type. Thus, the `PlaneWidget` class contains virtual functions that can or must be overridden by specific subclasses.

The most important mechanisms and features are described in the following subsections.

Reusability and Extensibility

The architecture of the toolkit was designed towards a high degree of reusability and extensibility. Common functionality is propagated towards the top levels of the inheritance hierarchy in form of abstract classes and virtual functions. The `AbstractButton` class is an example for this. This abstract class encapsulates all functionality related to button style widgets, like handling selection or activation events. It provides various signals, like the `sigMouseOverConnected()` signal indicating a selection of the widget or the `sigPressConnected()` signal reporting an actual activation of the widget. All of these signals can be used to implement the application logic, independent from the actual runtime type of the widget. Subclasses can be used to specify the appearance of the button and add additional properties, like a checked status. This allows an efficient implementation of widgets which need a selection and activation functionality like a push button, a checkbox or a *Quikwrite* entry.

This high degree of generalization is used throughout the whole `PlaneWidget` inheritance hierarchy and not only for button like widgets, but also for other widgets, like text entry widgets, where the common functionality of the *Quikwrite* and *Cirrin* widget is implemented in the abstract `TextEntryWidget` class. The *Quikwrite* and *Cirrin* widget of this toolkit are shown in figure 5.6.

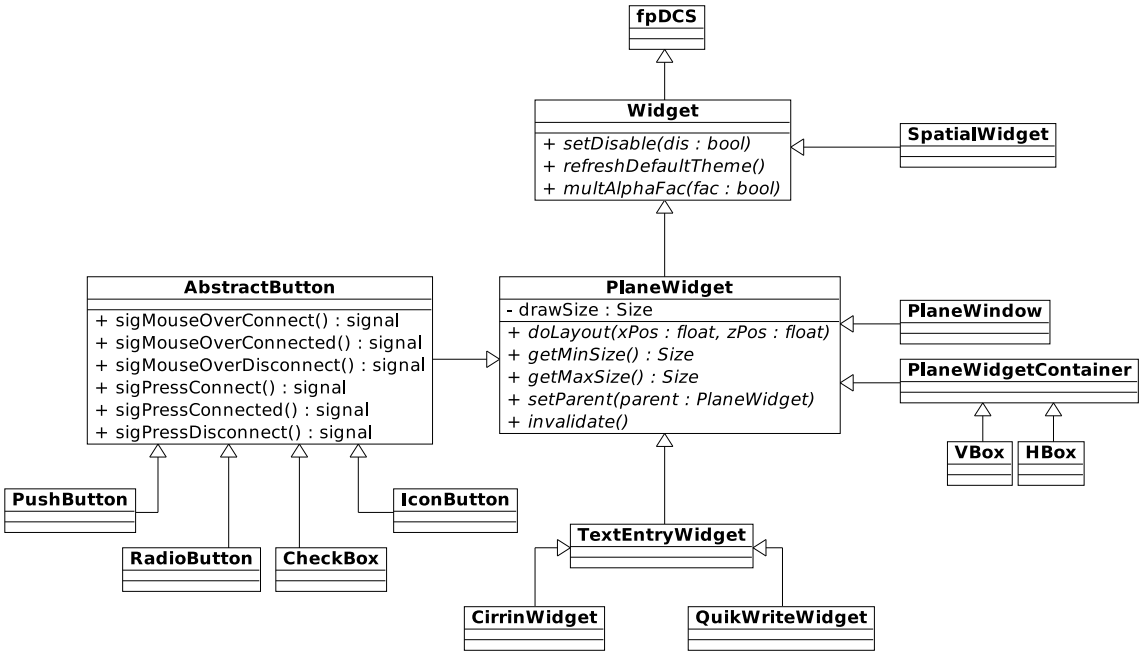


Figure 6.6: UML class diagram of a part of the `PlaneWidget` inheritance hierarchy. Common functionality is propagated towards the top levels of the hierarchy. E.g. all button related functionality, like the selection or the activation of a button is provided by the `AbstractButton` class. All subclasses such as the `PushButton` and the `Checkbox` class can immediately use this functionality and add additional properties to specify their appearance and functionality.

A UML class diagram containing a part of the `PlaneWidget` inheritance hierarchy is shown in figure 6.6.

Layout mechanism

As mentioned in section 5.3.1, a requirement of the widget toolkit was the support of a recursive box layout system.

To provide such a layout system, the widget toolkit uses two different widget container classes. A container with a vertical layout called `VBox` and a container with a horizontal layout called `HBox`. Both containers can contain an arbitrary number of children widgets. Since the container widgets are also inherited from the abstract `PlaneWidget` class, they can be recursively added to other containers.

The complete layout mechanism solely operates on the interface of the `PlaneWidget` class. Each plane widget provides a function `getMinSize()` to query its minimal extend and a function `getMaxSize()` to query its maximal extend. In case the widget is an instance of a `PlaneWidgetContainer`, it is recursively queries its children widgets for their minimal and maximal size. The layout calculation is done by the `doLayout(float xPos, float yPos)` function. Again, this is a recursive function which calls `doLayout(float xPos, float`

`yPos)` of all children widgets in case the widget is an instance of a container widget. Often, the layout of a widget need to be altered at runtime, due to dynamic content changes. E.g. an additional widget has been added to a container, or the text label of a button was changed. In these cases, not only the layout of the widget where the changes occurred, but also the layout of all its parent widgets is affected and need to be recalculated. Thus, each widget must know its parent widget, in order to propagate the layout request to the top level of the widget parent hierarchy. The parent of a widget is set using the `setParent(PlaneWidget *w)` function. In case a layout change of a widget occurs, the `invalidate()` function automatically propagates the layout request to the top level widget, in most cases an instance of the `PlaneWindow` class, which in turn initiates a new recursive layout call.

6.2.3 Spatial Widget Hierarchy

The other major class of widgets are 3D widgets. The VEAM system uses several 3D widgets for different system control tasks (see figure 5.7 and 5.8a). In the following, the term spatial widget is used as a synonym for 3D widget.

The abstract class `SpatialWidget` is the base class of all spatial widgets of this toolkit. At runtime, the system treats all 3D widgets as pointers to this abstract class. It provides virtual functions, which can be overridden by concrete subclasses to implement the concrete behavior. An example for this is the calculation of the bounding box or the layout calculation of the widget.

Reusability and Extensibility

The inheritance hierarchy of the spatial widgets aims towards a high degree of generalization. However, the functionality of two 3D widgets is commonly quite different and therefore not the same high degree of generalization can be achieved for the spatial widget inheritance hierarchy as for the planar widget hierarchy described in the previous section.

A UML class diagram of a part of the `SpatialWidget` inheritance hierarchy is shown in figure 6.7.

Layout Mechanism

In contrast to the recursive box layout for adapted two-dimensional widgets, spatial widgets commonly do not need such a recursive layout mechanism. Every spatial widget uses its own layout mechanism, due to the greatly varying appearance and functionality of the spatial widgets.

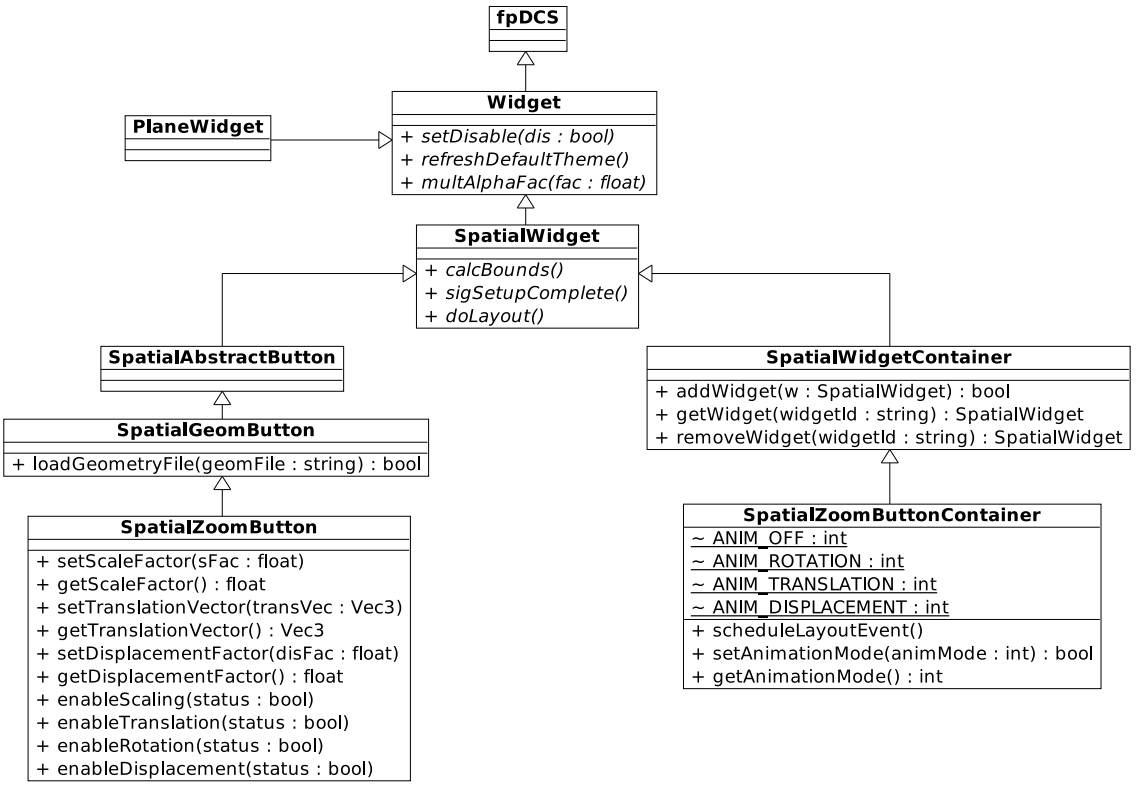


Figure 6.7: UML class diagram of a part of the `SpatialWidget` inheritance hierarchy. The hierarchy aims towards a high degree of generalization by implementing common functionality in abstract top level classes, like the `SpatialWidget` or the `SpatialWidgetContainer` class, which encapsulates the basic functionality of a container for spatial widgets.

The system provides the abstract class `SpatialWidgetContainer` to encapsulate the functionality to add and remove spatial widgets to a container widget. This abstract class is solely an aggregation object and does not apply any layout calculations to its children widgets. Concrete subclasses of this abstract class can then be used to implement specific layout calculations.

A spatial widget used by the VEAM system, which needs a dynamic layout mechanism, is the part selection widget described in section 5.3.2. This widget contains miniature models of all parts known by the VEAM system and per default arranges them in a vertical row. These miniature models can be directly selected in order to add the corresponding parts to the workspace.

The `SpatialZoomButtonContainer` implements this layout mechanism and all miniature models are represented by `SpatialZoomButton` objects. Since, all spatial widgets are derived from the abstract `SpatialWidget` class, the `SpatialZoomButton` provides the signal `sigSetupComplete()` which is emitted as soon as the setup process of the widget is completed. This signal is important for the layout calculation, because the `SpatialZoomButton` uses a geometric model defined in a file for its visual representation and most Virtual

Reality frameworks, including AVANGO™, are using an asynchronous file loading mechanism to ensure real time functionality. However, the layout mechanism implemented in the `SpatialZoomButtonContainer` widget needs to know the exact extend of all children widgets and in turn, the extend of a `SpatialZoomButton` widget can only be calculated after the file containing the geometry has been loaded and processed.

The `SpatialZoomButtonContainer` widget offers different mechanism to highlight the selected widget. These highlight modes, such as a rotation, translation or displacement of the selected widgets can be set using the `setAnimationMode(int mode)` function.

7 System Evaluation

This chapter presents the evaluation of the developed VEAM system and the analysis of the conducted user study.

The first section introduces the relevant terminology for this evaluation. In the second section, the general purpose of an evaluation and the specific objectives of this evaluation are described. The structure, the task description and the actual test procedure of the conducted user study are described in the third section. The last section discusses the results and draws conclusions of the evaluation.

7.1 Terminology

The general objective of an evaluation is the analysis, the review and the testing of an artifact. This provides a possibility of comparison and should contribute to the identification of usability issues and it could lead to potential changes of the artifact. A metric system is needed to identify and measure these usability issues. In [BKJP05] usability metrics are distinguished between *system performance*, *user task performance* and *user preference*.

System performance refers to computer or graphical system performance. Typically measured properties are the frame rate or the latency of the system. From the modeling system point of view, the *system performance* metric is not relevant for this evaluation.

The *user task performance* denotes the quality of the user performance of a specific task. Commonly, the most important measured properties are speed and accuracy of a task. For most tasks there is an implicit relationship between speed and accuracy. On the one hand, a user can perform the task quicker, which might result in a decline of the accuracy. On the other hand, a user can increase the accuracy by reducing the speed.

The user specific perception and handling of the evaluated artifact is called *user preference*. Typically this metric includes properties like the comfort of use or the learning curve of an artifact. The collected data is highly subjective and is often measured via interviews or questionnaires.

The two major roles in every usability evaluation are the *evaluator* and the *user*. An evaluator is a person who designs and analyzes the evaluation. The user is the person who

participates in the evaluation and performs specific tasks or answers specific questions.

7.2 Purpose and objectives

One of the most fundamental insights of human computer interaction research is that even the most carefully designed technique can still go wrong. Thus, despite of the great effort spent on the development of the VEAM system, on the design of its interaction techniques and on the selection of appropriate input devices, an evaluation of the system is needed.

In general, this evaluation is carried out in order to determine and classify the usability of the developed modeling system and it had two specific objectives.

The first objective was to determine and compare the *user task performance* and the *user preferences* of the VEAM system with a reference modeling system.

The evaluation of the collaborative two-user modeling techniques of the VEAM system was the second objective. This was done by a direct comparison of the *user task performance* of a single user and a collaborative two-user assembly task.

A questionnaire was developed to collect the personalized data of the *user preferences*. The *user task performance* was measured by means of task completion time of a specific assembly task.

7.3 User Study

This section describes the user study conducted within this evaluation. First, the choice of the reference modeling system used for the comparison of the *user task performance* is explained, followed by the description of the modeling tasks which had to be accomplished by the user and finally, the realization of the user test is described.

7.3.1 System Selection

The selection of the reference modeling system, which should be used for the comparison of the *user task performance*, was one of the key decisions in this evaluation. The first idea was to select one of the four related immersive modeling systems described in section 4.1. Unfortunately only publications containing the description of the functionality and the performance of these systems and no source code nor executables were available. Therefore, none of these systems could have been used for the comparison.

The developed VEAM system is capable of modeling with arbitrary geometric models. Since LEGO parts were used during this thesis to illustrate and describe its features, a

specialized desktop based LEGO modeling system was used as reference. The system is called LEGO *Digital Designer* (LDD)[1] and provides specifically tailored interaction techniques and user interfaces for LEGO modeling with a desktop system.

7.3.2 Task Description

All participants in the user study had to solve a specific LEGO assembly task. The given assembly task involved several docking tasks. A docking task includes the positioning and the alignment of an object at a specified target location. Docking tasks are frequently used to evaluate the performance of three dimensional interaction techniques and input devices [Zha98].

In the given assembly task, the participants had to assemble a table out of a table plate and four table legs and place it on a specific position on a floor plate. This involves five docking tasks. Four docking tasks are required for the assembly of the table legs and the table plate, whereas the orientation of the table legs is irrelevant. The fifth docking task is the placement of the assembled table on the floor.

The complete assembly task is described in the VEAM system using the product configuration grammar introduced in section 4.5. The table plate is equipped with four *AxisMatch* handles representing the docking locations for table legs. In turn, the top and the bottom of each table leg is also equipped with an *AxisMatch* handle. These handles are used to connect the table legs with the table plate and with the floor plate. Four *AxisMatch* handles on the floor plate are defining the docking location of the assembled table. An illustration of this assembly task within the VEAM system is shown in figure 7.1a.

The product configuration grammar defines matching relationships between all involved handles. The first matching relationship is defined between the handles of the table plate and the top handles of the table legs. Each of the bottom handles of the four table legs are configured to be compatible with the handles on the floor using a second handle relationship. These generic rules are sufficient to define all valid assembly configurations for this task.

The LDD modeling system uses a similar set of rules for the modeling like the VLEGO system described in section 4.1. Since the LDD system is designed for handling only LEGO parts it heavily exploits the regular structure of these parts. Amongst other restrictions, all parts can only be translated on a grid, which is defined by the regular stud structure of the LEGO parts and all rotations of the parts are constrained to be exactly 90 degrees.

[1] Further information about the LEGO *Digital Designer* can be found under: http://ldd.lego.com

7.3.3 Test Procedure

The user study is carried out in a laboratory at the *Fraunhofer IAIS*. Nine users participated in this user study. They were all research scientists or student assistant researches in the Virtual Environment department at the *Fraunhofer IAIS*. All of them were male and in the age of 28 to 46 years. They all had somehow experiences with Virtual Reality applications. Some of them were experts and others were novice Virtual Reality users. All of them completed the entire user tests.

The same gender of all participants and the close age group considered should allow an easy pooling of the participants.

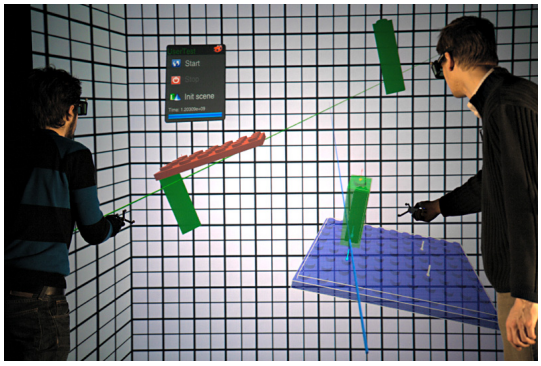
The users had to complete three different assembly tasks. The first task, in the following denoted as T_1, was the single user assembly task using the VEAM. The second task, T_2, was the same assembly task in the VEAM system, however two-users now worked collaboratively on this modeling task. To create equal conditions for all users during the task T_2, all participants worked collaboratively with the same independent user. In task, T_3, the users had to perform the assembly task with the LDD system. All users had to do five repetitions of all three assembly tasks, in order to identify outliers and to obtain meaningful results. They all completed the assembly tasks in the following sequence: T_1, T_2 and then T_3.

Prior to each of the three assembly tasks, the users had the possibility to practice the particular task. There was no limit or dictation of the duration of the training time for each task, in order to take the different experience of the users into account. To introduce as little distraction as possible, only the evaluator and the test user were present in the room during the entire user test.

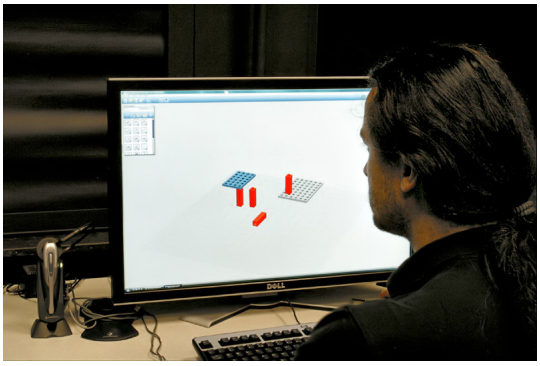
The VEAM system uses the *TwoView* system (see section 2.1.1) as visual display and the *Wiimote* (see section 5.1) as physical input device. The LDD modeling system is installed on a commodity personal computer equipped with a 30 inch LCD monitor as visual output device and a standard keyboard and a mouse as input devices.

After the completion of all repetitions of the assembly tasks, the users had to fill out a two-paged questionnaire, which contained questions about their subjective impression of the VEAM and LDD modeling system in respect to the performed assembly tasks (for details about the questionnaire see appendix B). All of these questions could be marked with school grades ranging form one to five, whereas one corresponds to the best and five to the worst possible grade. At the end of the questionnaire, the users could make written free text annotations and suggestions for further improvements and express criticism.

The duration of the complete user test, including the time needed for the completion of the questionnaire was about 20-30 minutes per user. During the entire time, the evaluator took written notes to document any noticeable events or problems.



(a) Two users during the training phase of the collaborative two-user modeling task within the VEAM system.



(b) Single user during the training phase with LDD modeling system.

Figure 7.1: Two images taken at the evaluation process of the VEAM and the LDD system. The objective was to assemble a table out of four table legs (green in the VEAM and red in the LDD system) and a table plate (red in the VEAM and blue in the LDD system) and place it at a specific position on the floor plate (blue in the VEAM and gray in the LDD system).

Figure 7.1a shows a sample scene of the collaborative two-user assembly task within the VEAM system. In figure 7.1b the single user modeling task with the reference LDD modeling system is shown.

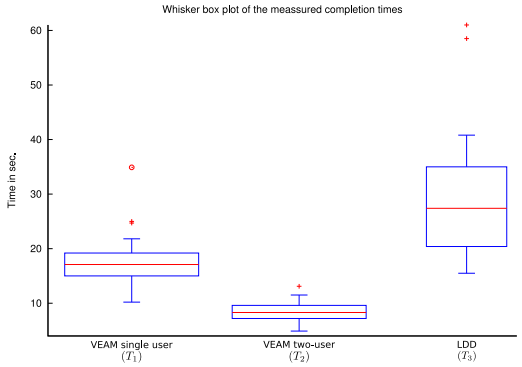
7.4 Results

The sequence of the assembly tasks (T_1, T_2, T_3) was kept constant for all users. There might be a certain influence of the constant order of the assembly tasks on the *user task performance*. Due to this and the limited number of users that participated in the study, the generality of the measured *user task performance* might not be given.

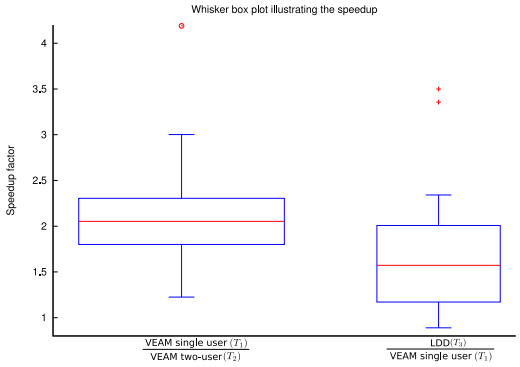
7.4.1 User Task Performance

The first objective of the user study was to determine and compare the *user task performance* of the VEAM system with the LDD reference modeling system. 75% of the user completed the VEAM single user assembly task, T_1, in less than 19 seconds and moreover, the variance of the measured data is small. For the LDD modeling task, T_3, the users needed more time to complete the task. Only 28% of the users completed the assembly task in the same time span needed for the task T_1. All 72% of the users needed significantly more time. In addition, the variance of the measured data for task T_3 is relatively large compared to the variance of the measured data of task T_1.

The evaluation of the collaborative two-user modeling techniques of the VEAM system



(a) Box plots illustrating the measured task completion times. The given assembly task could be solved with the highest effificeny by the VEAM collaborative two-user modeling (median 8.3 sec.) followed by the VEAM single user modeling (median 17.1 sec) and the LDD modeling (median 27.4 sec).



(b) Box plot illustrating the speedup factors of the different assembly tasks. The left plot shows that the VEAM collaborative modeling was twice as fast as the VEAM single user modeling and the right plot shows that the speedup factor of the VEAM single user modeling in respect to the LDD modeling was approximately 1.5.

Figure 7.2: Two whisker box plots illustrating the *user task performance* measured in the user test.

was the second objective of the user study. This was done by a comparison of the *user task performance* of the VEAM single user assembly task, T_1 and the VEAM collaborative two-user task, T_2. All users completed the assembly task T_2 in less than 12 seconds and over 75% of the users completed the task in less than 10 seconds and therefore needed less time than anybody needed for the completion of T_1. This clearly shows that the given assembly task could be solved with the highest efficiency by the VEAM collaborative two-user modeling.

In figure 7.2a the *user task performance* of the three assembly tasks T_1, T_2 and T_3 is graphically illustrated with a whisker box plot. A whisker box plot is a traditional way to graphically visualize numerical data without making any assumptions about the underlying statistical distribution of the data. It uses a five number summary for each data set. This set includes the value of the smallest and largest observation, the median and the lower quartile, Q_1, (cuts of the lowest 25% off the data) and the upper quartile, Q_3 (cuts of the highest 25% of the data). The interquadrile range, IQR, is also called the *middle fifty* and corresponds to the difference between the first and the third quartile ($IQR = Q_3 - Q_1$). In the used whisker box plots, the IQR is drawn as a rectangle. The IQR is used to discover outliers in a data set. Commonly, all observations which are lying more than $1.5 \cdot IQR$ lower than Q_1 or higher than Q_3 are considered as mild outliers. Observations that are lying more than $3 \cdot IQR$ lower than Q_1 or higher than Q_3 are treated as extreme outliers. Mild outliers are shown as "+" and extreme outliers are marked with "o" in the used whisker box plots.

The diverse size of the IQR of the measured data of the three modeling tasks is an indicator for the usability of the evaluated modeling systems. In case the IQR of the data is small,

all users roughly needed the same time to complete the assembly task. A large IQR implies that some users accomplished the task much faster than others, which can be an indicator for potential usability problems. The size of the IQR of the completions times for the assembly tasks T_1 and T_2, which both used the VEAM system is relatively small. This indicates that the usability of the single user and the collaborative two-user modeling techniques of the developed VEAM system is quite good. The IQR of the LDD modeling task T_3 is relatively large, due to the high variations of the needed task completion times of the users.

The whisker box plot in figure 7.2b shows direct speedup factors of the completion time of two assembly tasks. The speedup factor of the VEAM collaborative two-user modeling to the VEAM single user modeling is approximately 2. This implies that the VEAM collaborative two-user modeling is very efficient and well suited for this particular assembly task. Also, the VEAM single user modeling provides a speedup of factor 1.5 compared to the LDD modeling.

7.4.2 User Preferences

Directly after the completion of the three assembly tasks, the users had to fill out a questionnaire. This questionnaire was used to collect the individual and personalized data of the *user preferences*. The questionnaire including the average grades and the standard deviations of the grades of the answers can be found in appendix B.

All users, and even the novice Virtual Reality users clearly preferred the VEAM system with respect to the LDD system to solve the given assembly task.

The visual feedback given by the VEAM and LDD system was classified as extremely helpful and important during the modeling process. The users favored the feedback given by the VEAM system.

The VEAM collaborative two-user modeling was ranked as very intuitive and natural by all users. Consistent with the measured *user task performance*, the users preferred VEAM collaborative two-user modeling over VEAM single user modeling and stated that it was much more efficient for solving the given assembly task.

The experience of the study participants varied in respect to Virtual Reality applications and desktop modeling systems. On average, they use general Virtual Reality applications slightly more often than desktop modeling systems.

The conclusions drawn from the *user task performance* measurements could be verified by the collected personalized data from the questionnaire. In particular the statement, that the developed VEAM system is better suited for the given assembly task than the LDD system and that the VEAM collaborative two-user modeling is the most efficient assembly

method.

7.4.3 Comments and Observations

By providing the possibility make comments and express criticism at the end of the questionnaire and by taking notes during the execution of the assembly tasks, interesting observations have been made.

A noteworthy observation was that almost all users used the same assembly sequence when using the LDD system. They first completed the assembly of the table and then put it on the floor. In contrast to this, no general assembly sequence could have been identified when the users used the VEAM system. Some users first assembled the table and then put the table on the floor plate, whereas other user first connected the table legs with the floor plate and then put the table plate onto the table legs.

Most users stated, that the VEAM system is very intuitive and easy to use. In particular the users mentioned that the collaborative two-user modeling was very natural, easy to use and did not require any explanation.

The visual feedback of the VEAM system was considered to be good. One user suggested to display a wireframe model of the part prior to the snapping in order to provide a visual feedback before the snapping is applied.

The choice of the *Wiimote* as an input device for the immersive modeling system was confirmed by some users. They considered these devices to be very well suited for the interaction techniques of the VEAM system.

The part snapping mechanism of the LDD system depends on the perspective of the virtual camera. This was problematic for some users, because they had to alter the perspective by using the keyboard or the mouse before the snapping of the part to the desired location could be executed. In addition, a couple of users were distracted by the used parallel projection.

Nearly all users and especially users wearing eyeglasses complained about the wearing comfort of the used *CrystalEyes*[2] shutter glasses. Mostly their weight and the loose seat was criticized.

Further user comments and observations made during the user test can be found in appendix B.

[2]For more information about the *CrystalEyes* shutter glasses look at: www.reald-corporate.com/scientific

8 Conclusion and Future Work

In this thesis the concepts and the development of an immersive modeling system called VEAM *(Virtual Environment Assembly Modeler)* is described. This does not only include the development of novel assembly algorithms, but also the design and implementation of innovative selection, manipulation and system control interaction techniques.

The VEAM system provides rapid prototyping and product customization functionalities, which are extensively used by the automotive industry where a broad range of products is derived from a relatively small set of basic mature product models.

The developed assembly algorithms do not rely on the geometric representation of the parts as a criterion for possible part connections, since every polygonal geometric model is only an approximation of a real geometry and therefore does not represent the part correctly. Instead, the VEAM system uses a grammar-based model to encode the connectivity information of parts. The basis of this grammar model is the so-called *Handle* concept. A handle is an attribute of a part that specifies its connection possibilities in conjunction with various connection semantics. Different types of handles are implemented, each defining its own connection semantics. Matching relations between different handles are specified by adding rules to the grammar module of the VEAM system. In general, handles are used to define the set of models possible to construct out of a set of parts. Moreover, they assist the user during the actual process of assembling these parts.

The connection information of assembled models is stored in a data structure called *ConnectionGraph*. One fundamental property of the *ConnectionGraph* is its interconnection characteristic. Every *ConnectionGraph* of a valid model must fulfill this property. After each modeling operation which might result in a violation of this property, like the part disassemble operation, the VEAM system applies algorithms to ensure the validity of the interconnection property of all models.

A key feature of the VEAM system is the possibility of integrating the system into existing manufacturing process chains. For this purpose, the system provides an interface which offers the functionality to access the required information, such as the list of needed parts, their material information and the assembly graph. The integration into the INT-MANUS [FSM+06] manufacturing process chain was accomplished and successfully tested in the advanced machining laboratory at the *Centro Ricerche Fiat* [1] in Turin.

[1] More information about the *Centro Ricerche Fiat* can be found under: www.crf.it

Another integral part of this thesis was the development of innovative selection, manipulation and system control techniques for the VEAM system.

The VEAM system uses stylus-type devices, like the *Wiimote* or wireless presenters as physical input devices. However, the VEAM system is not restricted to this type of physical input devices, because it is built on top of the Virtual Reality framework AVANGO™ and therefore all physical input devices supported by AVANGO™ can be used.

The system provides a pick ray technique for the selection of single virtual objects and a conical group selection technique for the selection of multiple virtual objects at once. Virtual objects can be manipulated by diverse transformation techniques. Among other things, these techniques provide functionality for isomorphic and nonisomorphic dragging as well as for in-place object rotations.

The modeling system does not only offer single user functionality, but also supports collaborative two-user modeling. During the collaborative modeling, all single user tools can be used and the system additionally provides special two-user tools like a part hand-over tool.

For system control tasks, the VEAM system uses embedded graphical menus. To avoid or to at least reduce the cluttering of the workspace with menus, context menus are used instead of global menus. The content of the context menu refers to the actual user context, which in turn is defined by the active tool and the selected object.

Possible occlusion effects caused by the context menus are reduced by drawing the menus in a semitransparent mode. The visibility and the usability of the context menus is ensured by a rule based placement policy for all initial menu positions.

The widget toolkit developed provides a variety of different two-dimensional widgets, such as checkboxes, push buttons, radio buttons, sliders, file dialogs and widgets for handling symbolic input. All widgets offer a theme interface which can be used to adapt the look and feel of the widgets.

In addition to the class of two-dimensional widgets, the toolkit provides a set of 3D widgets. These widgets exploit the full degrees of freedom of the three-dimensional environment and can be seen as combination of geometry and behavior.

An example of a 3D widget used by the VEAM system is the disassembly widget. In most cases the selection of a model implies the start of a new assembly operation. However, this is not the only possible interpretation of this task. The user can as well intend to disassemble the part from the model. Instead of the selection of the disassembly tool from a graphical menu and the successive selection of the part to be disassembled, the collocated 3D disassembly widget of this part can be selected and the part will immediately be unhinged from the model. Thus, the selection and execution of a command form a fluent action sequence.

The developed modeling system has been evaluated and a first user study has been performed. This evaluation had two distinct objectives. The first objective was to determine and compare the *user task performance* and the *user preferences* of the VEAM system with a reference modeling system called LEGO *Digital Designer* (LDD). The second objective was to compare the *user task performance* of single user and collaborative two-user assembly in the VEAM system.

All participants of the user study had to perform the same assembly task which involved at least five docking tasks. A comparison of the task completion times showed that all users were able to solve the given assembly task significantly faster with the VEAM system than with the reference modeling system. The speedup factor of the VEAM collaborative two-user assembly with respect to the VEAM single user assembly was approximately two. This speedup factor implies that the developed collaborative two-user modeling is user friendly and well suited for assembly tasks. It is important to note, that the generality of the *user task performance* analysis cannot be guaranteed due to the limited number of participants in this user study.

The VEAM system developed in this thesis can be seen as a strong foundation for future work in the area of immersive assembly based modeling in Virtual Environments.

One possible area of future work is the improvement of the handle editing functionality of the VEAM system. Currently, handles can be defined by XML files, which are parsed by the system and by *Scheme* scripts. These scripts are interpreted by the system and can be used to dynamically assign handles to parts at runtime. However, a user has to leave the Virtual Environment in order to use these mechanisms. Thus, it would be beneficial to have graphical mechanism for creating and modifying handles directly in the Virtual Environment. The development of such an immersive handle editing system in a Virtual Environment is a challenging tasks itself, because it requires high precision interaction techniques for the placement and the alignment of the handles.

The XML language describing the product configuration grammar currently only supports the explicit definition of handles. This is sufficient for parts with few handles, but is cumbersome for parts with a lot of handles, which are positioned in a regular structure. A rule based mechanism to automatically generate and place handles would be extremely helpful for the users in such situations and would help to significantly reduce the time needed for the description of new parts. An example for a possible rule would be a grid positioning rule. On all lattice points of the grid, a new handle would be created and positioned.

A frequently used operation during a modeling process with the VEAM system is the part disassembly operation. The current implementation of the part disassembly tool only allows the disassembly of a single part from the model at a time. Thus, in case a group of parts should be disassembled from the model, each part must be selected and

unhinged individually. Since all connectivity information of a model is encapsulated in the *ConnectionGraph*, intelligent part selection and group disassembly techniques, similar to the part separation techniques described in [OS03], could be used to overcome this limitation.

Due to the integration of the VEAM system into an existing manufacturing process chain, the system does not allow the shape modification of any modeling primitive, because the INT-MANUS [FSM+06] process chain needs to precompute the CNC programs for all primitives. However, there are approaches to allow an on-the-fly generation of CNC programs in this manufacturing process chain. In case these attempts are successful, the VEAM system could be extended with a *Constructive Solid Geometry (CSG)* [LTH86] module for geometry modification. For this purpose, first preliminary tests with an integration of the *Open CASCADE*[2] library into the VEAM system were done and already showed promising results.

The evaluation and the user study have shown that the VEAM system developed in the thesis is an intuitive and efficient platform for single user and collaborative two-user assembly based modeling in Virtual Environments.

[2] O*pen CASCADE* is a powerful software development kit providing CAD modeling functionality. Further information can be found under: www.opencascade.org

Bibliography

[BC03] Grigore C. Burdea and Philippe Coiffet. *Virtual Reality Technology, Second Edition with CD-ROM.* Wiley-IEEE Press, June 2003.

[BH97] Doug A. Bowman and Larry F. Hodges. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 35–ff., New York, NY, USA, 1997. ACM.

[BJH99] Doug A. Bowman, Donald B. Johnson, and Larry F. Hodges. Testbed evaluation of virtual environment interaction techniques. In *VRST '99: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 26–33, New York, NY, USA, 1999. ACM.

[BJH+01] Allen Bierbaum, Christopher Just, Patrick Hartling, Kevin Meinert, Albert Baker, and Carolina Cruz-Neira. Vr juggler: A virtual platform for virtual reality application development. In *VR '01: Proceedings of the Virtual Reality 2001 Conference (VR'01)*, page 89, Washington, DC, USA, 2001. IEEE Computer Society.

[BKJP05] Doug A. Bowman, Ernst Kruijff, Joseph J. Laviola Jr., and Ivan Poupyrev. *3D User Interfaces: Theory and Practice.* Addison-Wesley, Boston, MA, USA, 2005.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* The MIT Press, Cambridge, MA, USA, September 2001.

[CNSD93] Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the cave. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 135–142, New York, NY, USA, 1993. ACM.

[Cox03] Harold Scott MacDonald Coxeter. *Projective Geometry (2nd Edition).* Springer, Berlin, Germany, 2003.

[CW95] Michael Cohen and Elizabeth M. Wenzel. The design of multidimensional sound interfaces. pages 291–346, 1995.

[DH04] M. Pauline Baker Donald Hearn. *Computer Graphics Using OpenGL (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004.

[Dod05] Neil A. Dodgson. Autostereoscopic 3d displays. *Computer*, 38(8):31–36, 2005.

[Dre07] Armin Dressler. Benutzergerechte menügestaltung fÃŒr virtuelle umgebungen im expertenkontext. Technische Universität Ilmenau, Diplomarbeit, 2007.

[Dyb96] R. Kent Dybvig. *The Scheme Programming Language: ANSI Scheme*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1996.

[FDL01] Martin Holmberg Fabrizio Davide and Ingemar Lundström. Virtual olfactory interfaces: Electronic noses and olfactory displays. pages 193–219, 2001.

[FHZ96] Andrew Forsberg, Kenneth Herndon, and Robert Zeleznik. Aperture based selection for immersive virtual environments. In *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 95–96, New York, NY, USA, 1996. ACM.

[Fit54] P. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, 1954.

[FMS93] Steven Feiner, Blair Macintyre, and Dorée Seligmann. Knowledge-based augmented reality. *Commun. ACM*, 36(7):53–62, 1993.

[FSM+06] M. Foursa, T. Schlegel, F. Meo, A. Herrmann Praturlon, J. Ibarbia, S. Kopácsi, I. Mezgár, D. Sallé, and F. Hasenbrink. INT-MANUS: revolutionary controlling of production processes. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Research posters*, page 161, New York, NY, USA, 2006. ACM.

[FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[FWd+08] Maxim Foursa, Gerold Wesche, David d'Angelo, Manfred Bogen, and Rainer Herpers. A two-user virtual environment for rapid assembly of product models within an integrated process chain. To be published in: *Proceedings of X Symposium on Virtual and Augmented Reality*, 2008.

[GB04] Domenique Gerber and Domenique Bechmann. Design and evaluation of the ring menu in virtual environments. In *8th International Immersive Projection Technologies Workshop*, 2004.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[JJW+99] Sankar Jayaram, Uma Jayaram, Yong Wang, Hrishikesh Tirumali, Kevin Lyons, and Peter Hart. Vade: A virtual assembly design environment. *IEEE Comput. Graph. Appl.*, 19(6):44–50, 1999.

[Jun03] B. Jung. Task-Level Assembly Modeling in Virtual Environments. In *Computational Science and Its Applications - ICCSA 2003, International Conference, Montreal, Canada, May 18-21, 2003, Proceedings, Part III*, LNCS 2669, pages 721–730. Springer, 2003.

[Kap03] Bill Kapralos. Auditory perception and virtual environments, 2003.

[KF94] Wolfgang Krueger and Bernd Froehlich. The responsive workbench. *IEEE Comput. Graph. Appl.*, 14(3):12–15, 1994.

[KGH85] Myron W. Krueger, Thomas Gionfriddo, and Katrin Hinrichsen. Videoplace - an artificial reality. *SIGCHI Bull.*, 16(4):35–40, 1985.

[KTK+97] K. Kiyokawa, H. Takemura, Y. Katayama, H. Iwasa, and N. Yokoya. Vlego: A simple two-handed modeling environment based on toy block. In *VRST*, pages 27–34, New York, NY, USA, 1997. ACM.

[LTH86] David H. Laidlaw, W. Benjamin Trumbore, and John F. Hughes. Constructive solid geometry for polyhedral objects. *SIGGRAPH Comput. Graph.*, 20(4):161–170, 1986.

[MA98] Jennifer Mankoff and Gregory D. Abowd. Cirrin: a word-level unistroke keyboard for pen input. In *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 213–214, New York, NY, USA, 1998. ACM.

[MB92] I. Scott MacKenzie and William Buxton. Extending fitts' law to two-dimensional tasks. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 219–226, New York, NY, USA, 1992. ACM.

[McT02] Michael F. McTear. Spoken dialogue technology: enabling the conversational user interface. *ACM Comput. Surv.*, 34(1):90–169, 2002.

[MFPBS97] Mark R. Mine, Jr. Frederick P. Brooks, and Carlo H. Sequin. Moving objects in space: exploiting proprioception in virtual-environment interaction. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 19–26, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[MH99] Tomas Möller and Eric Haines. *Real-time rendering.* A. K. Peters, Ltd., Natick, MA, USA, 1999.

[OS03] Ji-Young Oh and Wolfgang Stürzlinger. Intelligent manipulation techniques for conceptual 3d design. In *INTERACT*, 2003.

[PBWI96] Ivan Poupyrev, Mark Billinghurst, Suzanne Weghorst, and Tadao Ichikawa. The go-go interaction technique: non-linear mapping for direct manipulation in vr. In *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 79–80, New York, NY, USA, 1996. ACM.

[Per98] Ken Perlin. Quikwriting: continuous stylus-based text entry. In *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 215–216, New York, NY, USA, 1998. ACM.

[PIWB98] I. Poupyrev, T. Ichikawa, S. Weghorst, and M. Billinghurst. Egocentric object manipulation in virtual environments: Empirical evaluation of interaction techniques. *Computer Graphics Forum*, 17(3):41–52, 1998.

[PST+96] Randy Pausch, Jon Snoddy, Robert Taylor, Scott Watson, and Eric Haseltine. Disney's aladdin: first steps toward storytelling in virtual reality. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 193–203, New York, NY, USA, 1996. ACM.

[RH94] John Rohlf and James Helman. Iris performer: a high performance multiprocessing toolkit for real-time 3d graphics. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 381–394, New York, NY, USA, 1994. ACM.

[Rob65] LG Roberst. *Homogeneous matrix representation and manipulation of n-dimensional constructs*. Lincoln Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 1965.

[SC02] William R. Sherman and Alan B. Craig. *Understanding Virtual Reality: Interface, Application, and Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[SCP95] Richard Stoakley, Matthew J. Conway, and Randy Pausch. Virtual reality on a wim: interactive worlds in miniature. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 265–272, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[SG02] Andreas Simon and Martin Göbel. The i-cone - a panoramic display system for virtual environments. In *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, page 3, Washington, DC, USA, 2002. IEEE Computer Society.

[Sut65] Ivan E. Sutherland. The ultimate display. In *Proceedings of IFIP*, pages 506–508, 1965.

[Szy02] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[Tra99] Henrik Tramberend. Avocado: A distributed virtual reality framework. In *VR '99: Proceedings of the IEEE Virtual Reality*, page 14, Washington, DC, USA, 1999. IEEE Computer Society.

[Tra03] Henrik Tramberend. *Avocado: A Distributed Virtual Environment Framework*. PhD thesis, Universität Bielefeld, Technische Fakultät, 2003.

[VN94] Dan Venolia and Forrest Neiberg. T-cube: a fast, self-disclosing pen-based alphabet. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 265–270, New York, NY, USA, 1994. ACM.

[WAB93] Colin Ware, Kevin Arthur, and Kellogg S. Booth. Fish tank virtual reality. In *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, pages 37–42, New York, NY, USA, 1993. ACM.

[WF02] Greg Welch and Eric Foxlin. Motion tracking: No silver bullet, but a respectable arsenal. *IEEE Comput. Graph. Appl.*, 22(6):24–38, 2002.

[Zac00] Gabriel Zachmann. *Virtual Reality in Assembly Simulation*. PhD thesis, Technische Universität Darmstadt, Fachbereich Informatik, 2000.

[Zha98] Shumin Zhai. User performance in relation to 3d input device design. *SIGGRAPH Comput. Graph.*, 32(4):50–54, 1998.

[ZR01] Gabriel Zachmann and Alexander Rettig. Natural and robust interaction in virtual assembly simulation. In *Eighth ISPE International Conference on Concurrent Engineering: Research and Applications (ISPE/CE2001)*, West Coast Anaheim Hotel, California, USA, July 2001.

A Geometric Transformations

Geometric transformations can be applied to a geometric object in order to transform it in various ways. They are used in a variety of applications, like architecture, robotics or computer animation. E.g. geometric transformations can be used by an animator to move the virtual camera or the objects in a scene along a specific path. They are also often used in a hierarchical description of complex objects, which are composed of several parts, whereas each of these parts can be recursively made out of other parts. E.g. a car could be modelled as a chassis, a motor, a driving seat and wheels. Each of these parts can be described itself as a hierarchy of smaller components. Thus, the overall model can be described as a collection of these components with associated geometric transformations.

The basis of this short introduction of geometric transformations is based on [DH04, FvDFH90]. These books are also a good starting point for further reading about Computer Graphics in general.

A.1 Matrix Fundamentals

A matrix is a rectangular array of values, whereas the values are called elements of the matrix. A matrix can be identified by the number of rows, r, and the number of columns, c. In general, the matrix, M, can be written as:

$$M = \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1c} \\ m_{21} & m_{22} & \cdots & m_{2c} \\ \vdots & \vdots & & \vdots \\ m_{r1} & m_{r2} & \cdots & m_{rc} \end{bmatrix}, \tag{A.1}$$

where $m_{s,t}$ $s \in [1, r]$, $t \in [1, c]$ refers to the elements of matrix M. The first subscript denotes the row number and the second subscript the column number.

A matrix containing a single row or column describes a vector. The standard mathematical convention for applying matrix operations on a vector is to represent it as a column matrix:

$$V = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}. \tag{A.2}$$

Points and vectors often share the same representation, but it is important to distinguish between them. The representation of a point, p, always assumes that the vector points from the origin to the point p. Therefore vector operations like the dot- and cross product cannot be applied to points.

Matrix Multiplication

The matrix multiplication operation is a generalized form of the dot product for two vectors. Two matrices $U_{m,n}$ and $V_{s,t}$ can be multiplied, if the number of columns in U is equal to the number of rows in V. If this precondition is fulfilled, the resulting matrix M is calculated by summing up the products of the elements of the row vectors in U with the corresponding elements of the column vector in V. Thus it has m rows and t columns. Each element $e_{j,k}$ of matrix M can be calculated by:

$$e_{j,k} = \sum_{i=1}^{m} u_{ji} v_{ik}, \tag{A.3}$$

where u, v are elements of matrix U respectively V.

General matrix multiplication is not commutative, therefore:

$$UV \neq VU. \tag{A.4}$$

In some case it is useful to exploit the fact, that matrix multiplication is distributive in respect to addition:

$$U\left(V + W\right) = UV + UW. \tag{A.5}$$

Matrix Transpose

The matrix transpose, M^T is obtained by exchanging the rows and columns of matrix M:

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}^T = \begin{bmatrix} a & c & e \\ b & d & f \end{bmatrix}, \quad \begin{bmatrix} g \\ h \\ i \end{bmatrix}^T = \begin{bmatrix} g & h & i \end{bmatrix}. \tag{A.6}$$

The product of two transposed matrices satisfies:

$$B^T A^T = \left(AB\right)^T. \tag{A.7}$$

Matrix Determinant

For every nxn square matrix A a function exists, that calculates a single scalar out of the matrix elements called the *determinant, det*(A). This value can be useful for solving a range of problems, like *Eigenvalue* calculation or solving a set of linear equations. The second order determinant of a matrix is defined by:

$$det \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc. \tag{A.8}$$

Higher order determinants must be calculated recursively out of lower order determinants. For calculating a determinant d of order two or greater out of a matrix A, any column k can be selected as a starting point to calculate the determinant recursively using:

$$det(A) \quad = \quad \sum_{i=1}^{n} m_{i,k} C_{ik}, \tag{A.9}$$

whereas C_{ik} is the cofactor defined by:

$$C_{i,k} = (-1)^{i+k} M_{ik}, \tag{A.10}$$

and M_{jk} denotes the *Minor* (determinant of the reduced submatrix) of matrix A formed by deleting row i and column k. Calculating determinants for large matrices is a costly operation and can be accomplished more efficiently by using numerical methods than the analytical solution.

Matrix Inverse

A nxn square matrix A has an inverse matrix denoted A^{-1} if and only if the determinant $det(A)$ is nonzero. If this inverse matrix exists, the A is called a *nonsingular* matrix, otherwise the matrix is referred as a *singular* matrix. The fundamental property of a invertible matrix is:

$$AA^{-1} = A^{-1}A = I, \tag{A.11}$$

where I denotes the nxn identity matrix.

The inverse matrix A^{-1} can be calculated analytically from the elements of matrix A using:

$$A^{-1} = \frac{1}{det(A)} C_{ki}. \tag{A.12}$$

In this context C_{ki} refers to the cofactor (see equation A.10). The analytic solution is efficient only for relatively small matrices. When inverting larger matrices, numerical methods like *LU Decomposition* are used for calculating the determinants of the elements of the inverse matrix.

A.2 Homogeneous Coordinates

Homogeneous Coordinates were first introduced by Roberts [Rob65] in the area of computer graphics. They are no Euclidean coordinates, but they are coordinates of another kind of geometry, the so-called *Projective Geometry*. Since only one aspect of homogeneous coordinates are of interest in respect to this theses, the interested reader is referred to [Cox03] for a detailed description.

The main benefit of homogeneous coordinates in computer graphics applications is their ability to perform certain operations on points in euclidean space solely using matrix multiplications. Cartesian coordinates for a point consists of n coordinates, where n is the dimensionality of the space. The homogeneous coordinate of the same point has $n+1$ coordinates. The transformation of a cartesian coordinate into a homogeneous coordinate is straightforward. A new coordinate is appended at the end of the coordinate list. Thus a three-dimensional coordinate $p_c = (x, y, z)$ becomes $p_h = (x, y, z, w)$, where $w = 1$. Note that the overall scaling of homogeneous coordinates is unimportant, since the point $p = (x, y, z, w)$ is equal to $q = (\alpha x, \alpha y, \alpha z, \alpha w)$ for $\forall \alpha \neq 0$.

The transformation of a homogeneous coordinate into a cartesian coordinate is done by:

$$p_h(x, y, z, w) \rightarrow p_c(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}). \tag{A.13}$$

A.3 Three-Dimensional Transformations

This section describes three-dimensional rigid body transformations on the basis of the matrix definitions (see section A.1) and the use of homogeneous coordinates (see section A.2). This allows that all rigid body transformations can be combined into a single transformation matrix.

Translation

A three-dimensional point $P = (x, y, z)$ can be translated to a point $P' = (x', y', z')$ by adding the translation distances t_x, t_y and t_z to the coordinates of P.

Using homogeneous coordinates the translation operator T can be expressed in matrix representation as:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
$$P' = T \cdot P. \tag{A.14}$$

Axis aligned rotations

An object can be rotated in three-dimensional space around any axis, however the simplest cases are rotations where the rotational axis coincide with one of the axis of the cartesian coordinate system.

By convention a positive rotation angle is corresponding to a counter clockwise rotation. A three-dimensional rotation around the z-axis is defined as:

$$\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \\ z' &= z \end{aligned} \tag{A.15}$$

where θ is the angle of the rotation. To calculate rotations around the x- and y-axis, a cylindrical permutation of the parameters x, y and z can be used:

$$x \to y \to z \to x. \tag{A.16}$$

In order to obtain the rotation transformation around the x-axis, x, y and z in equation A.15 are cylindrically replaced as described above. Thus a rotation around the x-axis is defined by:

$$y' = y\cos\theta - z\sin\theta$$

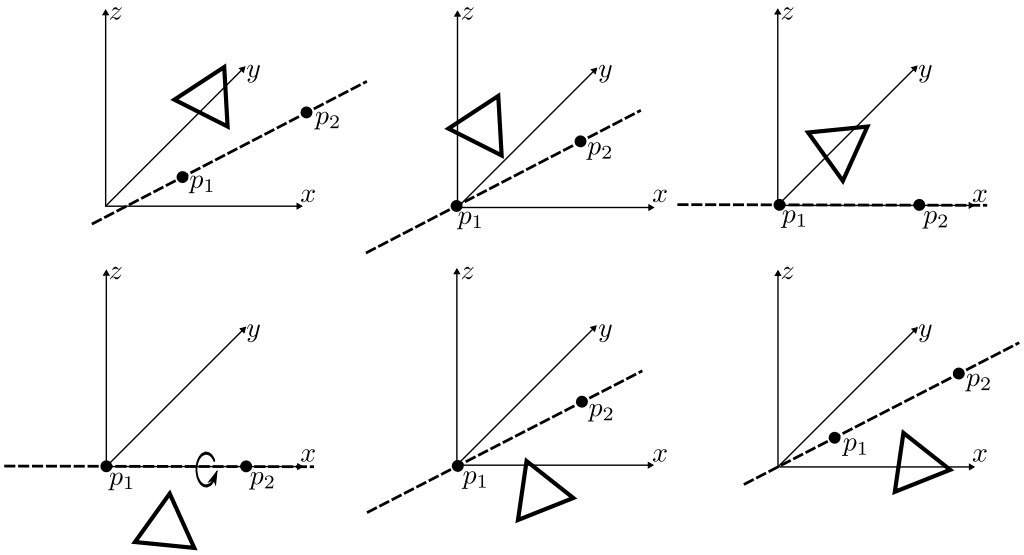


Figure A.1: General three-dimensional rotation around an arbitrary axis in cartesian space.

$$\begin{aligned} z' &= y \sin\theta + z \cos\theta \\ x' &= x. \end{aligned} \tag{A.17}$$

All rotations around the coordinate axis can be easily transformed into matrices. E.g. the rotation of a point P around the x-axis in matrix form is defined by:

$$\begin{aligned} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \\ P' &= R_x(\theta) \cdot P. \end{aligned} \tag{A.18}$$

Rotations around the y- and z-axis can be transformed to matrix form in the analog way.

The inverse rotation is obtained by the negation of the value of θ. Due to the fact the cosine function is not affected for changes of the sign, the inverse rotation matrix can be obtained by interchanging the rows and columns of the rotation matrix. Thus, the inverse of every rotation matrix is its transposed ($R^{-1} = R^T$).

General rotations

In many cases a rotation around an arbitrary rotation axis, not coinciding with one of the coordinate axis, is needed. This rotation is a composite transformation consisting of various translations and rotations around the coordinate axis. The general concept of a rotation around an arbitrary axis can be divided into three individual steps. First

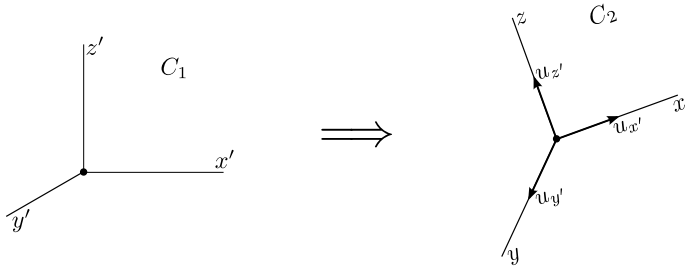


Figure A.2: Transformation of coordinate system C_1 into coordinate system C_2.

the rotation axis if moved onto one of the coordinate axis, followed by the execution of the rotation. After these two steps the rotation axis will be brought back to its original position.

In the special case where the rotation axis is parallel to one of the coordinate axis, the needed steps are straightforward:

1. Translate the object in a way, that the rotation axis coincide with a coordinate axis
2. Perform the rotation around this coordinate axis
3. Apply the inverse translation of step one, in order to move the rotation axis back to its original position

For the general case, where the rotation axis is not parallel to any of the coordinate axis, the following steps are required:

1. Translate the object in a way, that the rotation axis passes the origin of the coordinate system
2. Rotate the object so that the rotation axis coincide with a coordinate axis
3. Perform the rotation around this coordinate axis
4. Apply the inverse rotation of step two in order to bring the object back to its original orientation
5. Apply the inverse translation of step one. This will bring the object back to its original position

Figure A.1 illustrates each of these five steps individually.

A.4 Transformations between Coordinate Systems

Many computer graphics applications need the capability of transforming a coordinate system into another. Since in this thesis only cartesian coordinate systems are of interest,

this section will only deal with transformations between cartesian coordinate systems.

Suppose a system C_1 with the perpendicular axis vectors x', y' and z' is defined in respect to another system C_2 with its axis vectors x, y and z. The first step of the transformation of C_1 into C_2 is the translation of the coordinate origin of C_1 to the origin of C_2. After this translation, a series of rotations are applied to C_1 in order to align the coordinate axis of C_1 and C_2. In some cases, the scaling of the involved coordinate systems are not equal, thus a scaling transformation is used to compensate this. Figure A.2 illustrates this transformation from C_1 into C_2.

B Questionnaire Analysis

B.1 Comments and Observations

This section summarises the nodes taken during the user test procedure and the comments and annotations made by the users after the completion of the questionnaire.

Comments on the immersive modeling system:

- The modeling techniques are natural and easy to use
- Wiimote makes an ideal input device for immersive modeling
- Collaborative modeling is very fast and easy
- The system would benefit from two-handed interaction and rotation techniques
- The shutter glasses are relatively heavy and not well suited for people wearing eyeglasses
- After few trials, the system is easy to use
- Prior to the snapping of a part, a wireframe model of the part should be displayed at the potential snapping position

Comments on the desktop modeling system:

- Rotations are handled strangely by the system and only 90 degrees rotations possible
- Snapping mechanism depends on the actual perspective and is not intuitive
- Annoying parallel projection of the workspace
- Needs a higher cognitive load than the immersive modeling system
- User interface is clearly structured
- The system control with the mouse and the keyboard is not intuitive
- The usage of the system was not intuitive at all

- The system does not use full 3D, but rather 2.5D

Observations:

- The training phase of most users was very short for both the immersive and the desktop system (1-2 training passes).
- Within the immersive modeling system, the assembly sequence varied from user to user.
- Almost all users chose the same assembly sequence with the desktop modeling system.
- The part rotation technique in the desktop system caused problems for some users.
- Some user agreed on a specific division of work in the collaborative assembly tasks and others just started modeling without making any agreements. In the overall process, the task completion times of both methods were roughly equal.

B.2 Questionnaire

User Nr. Name: Date:

Questionnaire

1) Judge how well and intuitive the specified assembly task could be solved

a) with the immersive modeling system

1.66 (0.50)

b) with the dektop modeling system

3.44 (0.72)

2) Which of the two modeling systems do you prefer to solve this task?

1.22 (0.44)

3) For which modeling system did you need a longer training phase?

3.66 (1.00)

4) Which of the modeling systems provided a better perception of the model during and after the modeling process?

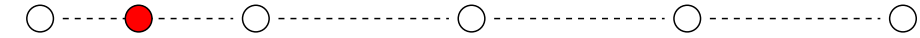
1.44 (0.72)

5) How important was the visual feedback given by the two modeling systems during the actual modeling process for you?

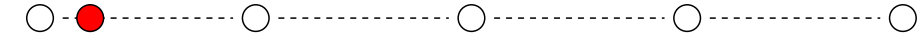
1.44 (0.72)

User Nr. Name: Date:

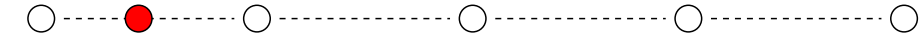
6) Which of the two modeling systems provided a better visual feedback?



7) According to your opinion, how intuitive and natural was the collaborative immersive assembly modeling?

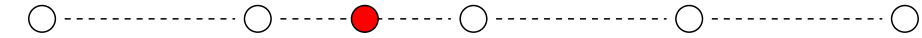


8) Judge if the immersive single user or the collaborative two user assembly was more efficient to solve the given modeling task.

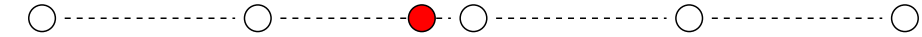


9) Please rate your experience in the following areas:

a) I am using Virtual Reality applications



b) I am using desktop modeling systems



10) Here is plenty of space for your annotations and suggestions. Did you have any problems with the used display devices, the input devices or with the interaction techniques? Do you have any ideas or recommendations? Did you find anything particularly good or bad?