Fachhochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Department of Computer Science

Fraunhofer Institut
Intelligente Analyse- und
Informationssysteme

Bachelor's Thesis

Submitted in Parital Fulfillment of the Requirements
for the Degree of Bachelor of Science in
Computer Science

Gesture Control in Virtual Environments

– Gestenerkennung für Virtuelle Umgebungen –

by
Sven Seele
sven.seele@smail.inf.fh-bonn-rhein-sieg.de

Thesis advisor:	Prof. Dr. Wolfgang Heiden
Second examiner:	Prof. Dr. Marlis von der Hude
External advisor:	Dipl.-Phys. Thorsten Holtkämper

Submission date: 16. September 2008

Acknowledgments

I would like to express my gratitude to those who supported me during the process of writing this thesis. First, I would like to thank Prof. Dr. Marlis von der Hude and Prof. Dr. Wolfgang Heiden for supervising this thesis, but also for their valuable advice and their suggestions.

Special thanks goes to all my colleagues at the Competence Center Virtual Environments at the *Fraunhofer Institute for Intelligent Analysis and Information Systems IAIS*, especially to those involved in the *VRGeo* project. Dipl.-Phys. Thorsten Holtkämper deserves a special mentioning for his everlasting assistance and his adjuvant ideas. Dr. Manfred Bogen and John Plate provided important contributions as well.

I also give credit to Janne Grunau and Benjamin Georgi of the Algorithmics group at the Max Planck Institute for Molecular Genetics, who are involved in the development of the GHMM Library. Their friendly and dedicated email response helped me understand not only the usage of the GHMM library but also the concepts of Hidden Markov Models in general.

I want to thank Kathleen for her encouragement, understanding and patience, despite spending so much time at work or in front of my computer and so little time with her while writing this thesis. I am also very grateful for the love and support which my family has given me throughout my life.

Finally, I thank everybody who was a help to me in any form, particularly Andreas Laude and Ole Tangen for their recommendations and feedback.

Abstract

The introduction of gestures as a supplementary input modality has become of increasing interest to human computer interaction design, especially for 3D computer environments. This thesis describes the concepts and development of a gesture recognition system based on the machine learning technique of Hidden Markov Models. Well-known from the field of speech recognition, this statistical method is employed in this thesis to represent and recognize predefined gestures. Within this work, gestures are defined as symbols, such as simple geometric shapes or Roman letters. They are extracted from a stream of three-dimensional optical tracking data which is resampled, reduced to 2D and quantized to be used as input to discrete Hidden Markov Models. A set of prerecorded training data is used to learn the parameters of the models and recognition is achieved by evaluating the trained models. The devised system was used to augment an existing virtual reality prototype application which serves as a demonstration and development platform for the *VRGeo* consortium. The consortium's goal is to investigate and utilize the benefits of virtual reality technology for the oil and gas industry. An isolated test of the system with seven gestures showed accuracies of up to 98.57% and the review from experts in the fields of virtual reality and geophysics was predominantly positive.

Zusammenfassung

Gesten als ergänzende Eingabemodalität haben mittlerweile einen hohen Stellenwert bei der Entwicklung von Mensch-Computer-Interaktion, vor allem für 3D Computerumgebungen. Diese Arbeit beschreibt die Konzepte und Entwicklung eines Systems zur Gestenerkennung, das auf Hidden Markov Modellen basiert. Dieses statistische Verfahren ist in der Spracherkennung sehr verbreitet und wird in dieser Arbeit für die Repräsentation und Erkennung von vordefinierten Gesten eingesetzt. Dabei werden Gesten als Symbole definiert, z.B. als einfache geometrische Formen oder Buchstaben. Diese werden aus drei-dimensionalen Daten eines optischen Trackingsystems extrahiert und dienen, nach einer Quantisierung, als Eingabe für diskrete Hidden Markov Modelle. Während die Parameter der Modelle von aufgenommenen Trainingsdaten erlernt werden, wird die Erkennung von Gesten durch die Auswertung der trainierten Modelle erreicht. Mit dem entwickelten System wurde eine bestehende Virtual Reality Anwendung erweitert, die dem *VRGeo* Konsortium als Demonstrations- und Entwicklungsplattform dient. Ein Ziel des Konsortiums ist es, die Nutzbarkeit der Virtuellen Realität für die Öl- und Gasindustrie zu erforschen. In einem isolierten Test des Systems mit sieben Gesten wurden Erkennungsraten von bis zu 98.57% erreicht. Gleichzeitig fiel auch die Begutachtung durch Experten aus den Bereichen Virtuelle Realität und Geophysik überwiegend positiv aus.

Contents

Acknowledgments iii

Abstract v

Zusammenfassung v

List of Figures ix

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Goals . . . 2
- 1.3 Structure . . . 3

2 Related Work 5
- 2.1 Symbol Recognition . . . 5
- 2.2 Gesture Recognition . . . 7
- 2.3 Gesture Control . . . 9

3 Virtual Environments 11
- 3.1 Displays . . . 11
  - 3.1.1 Visual Displays . . . 12
  - 3.1.2 Auditory Displays . . . 15
  - 3.1.3 Haptic Displays . . . 16
- 3.2 Input Devices . . . 17
- 3.3 Interaction . . . 19
- 3.4 VR Software Frameworks . . . 22

4 Hidden Markov Models 23
- 4.1 Definition . . . 23
- 4.2 Parameter Estimation . . . 25
  - 4.2.1 Evaluation . . . 26
  - 4.2.2 Decoding . . . 28
  - 4.2.3 Learning . . . 30
- 4.3 Computational Consideration . . . 32

5 Gesture Control in the VRGeo Demonstrator 35
- 5.1 VRGeo Demonstrator . . . 35

5.2 Selection of Tools 38
5.3 AVANGO NG 39
5.4 Preprocessing 41
5.4.1 Resampling 41
5.4.2 Dimensional Reduction 42
5.4.3 Discretization 43
5.5 HMM Initialization 44

6 Training and Results **47**
6.1 Training the Hidden Markov Models 47
6.2 Evaluation of the Recognition System 49

7 Conclusion and Future Work **55**

A Media Content **57**

Bibliography **59**

Declaration of Authenticity **67**

List of Figures

2.1 Graffiti and Unistroke alphabets 6
2.2 Token look-up for a point on the outline of a symbol 6
2.3 Prototype curves and fuzzy states 7
2.4 The SoapBox sensor device 10

3.1 Monoscopic depth cues 12
3.2 Motion parallax 13
3.3 Three examples of visual VR displays 14
3.4 The *TwoView* and *i-Cone* displays 15
3.5 Two common haptic displays 17
3.6 The *Pinch* glove 18
3.7 Four categories of manipulation in VR applications 21

4.1 Illustration of different HMM topologies. 26
4.2 Computation of forward variables 27

5.1 The *VRGeo* Demonstrator 36
5.2 Example of user feedback 37
5.3 Gesture symbols used in the *VRGeo* Demonstrator. 38
5.4 2D example of thresholding process 41
5.5 Illustration of the dimensional reduction 43
5.6 Vector quantization 44
5.7 HMM prototype 45

6.1 Classification results 50
6.2 Symbols used to test the noise HMM 51
6.3 Misclassification results 52

1 Introduction

Software usually offers a variety of functionality to a user via graphical menus. An expert regularly using such software wishes to minimize efforts of executing these functions in order to create a more efficient and fluent workflow [55, p. 68]. In desktop environments it is a common and widely accepted solution to use shortcuts, e.g. specific key combinations, to create a direct call to frequently used functions, superseding time-consuming navigation through menu structures. When designing interfaces for virtual environments (VEs), it is often impossible to use conventional input devices such as mouse and keyboard. This makes it much more difficult to introduce concepts that maximize workflow efficiencies.

Another important aspect in VEs is to create a natural user interface, meaning that a user communicates with a system in a way as similar as possible to interaction in real life. The most natural interface would allow a user to utilize the entire body as well as multimodal interaction in a three-dimensional environment without using any additional devices. Due to technical difficulties, imprecisions and other factors, this goal still remains far from being realized.

1.1 Motivation

Virtual reality (VR) applications are almost exclusively designed for a certain area of work. Functionality is usually restricted and in most cases a user is an expert in the particular field of interest. Input devices are usually highly dependent on the type of application. However, just as in the case of desktop applications, a user will want to avoid navigation through menu structures if it slows down the progress.

The work presented in this thesis was conducted in the context of the *VRGeo*[1] project at the *Fraunhofer Institute for Intelligent Analysis and Information Systems IAIS*. The intention of the project is to support the *VRGeo* Consortium. Members of the consortium represent companies from the oil and gas industry as well as their software and hardware suppliers. The consortium meets twice a year to discuss the value of virtual reality technology to applications for the geosciences. Part of the project is to develop a software prototype, the *VRGeo* Demonstrator. This VR application enables a user to visualize and explore seismic data in virtual environments. [31]

One research topic of the consortium was to include gesture control to the interaction methods used so far, to enhance the existing *VRGeo* Demonstrator. With the new feature it should be possible to access frequently used functions more quickly.

[1] VRGeo: *Virtual Reality research for the Geosciences*

The request to employ gesture control as a supplement to other existing input modalities is not without reason. This concept was successfully integrated in many other application areas in VR. Examples can be found in [11, pp. 270–273]. Gestures are a very natural form of interaction and communication to people. We use it in everyday life, for example for co-articulation in speech, to interact with the environment or to signal emotions, certain expressions or intentions. Hence, gestures have become of increasing interest in the field of human computer interaction (HCI) design since its goal is to make HCIs more natural and intuitive. Additionally, natural interfaces increase the sense of presence which is one of the main goals of virtual environment design. The sense of presence is the user's feeling of really being in the virtual environment as opposed to the actual location. Therefore, gestures could be a general solution to achieve task efficiency and improve the sense of presence. [14, 60, 67]

1.2 Goals

This thesis traces the development of a gesture recognition back-end that was to be included in the existing virtual reality application prototype of the *VRGeo* project. The design of the recognition system was based on the experiences and requirements gathered during previous attempts of introducing gesture control to the prototype. Furthermore, it was important to use existing data and input devices for the purpose of recognition and control to avoid the introduction of additional devices or constraints.

With the present setup, gesture input could have been introduced to the mentioned virtual environment prototype for several possible scenarios. Application control such as object manipulation or mode changes, device management and navigation are common scenarios for gesture control that would have been plausible. The focus of this thesis lies on gesturing and recognizing predefined symbols to manipulate objects within the prototype.

For this purpose the statistical concept of Hidden Markov Models is deployed. Well-known from speech recognition applications, they have gained a strong foothold in the field of gesture recognition as well. Many publications on gesture recognition systems like [42, 58, 57, 63, 66, 89, 91] make use of this powerful method achieving high recognition rates.

The purpose of this thesis is to provide the theoretical background on Hidden Markov Models and a proof of concept: The result should be a functional module integrated into the VR prototype of the *VRGeo* project. It should recognize a set of predefined symbolic gestures which can be mapped to certain tasks. The identification of the most frequently performed tasks and the selection of corresponding gestures is made rather intuitively, because the important goal is to have a working system. Sophisticated tools such as user studies are not employed.

The positive response at the *VRGeo* Consortium meeting in June 2008 determined that the concept would be kept as part of the prototype. Keeping the gesture recognition system allows further development based on the given feedback. Therefore, it was important to build a system that would be easily extendable to ensure

successive research. Another difficulty was the presentation of the system. The representatives of the consortium should be able to test the system themselves. Thus, the system had to be user-independent.

1.3 Structure

Chapter 2 will present the research related to this thesis. This will show the origin of most of the ideas and concepts used for the work described here. It also contains some approaches which could be interesting for future work.

In Chapters 3 and 4 the basics of virtual environments and Hidden Markov Models are discussed to provide a context for the following chapters. Chapter 3 also includes a presentation of common input and output devices for virtual environments. Chapter 4 will deal with the theoretical foundations of Hidden Markov Models. After giving a definition the three fundamental problems of evaluation, decoding and training are examined before mentioning some computational issues.

Chapter 5 demonstrates how the ideas and theories described in the previous chapters are put into practice. At the beginning of the chapter, the *VRGeo* Demonstrator is introduced since it is the application the recognition system was tested in. Considerations, difficulties and the individual processing steps are explained in detail starting with an examination of the choice of utilized computational libraries followed by the basic concepts of the AVANGO® NG framework [47] which was used for the implementation. A section on preprocessing gives insight into how collected data is prepared for the Hidden Markov Models. The end of this chapter will present details on the choices made for creating the Hidden Markov Models used for training and recognition.

Chapter 6 illustrates the training process and experimental results of the devised system. Especially the impact of the definitions from Chapter 5 are discussed based on the obtained results.

Chapter 7 summarizes the concepts developed and implemented in this thesis, discusses whether the goals of this thesis were achieved and introduces ideas for future work.

2 Related Work

As mentioned before, gestures are a natural supplement to human communication and interaction. Hence, many attempts have been made to utilize gesture control in general or specifically for virtual environments. Those trials include vision-based approaches [58, 78, 83, 89] or specific device input [42, 63, 91] to collect either two- or three-dimensional data which is analyzed using template-matching [53], dictionary look-up [76], statistical matching [72], neural networks [28] and others. Suggested applications span from human/robot interfaces [50] to weather narration [68] to musical score editing [13]. Presenting all of the important contributions is far beyond the scope of this thesis. Instead of giving a broad overview of the work performed in this field, this chapter will concentrate on a few selected publications and systems which relate closely to the project presented within this work.

2.1 Symbol Recognition

Depending on point of view and context, gestures can be defined very differently. For simplicity the work presented in this thesis concentrates on the recognition of gestures imitating certain symbols or characters. Obviously, this issue is closely related to optical character recognition (OCR) in general. Research in the field of OCR dates back to first experiments by Gustav Tauscheck in 1929 [61], though the goal of enabling machines to read characters or numerals was not accomplished until the 1950's. Since the history of OCR research is relatively old, there have been numerous methods, experiments, publications and implementations. One of them is the well-known Graffiti software by Palm, Inc. deployed in PDAs. The advantage of this software was that it avoided the biggest problems that other handwriting recognition system have by taking a much simpler approach. Instead of trying to recognize whole words or sentences, input was restricted to one symbol at a time. This made the system faster and similar to keyboard input which people were used to. However, more important for its success was the introduction of a new alphabet based on single strokes. A similar idea was presented by David Goldberg [33] who also designed an unistroke alphabet before the development of Graffiti. Although the idea was not new, the alphabet used for Graffiti was designed to closely resemble Roman letters which helped users get accustomed to this technique rather quickly. As for the recognition process, the idea of unistrokes and different input modes helped to minimize classification errors when dealing with individual and sloppy handwriting. [8, 54]

Elliman and Connor [26] extended the OCR idea to feature a sort of learning algorithm. The goal was to recognize a symbol regardless of its font, orientation

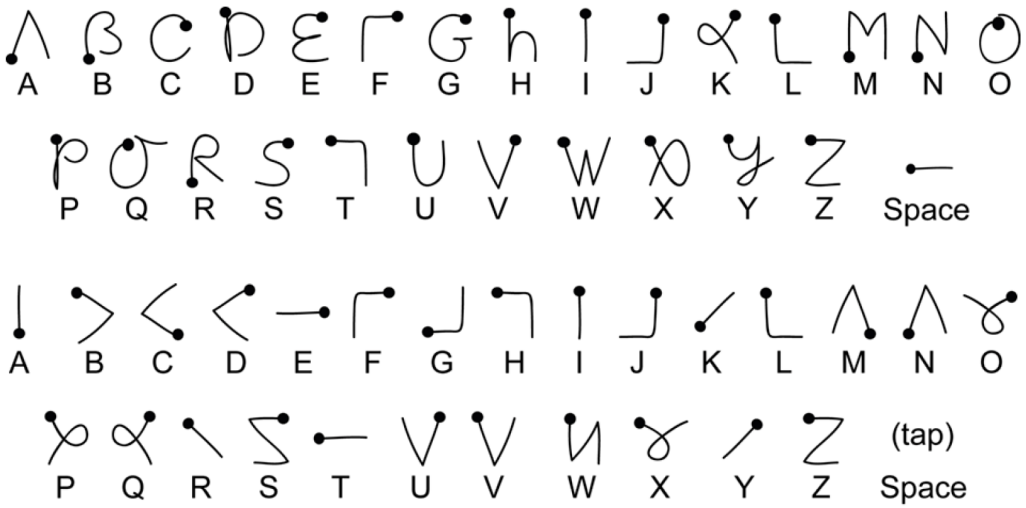
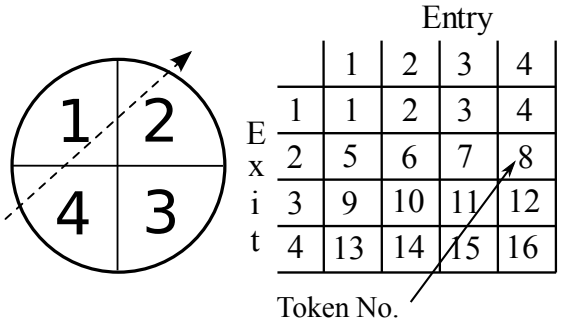


Figure 2.1: Graffiti (top) and Unistroke (bottom) alphabets. (Illustration from [16])

and size. They encoded the outline of a symbol as a sequence of discrete tokens. This is done because discrete Hidden Markov Models are used for the learning and recognition steps. After extracting the outline of a symbol, the centroid is calculated and is set as the origin. Further the symbol is scaled so that the outline is of a certain standard length. To obtain the token sequence, a bounding sphere is put around the current symbol which is divided into four quarters. Starting at the point of the symbol furthest from the centroid, a tangent direction at each equidistant point on the outline is calculated in clockwise order. With the help of the quartered circle and a look-up table (LUT), the tokens are derived as shown in Figure 2.2. Using 16 symbols with 3 samples of 25 different postscript fonts and a twenty-state Hidden Markov Model, the recognition rate was 97% for symbols from the trained fonts. When using symbols from untrained fonts, the rate was still around 70%.



Exit \ Entry	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

Figure 2.2: Token look-up for a point on the outline of a symbol. (Illustration from [26])

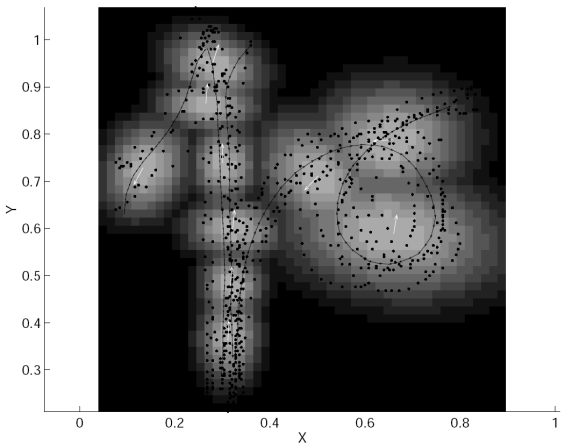


Figure 2.3: Combination of two prototype curves (black) of two different gestures with fuzzy states centered at about the white vectors. (Image from [88])

2.2 Gesture Recognition

Much work has been done dealing solely with the issue of recognizing gestures. For example Wilson and Bobick [88] introduced *configuration states* to represent and recognize gestures. Using a sequence of n such states they define a *prototype gesture*, $G_\alpha = \langle \alpha_1 \alpha_2 \dots \alpha_n \rangle$. For recognition, a set of trajectory sample points $x_1, x_2, \dots, x_N$ is compared against the prototype of a certain gesture. Since a gesture will unlikely be repeated with the exact same state sequence, the states need to incorporate "fuzziness" to account for the variance of position. To solve this problem, a point x in configuration space[2] is assigned to a fuzzy state s_i using a *fuzzy membership function* $\mu_{s_i}(x) \in [0, 1]$. However, besides the variance in position, a repetition of the same gesture can also vary in time which should not alter the result of the recognition process. Wilson and Bobick deal with this by computing a prototype curve as seen in Figure 2.3. It is parametrized by arc length while at the same time preserving the correct order of states. Their concept is proven by successfully applying their method to three different types of sensory data: two-dimensional mouse gestures, hand movements measured by a magnetic spatial position and orientation sensor, as well as digitized image sequences of hand movements.

Yang and Xu [91] chose a different approach. For dealing with the variability of human gestures, they employed Hidden Markov Models. The interesting part of the work done by Yang and Xu is the conversion of gestures into a sequence of discrete symbols. The short-time Fourier transformation (STFT) is used to block the original signal into frames. From the amplitude of the FFT coefficients, a set of feature vectors is constructed which is then discretized using the *LBG algorithm* [52] for vector quantization. The indices from the code book represent the input symbols for two-dimensional Hidden Markov Models. Experiments showed a recognition rate of

[2]configuration space : space of measurements that define each point of an example gesture

up to 98.78% for an isolated recognition task with nine gestures performed with a mouse input device.

In an effort to give users the opportunity to interact without any additional devices, Stark and Kohler [78] presented ZYKLOP, a video-based gesture recognition system for human computer interaction. While handling classic computer vision issues such as image segmentation, their system produces knowledge bases containing information of gestures via a learning step. This step includes the construction of reference vectors from the segmented data of several gesture samples. One part of this process is to calculate the center of mass and ensuring rotational invariance. The recognition is done through correlation of the feature vector and the stored reference vector, or through decision surfaces. For the latter, each feature vector is interpreted as point in d-dimensional space. The space is separated by hyperplanes into regions which correspond to one of k gestures. In this case, the learning step is used to determine the coefficients of the hyperplanes. With this approach, the system is able to recognize single frame hand gestures which are defined as hand postures with an associated meaning. To be able to recognize a sequence of gestures, the system implements the concept of a finite automaton. Every automaton is assigned a validity value and a gesture sequence. After processing the sequence input, the automaton with the highest validity that is in a final stage represents the recognized gesture sequence. Additionally, the system is also capable of recognizing motions by evaluating a sequence of centroids. For all centroids in a sequence, a center of mass, distance of each vertex to the center and the connection angle for two succeeding points are calculated. The resulting feature vectors can again be compared to reference vectors. The ZYKLOP system was used in two different applications (a presentation system and *Geomview*[3]) with eight different gestures taken from 14 different persons achieving recognition rates of up to 93.44%. Stark and Kohler also identified the importance of providing feedback to the user. While running the ZYKLOP interface, a window permanently displays the image recorded by the camera. A colored frame indicates whether a gesture has been recognized or not.

Eisenstein et al. [23] stated that most work done in the field of gesture recognition suffers from device dependency. They introduced a layered framework where only the lowest layer is device dependent. Although in their work they showed only independence of different glove input devices. By transforming the device-dependent raw input data to *postural predicates*, they were able to make use of simple template matching algorithms instead of using more complicated techniques such as neural networks.

In coherence with his attempt of providing a mathematically uniform presentation of the theory of Hidden Markov Models, Mäntylä presented a gesture recognition system based on the idea of unistroke alphabets [57]. The gestures were to be performed with a pen-like accelerometer system around a fixed center of action. Preprocessing and feature extraction were performed using a MATLAB environment. The *k-means*

[3]Geomview is a software to present and manipulate geometric data, developed by the Geometry Center at the University of Minnesota.

algorithm was applied to quantize the resulting feature vectors of 90 training samples for each gesture. The quantized data was fed into Hidden Markov Models of a strict left-right topology. His results suggest that the quality of recognition generally increases proportional with the number of training samples. He also found that the more complex a gesture, the more states should be used in the corresponding Hidden Markov Model.

2.3 Gesture Control

While the before mentioned approaches set the goal predominantly on recognizing gestures, all of them integrated experiments to use their algorithms within a certain application. The following section introduces gestures as a specific means to interact within a virtual environment. In 1996 Pavlović et al. [66] presented a gestural interface for a system called MDScope stating that the original mouse and magnetically tracked pointer interface hindered a natural interaction. The interface included a simple image-based algorithm to use the index finger as 3D pointing device. To manipulate objects a set of static and dynamic hand gestures was defined. Extracted features were then used to train four-state Hidden Markov Models with 35 training sequences yielding 80% of correct recognitions.

O'Hagan et al. [64] introduced a vision-based gesture recognition framework for a virtual working environment. An underlying manually acquired model of the hand simplifies feature extraction and aids the tracking process after segmentation. Five gestures were chosen to correspond to as many tasks common for virtual environments: selection, object/scene translation, object/scene rotation, object resizing and scene zoom. To classify the gestures, the statistical approach of building a set of logistic regression models was applied. For each gesture one model is constructed as a linear function of predictor variables X_i such that

$$g(x_1, \ldots, x_p) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p, \tag{2.1}$$

where $(x_1, \ldots, x_p)$ is a feature vector of predictor variables and $\beta_0, \ldots, \beta_p$ are the model parameters. The model parameters are estimated through a set of feature vectors from a training set of 900 images per gesture. The model equations are used during the classification process to calculate the probability of the extracted vector elements belonging to one of the models. The equation producing the highest probability represents the recognized gesture. If the probability value lies beneath a certain threshold, the image is classified as unrecognized. Even with the difficulties of feature extraction and segmentation in a vision-based system, this technique achieved a recognition rate of 92.60% while maintaining a frame rate of 30 Hz.

For the VE application *Smart Design Studio*, developed at the Italdesign-Giugiaro Virtual Reality Centre, Kela et al. [42] presented a study on the usage of gestures for controlling this application. They found that the "right" gesture for a certain task depends on the individual, making it important to give the user the opportunity to personalize the set of control gestures. Their study also showed that the modality

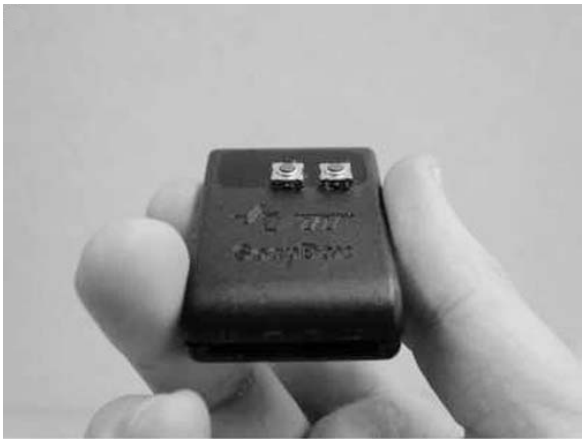


Figure 2.4: The SoapBox sensor device. (Image from [42])

preferred by a user depended heavily on the task performed or the educational background of a user. Those results indicate the advantages of having a multimodal system to give a user the choice which input method to use for which task. To input gestures a special device called SoapBox (*Sensing, Operating and Activating Peripheral Box*) was used. Figure 2.4 shows this matchbox-sized device which includes a three-axis acceleration sensor, an optical proximity sensor and an optional temperature sensor [82]. The three-dimensional acceleration data was preprocessed to first train discrete Hidden Markov Models and later to recognize the issued gesture. To minimize the number of training vectors, noise was added to copies of the original gesture data [41]. With this technique, a recognition rate of 96.40% was achieved for only 2 training vectors if the gestures to be recognized were performed by the same user whose data was used to train the Hidden Markov Models.

A similar approach was tested in connection with the *VRGeo* project at the *Fraunhofer IAIS*. The accelerometer of Nintendo's *Wii Remote* was used to record three-dimensional trajectories of predefined gestures. After discretizing the data it was used to train a nine-state Hidden Markov Model. Unfortunately, this attempt did not yield the desired results leading to the work presented in this thesis.

3 Virtual Environments

A virtual environment is a computer generated environment aimed to provide a user with the experience of being in a simulated place. This is usually achieved by blocking stimuli from the "real" world and replacing them with synthetic ones. Sherman and Craig [73] identify *immersion, sensory feedback* and *interactivity* as the key elements of perceiving a virtual environment. Immersion is closely related to the deployed display technology (see Section 3.1). It can be described as the feeling of being part of the virtual environment or becoming unaware of the physical surroundings. Sensory feedback allows a presentation based on a user's position and depends on input devices as well as output hardware. Interactivity refers to the ability of a user to interact in some way with the artificial surroundings or the objects in this virtual world. Typical examples would be navigating through the environment, moving objects or initializing simulations. All of those elements eventually have an effect on the presence, the sense of actually being in the virtual environment [84, pp. 4–5]. Obviously, fidelity of sensory feedback and the level of interactivity can increase the feeling of immersion and the sense of presence. On the other hand, inaccurate, cumbersome or unfamiliar devices and displays can also distract a user from the environment which influences immersion and presence negatively.

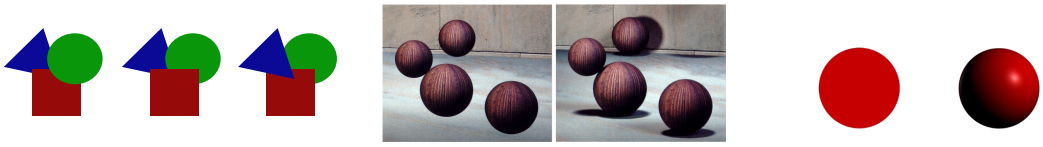
The term virtual reality (VR) is commonly used synonymously to virtual environments. However the definition of VR differs since it is relatively new. It can mean the field of study which deals with creating and improving the synthetic experience described earlier, but also the technology or medium used to create this experience or even the experience itself. [14][43, pp. 3–7]

At first, this chapter introduces the basic components of virtual environments contributing to the key elements mentioned above. Afterwards, virtual reality software frameworks are discussed, because they are extensively used to design virtual environments and the interaction within it.

3.1 Displays

When dealing with virtual environments, the term display is not constrained to the conventional meaning of *visual* display. Although visual displays are used most extensively in VR, it refers to any sensory output used to simulate stimuli. The goal is, of course, to synthetically stimulate as many of the human senses as possible. As a result various kinds of output devices exist. The most common devices simulate visual, auditory and haptic stimuli. Some applications also make use of the olfactory[4]

[4]The sense of smell.



(a) Occlusion (b) Shadow casting (c) Illumination

Figure 3.1: Several monoscopic depth cues that can be achieved using still images. (a) Through occlusion it is depicted which object is further away from a user. (Illustration from [21]) (b) Shadows can be used to show spatial relationships between objects. (Illustration from [6]) (c) Proper illumination can also depict depth information about an object.

sense. Stimulating all senses simultaneously is often too expensive or too unsatisfactory because of imperfect technology or in most cases not necessary. For instance, the displays used for this thesis appeal mostly to the visual sense. Other senses are disregarded for the most part. Therefore, this section will emphasize visual output devices and briefly mention auditory and haptic displays. A comprehensive description of output devices can be found in [12] and [73].

3.1.1 Visual Displays

Virtual environments usually simulate three-dimensional scenes. To convey the three-dimensional structure to a user, the depth information of the scene has to be visualized. Several techniques exist to provide depth cues out of static images but also out of image sequences. Well-known techniques include occlusion, shadow casting, illumination or motion parallax (illustrated in Figure 3.1 and Figure 3.2). A very powerful method, however, is to use *stereopsis*. The human vision is based on an effect called *binocular disparity*. Each eye receives an image which has a slight offset to the other due to the position of the eyes. Fusing both images results in a mental image that contains very strong depth cues called *stereopsis* which is especially effective within a range of approximately 5 meters. Stereoscopic VE displays reproduce this effect by generating separate images for each eye. [11, pp. 34–40][73, pp. 116–121]

One of the most well-known display systems to achieve this effect is the *head-mounted display* (HMD). HMDs are the most common of the so called head-coupled devices which are attached to a user's head. They usually include two separate screens mounted right in front of each eye displaying the according picture. In most cases this type of visual display totally occludes the outside world. Alternatively, the screens can be replaced with semi-transparent mirrors onto which the scene is projected. This allows users to see the real and the virtual environment simultaneously. The perception of reality is augmented with additional information, hence, this technique is called *augmented reality* (AR).

Another type of visual displays are projection-based, examples are the CAVE [19], the Responsive Workbench [46] or the ImmersaDesk [20]. The advantage of

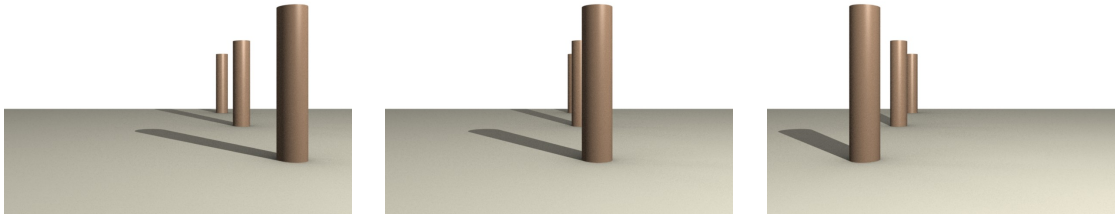
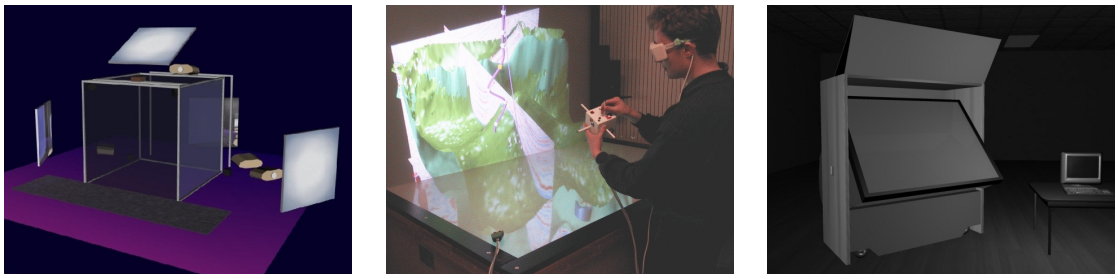


Figure 3.2: This sequence of images conveys depth information using motion parallax. As the view changes the object in front seems to have moved further than the others. (Image from [21])

those systems is that users can move freely in front of the display and that they are able to see their own bodies. The latter is advantageous because the parts of the body that a user needs to see in the virtual environment do not have to be remodeled in 3D. The problem with projection-based systems is that there are not two separate screens to display pictures for both eyes. While monoscopic and motion depth cues can easily be achieved, special techniques have to be applied to realize stereoscopic depth cues. The common techniques are divided into *passive* and *active* stereo vision. When using a passive method, filters are attached to a pair of 3D glasses worn by a user. Older systems often used spectral multiplexing generating the different views in a different color. The glasses would then filter the colors so that only the "right" picture would reach the eye it was designated for. Today *INFITEC* (interference filter technique) [39] is the most advanced spectral multiplexing technique. For this method, triple band filters are placed in front of the projectors to additional filter material deposited onto glasses. The stereoscopic images are displayed in parallel with two primary color triplets of different wave lengths. In this way *INFITEC* systems are capable of visualizing high quality images in full color. While the older technique caused the effect of seeing one reddish and one greenish picture disturbing the experience, *INFITEC* eliminates this problem, increasing the feasibility of this technique for VR applications. It is currently more common to use polarization for image separation. A filter in front of the projectors polarizes the emitted light so that for example the picture for one eye is made up of only horizontal light waves and that of the other eye of vertical. The glasses are built so as to allow only the corresponding polarized light waves to pass through and reach the eye. Since a tilt of the head will noticeably decrease image quality when using linear polarization it is advantageous to polarize the light circularly. However, no matter which one of these techniques is used, two aligned projectors are necessary to create a single image. In the case of polarization, it is also obviously important that the material the scene is projected upon must not change the polarization of the light. [73, p. 124]

Other systems use *shutter glasses* to achieve temporal multiplexing of the visual signal. Since they actively open and close their shutters, this technique is called active stereo. The glasses are synchronized with the refresh rate of a projector or monitor.



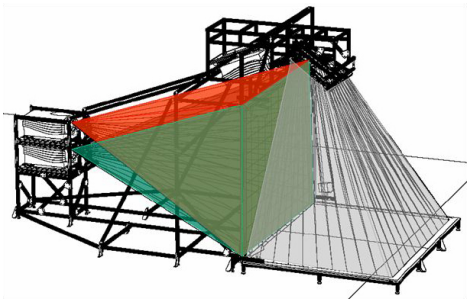
(a) (b) (c)

Figure 3.3: Three examples of visual VR displays. (a) The CAVE Automatic Virtual Environment (Image from [20]) (b) The Responsive Workbench. (c) The ImmersaDesk. (Image from [20])

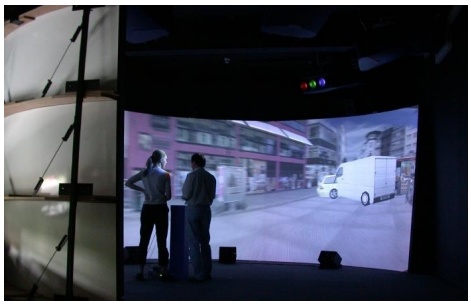
The pictures for the left and the right eye are alternated so that at the time the picture for the left eye is on screen the shutter in front of the right eye is closed and vice versa. To avoid flickering the projection system must perform at high refresh rates. Since two separate pictures are alternated the actual frame rate is cut in half. [11, p. 41]

To ensure the presentation of the virtual world in a correct perspective, position and orientation of the user's eyes have to be tracked (see Section 3.2). Most visual display systems are capable of generating one stereoscopic image at the time. Consequently, only one user can be tracked by the system. This results in a more or less distorted view for every other user depending on the distance to the tracked user. However, there are a few multi-user systems such as the *TwoView* display developed at the *Fraunhofer IAIS* described for example in [71]. The *TwoView* (see Figure 3.4 (a)) is able to display to active stereoscopic image pairs at the same time. Thus, two users can be tracked. The separation of the user dependent views is achieved by circular polarization while the separation of the stereoscopic images is done using shutter glasses. Two active stereo projectors with attached oppositely circularly polarized filters project a picture onto the back of a screen that preserves the polarization. Thus each projector generates a stereoscopic picture for each user. Additionally, the *TwoView* also allows an L-shape configuration for a single user where a second image is projected onto the floor (illustrated by the gray projection in Figure 3.4 (a)). Other multi-user systems use modified shutter glasses and projectors for temporal multiplexing. For example, the *Multi-Viewer Stereo Display* [32] is capable of showing stereoscopic pictures to up to four users.

Systems with a panoramic display intended for a larger audience like the *i-Cone* [74] (see Figure 3.4 (b)) abandon head-tracking entirely. Instead, the view of a fictive person standing in a central position is displayed. All viewers standing far from the center see a slightly distorted image. For objects further away in the scene, this distortion is barely noticeable and can be disregarded. One problem with the fixed viewport is the loss of correct perspective stereo when turning away from the fixed viewing direction. This problem can be solved by using the OmniStereo technique



(a) The TwoView display. The independent user views are indicated by the red and green projection.



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

Figure 3.4: The *TwoView* and *i-Cone* displays developed at the *Fraunhofer Institute for Intelligent Analysis and Information Systems IAIS*.

[75]. The total 240 ° display is split into partial segments and for each segment the viewport is slightly turned to face that segment. The result is an approximately correct perspective in every viewing direction. A similar technique is applied to the projections in the *Immersion Square* [35], a CAVE-like virtual environment based on standard components developed at the Bonn-Rhein-Sieg University of Applied Sciences.

3.1.2 Auditory Displays

Authentic sound adds to the realism of a virtual environment. A visualized scene lacking the sound present in the real world just does not seem credible. Just as visual displays utilize several depth cues to create 3D images; azimuth, elevation and range cues can be used to convey three-dimensional information about sound sources. If the simulated position of the source complements the perceived visual feedback, immersion, interactivity and image quality are increased tremendously. Although highly immersive virtual environments should incorporate 3D sound for the sake of realism, many VR applications omit this feature. In many cases playing ambient sounds and providing strong visual cues[5] to aid localization of sound sources is enough to enhance the sense of presence.

If 3D sound is necessary or desired it can be realized using multiple speakers. Simple forms are the stereo or the well-known *5.1 surround* format (see for example [45]). The three-dimensional sound field can be recorded using separate channels for each speaker. Each channel must then be mapped to the corresponding speaker. When actually rendering audio, i.e. localizing a mono signal in 3D space, three main techniques can be applied. The most widespread method is amplitude differencing. As

[5]The so called *ventriloquism effect* explains how humans localize sound sources based on visual cues. From experiences learned, an assumption is made where the sound is supposed to be coming from. E.g. by hearing somebody talk and seeing someone's mouth move, it is assumed that this person is the origin of the sound. For a more precise definition see [3] or [18].

the name suggests, differences between the signal's amplitude from different speakers are used to suggest sound coming from a certain direction. The two most influential amplitude differencing techniques are *ambisonics* [25] and *vector based amplitude panning* (VBAP) [69]. Both approaches are capable of rendering three-dimensional sound with a variable quantity of speakers. Otherwise it is possible to create *phantom speakers* through convolving the sound signal displayed by only two speakers. This binaural technique developed at NASA makes use of two Fourier transforms called *head-related transfer functions* (HRTFs) [79]. These functions can be measured for each individual and take in account the physical characteristics of a human that influence how sound reaches the ear canal. Some of these characteristics are the structure of the pinnae, distance between both ears or even the shape of the upper torso. Filtering the signal with the HRTFs, in combination with tracking the user's head, the user experiences the sensation of hearing the sound from a virtual speaker at the position of the virtual sound source. The problem with HRTFs is that they are unique for every user and convolving a signal with the HRTF of a different individual results in a worse localization. It is still not possible to devise sound systems that are taylor-made for a user's own HRTF. Modern 3D sound cards use digital signal processing (DSP) chips and HRTF look-up tables to approximate this effect for a certain zone in front of two speakers. The major drawback of amplitude differencing and binaural methods based on a loudspeaker setup is that a user needs to remain in a certain "sweet spot" to experience the sensation of 3D sound. Wave field synthesis (WFS) [80] utilizes an array of many speakers to distribute sound throughout an entire room. Hence, a user is able to move freely about the environment. The sweet spot problem can also be solved using headphones. Since the head-based aural displays are mounted directly in front of each ear the recipient will always remain in the same spot compared to the output device. The use of headphones includes the additional advantage that crosstalk is precluded. The often discomforting feeling of being sealed off from the physical environment by closed-ear headphones can be attenuated with open-ear (hear-through) headphones, a concept similar to that of conventional and see-through HMDs. [12, pp. 84–92][73, pp. 164–177][84, p. 63][90]

Other auditory displays pursue a different goal than creating realistic 3D sound. Sometimes sound can be used as a *sensory substitution* for a sense not displayable in the current environment. For example, pressing a button in the virtual world could play a sound instead of providing an haptic feedback like in reality. Another possibility tor the use of audio is *sonification*. Just as visualization can help humans interpret and understand data, it is imaginable, for example, to convert one or more dimensions of a multidimensional data set into sound. [11, p. 67]

3.1.3 Haptic Displays

Sometimes it is necessary, even in the real world, to touch something to convince ourselves of its realness. Simulating physical contact can, therefore, convey a great deal of information about a virtual object to a user. Mechanical devices are needed to transport those information such as weight, surface texture or rigidity to a user. The

(a) (b)

Figure 3.5: Two haptic displays. (a) A version of the ground-referenced *Phantom* with an integrated stylus-like device. (Image courtesy of SensAble Technologies, www.sensable.com) (b) The CyberGrasp haptic force display. (Image courtesy of Immersion Corporation, www.immersion.com)

Phantom is a well-known ground-referenced haptic device which is able to simulate contact with hard objects (see for example [12, pp. 104–105]). The advantage of such ground-referenced devices is that a user does not need to carry their weight, usually increasing comfort. On the other hand they can only be used within a limited range. Body-referenced devices are usually connected to a part of the user which allows the user to move freely, solving the problem of limited interaction range. However, since the user needs to carry the device, size and weight are critical factors that must be considered. An example of a body-referenced device is the *CyberGrasp* (see for example [12, pp. 107–108]). Yet another type of haptic displays are tactile devices. Those are usually much smaller and lighter than *force displays*, i.e. the aforementioned body-referenced and ground-referenced devices. Normally, they use actuators like vibrators as those common in modern video game controllers. The simulation of temperature is also possible but not very common. [12, pp. 92–110][84, pp. 79–80]

3.2 Input Devices

The selection of input devices for virtual environments is just as important as the choice of output devices, as a user needs them to communicate with an application. Which devices are needed usually depends heavily on the type of application and the tasks that needs to be supported; thus, they are often designed for a specific cause. However, input devices are not only used to allow interaction, but also to display information correctly to immerse a user. The most characteristic property of an VR input device is its degree of freedom (DOF). The degrees of freedom identify how many independent translational displacements and rotations can be detected by the device. E.g. common tracking devices are able to determine the three-dimensional position as well as the orientation in space of a tracked object, therefore, allowing six degrees of freedom. It is also possible for devices with smaller DOF to emulate higher

Figure 3.6: The *Pinch* glove, a discrete VR input device. (Image courtesy of Mechdyne former Fakespace Corporation, www.mechdyne.com)

DOF. For instance, on a generic computer mouse which only allows two DOF, the wheel could be used to provide a third degree of freedom.

Input devices can be classified as either discrete, continuous or a combination of both. Discrete input hardware generates one event at the time due to an action of a user. In most cases this will be the pressing of a button or the like. The most common discrete input device for desktop applications is the keyboard. In VR the use of a keyboard is usually impractical. There the *Pinch glove*, for example, would be more suitable (see for example [11, p. 297]). It is a glove that uses sensors to detect contact of two fingertips to generate the event (see Figure 3.6). Continuous input devices generate a stream of data, e.g. position, orientation, acceleration, etc. In practice, a combination of continuous and discrete input devices is often used. In this case a user often induces the continuous sampling of a certain property by generating an event, like pressing a button. Such combined devices are frequently used in connection with event-generating recognition systems, e.g. speech or gesture recognition. Input devices also need to support low latency to ensure real-time interactivity and high update rates for an ample temporal resolution.

The most common continuous devices for VEs are *trackers* and *data gloves*. In virtual environments it is necessary to know position and orientation of a user to ensure correct stereopsis and motion parallax. Therefore, the provision of a tracking system is most essential for VR applications. *Magnetic* and *optical* tracking systems are the most common techniques. Magnetic trackers contain a device (*source*) that transmits a magnetic field of low-frequency. Bringing small sensors (*receivers*) into the calibrated magnetic field will induce voltage that can be used to estimate position and orientation in respect to the source. The problem is that any other conductive material will interfere with the magnetic field and cause a loss of accuracy and possibly the need for recalibration. Common optical tracking systems usually emit infrared light to track retroreflective markers. Triangulation of points from different cameras allows the calculation of the exact position of the marker in 3D space. Combining several markers in a fixed constellation to form one target makes it possible to also retrieve the orientation of the target. Those targets can be attached to stereo glasses or interaction devices. Optical tracking systems and the individual targets only have

to be calibrated once and produce highly accurate results. One issue to be dealt with is occlusion of targets or individual markers, especially if several tracked users can move freely within the environment. Generally, it is possible to solve this problem by strategically placing more cameras. It is obvious, however, that more cameras will complicate the tracking algorithm and increase costs. The interference of ambient light or infrared radiation also needs to be kept at an minimum, e.g. by assuring constant lighting conditions. Another problem is that most optical techniques are not quite appropriate for tracking hand or even finger movements if necessary. In this case data gloves can be deployed to retrieve hand position and even the rotation of finger joints.

Alternative tracking methods utilize sound waves, accelerometers or gyros or sense joint movements of mechanisms. Depending on the application or setup of the system, the optimum method needs to be determined by the designer based on the individual properties. [11][43, pp. 115–120][73]

3.3 Interaction

As mentioned before, interaction is one of the key elements for experiencing virtual environments. It allows a user to influence certain properties of the virtual world or objects within it. The raw data coming from the input devices is mapped in a fitting manner into the virtual environment. According to Mine, the interaction techniques implemented with suitable input devices can be categorized into the following main forms [59]: *navigation*, *selection*, *manipulation*, *scaling* and *virtual menu and widget* interaction. Scaling can be seen as part of manipulation and selection is usually closely related to manipulation. In general it is arguable how many categories are used to define interaction techniques in virtual environments depending on the point of view, the approach and the necessary detail. However, it should be apparent that *navigation*, *manipulation* and some sort of *system control* are three generic categories inevitable for creating an interactive VR experience [11, 73]. Several techniques have been created to perform interaction tasks in VE. Gesture recognition is one of them and has actually been successfully applied to navigation [64], selection [66] and manipulation [78] tasks.

Navigation is always a part of a virtual environment in some part. While this interaction is mostly perceived just as moving through the virtual world, it is actually a combination of such movement (*travel*) and *wayfinding*. The purpose of wayfinding is to enable a user to figure out the current location and a path to a future destination. To achieve this goal a user needs to construct a cognitive map of the synthetic surroundings. This process is usually supported by wayfinding aids such as placing key landmarks, providing a map or instruments like a compass. After users find out where they are and where to go, they need a way of getting there. Physical movement is obviously the most natural way of exploring a virtual environment but might not be the most fitting. Humans are able to get accustomed to new interfaces. This skill can be taken advantage of when designing new interfaces. The method of control used

by the interface depends on the type of experience the application should provide. The most common are physical and virtual controls. Physical controls utilize physical devices such as steering wheels, usually in an attempt to recreate the experience of controlling a specific vehicle. The biggest advantage is the haptic feedback users receive automatically as they interact with a device. Virtual controls have the advantage of being more generic, flexible and without constraints of the real world. They can also easily be used to emulate physical devices. Since they are part of the virtual world, they can be positioned arbitrarily within the scene or even be hidden until specifically needed. Independent from the type of control a user needs to be able to control velocity and direction of travel. The speed depends on the distance between objects in a scene. If the participant needs to move across long distances it makes sense to travel at higher speeds. The method of travel dictates whether a user must be capable of changing the direction. E.g. when using the *ride along* travel paradigm, the path is predetermined by the application, but in a fly-through or walk-through environment, it would not make sense to constrain a user to movement on a straight line only. [73, pp. 332–362]

Being capable of manipulating objects in a virtual environment or even the entire virtual world is what makes VR applications so interesting. The most common manipulations are positioning and sizing objects, applying force to virtual objects, modifying global or object attributes and changing the state of virtual controls. In the majority of cases this will require an additional selection process to determine what is to be modified. Selection is either performed before or sometimes at the same time as the manipulation. Methods to achieve both selection and manipulation are manifold and will not be covered by this thesis in detail. A good overview is presented in [73, pp. 286–332]. Similar to the navigational task, most manipulations can be achieved through physical or virtual controls, but direct user controls and agent controls are also common (see Figure 3.7). Agent control is a method where an aide actually performs a task given by a user. In most cases speech control will be used for this type of interaction, but gestures are also common. An example is MIT's *Officer of the Deck* [93] where a trainee commands a virtual submarine crew. In *direct user control*, the application tries to simulate the real world interaction between a user and an object. The *grab with fist* interaction is such a method where the participant needs to close the hands to form a fist. The object close to the hands is "grabbed" and can be moved with the hands until the fists are unclenched. Direct user control is probably the most natural way of manipulating virtual objects. However, just as is the case with navigation, the most natural form is not necessarily the most adequate. One major advantage of virtual reality is that manipulation is not bound to the same restrictions as in the real world. A designer has much more freedom in creating an appropriate experience for the given task at hand. Often when people speak of a natural interface they actually mean an intuitive interface, one that is easy to learn. However, in the case of interfaces it is simply a matter of familiarization. If someone is used to a certain method of interaction, any similar method will be easy to use quickly, even if the familiar way is not natural at all. A good example is the computer mouse. People have become accustomed to interacting with a desktop environment

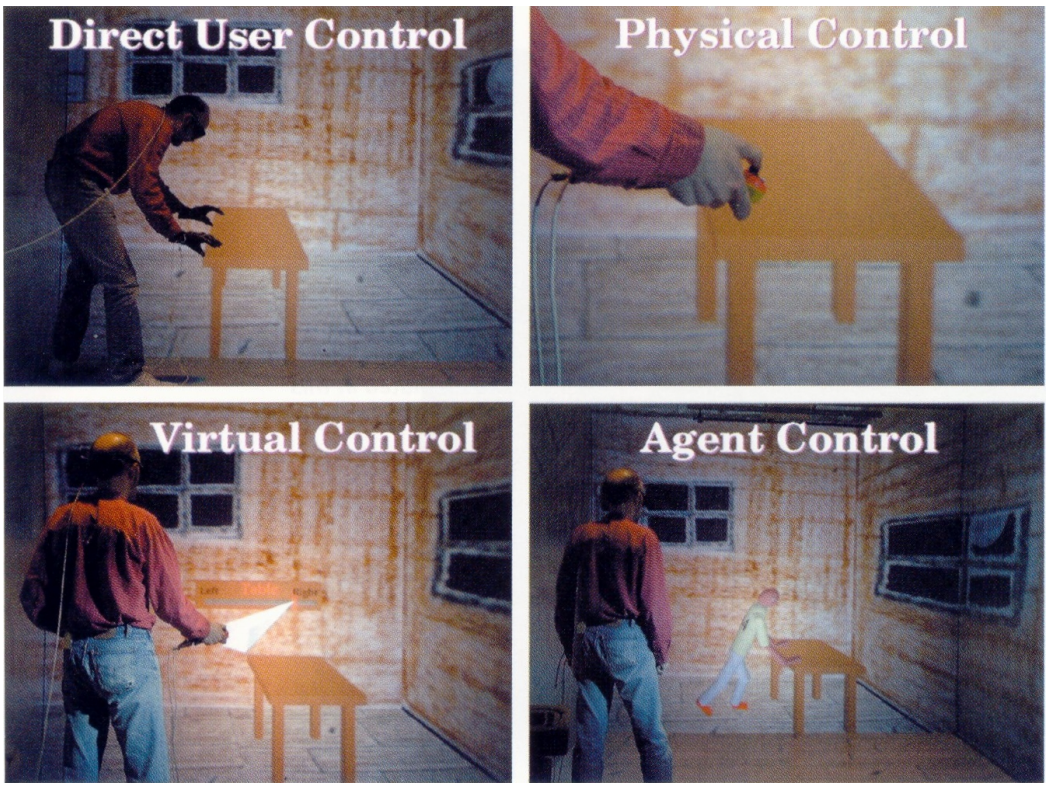


Figure 3.7: Four categories of manipulation in VR applications. An example for each class shows a user moving a table. (Image from [73])

with this device. Although no interaction we perform in nature is similar to the usage of a computer mouse, every new method that relates to this methodology is quickly dubbed natural. [73]

To allow navigation as well as manipulation, the participant needs to be capable of communicating with the application, i.e. to control the system. In most cases this will involve a physical input device and a variety of virtual controls. Some considerations about the way users communicate within a virtual environment have already been presented in the two preceding paragraphs. Besides performing navigation and manipulation tasks, it might also be necessary to pass *metacommands* to the application. These are commands that control the underlying simulation, e.g. loading a new data set or scenario or undoing an action. Since controlling an application on this level will usually decrease the immersion effect, it may be intended by the designer to specifically allow or deny the participant to perform such actions. Thus, it is important to consider what kind of experience is meant to be achieved. Collaborative environments are another case that must allow communication between all participants. Only then is it possible to share the VR experience, as well as thoughts, suggestions and ideas. If users are in the same room using seethrough glasses or HMDs like when using the *TwoView* display, this is not an issue. The participants can see each other, see what the other is doing and easily talk to each other. This

changes as soon as the participants do not share the same room. It is usually not enough to display the absent user's actions in the local environment. Collaboration relies heavily on communication, especially aural and visual signals. Therefore, the application should relay a participants voices and maybe even their body movements so the others are able to perceive them. [73, pp. 362–377]

Which type of control to use depends on the experience that is meant to be created. If the goal is to create an environment which is as realistic and immersive as possible, the choice of truly natural interaction such as the *direct user control* is reasonable. However, in many contexts such as the sciences or in the industry, a crucial factor is time. The value of getting a task done more quickly is appreciated by many professionals. In this case the interface does not necessarily have to be natural, it must allow efficient work. One way to achieve this efficiency is the provision of *expert controls*. These controls are usually neither natural nor intuitive, but must be learned and memorized. The classic example are key combinations to quickly call a certain function of an application, such as pressing the *crtl*-key and the *c*-key simultaneously to copy text or images (cf. [55, pp. 67–68]). Once the controls are memorized they can be deployed to rapidly achieve tasks that would otherwise require several subtasks and slow down the overall workflow. In the example it would mean selecting the text or image to be copied, opening a menu, finding and selecting the copy command instead of just using the key combination. In VR applications, input devices normally lack a great number of keys or buttons, limiting the adoption of the mentioned key combination method. Thus, a different technique is necessary to achieve the same effect without disturbing the workflow. Voice or gesture input are common choices to create similar short cuts.

3.4 VR Software Frameworks

VR software frameworks are the basis for the development of interactive VR applications. They offer access to relevant technology and contain abstractions for tracking systems, input and output devices, as well as scene graph and rendering APIs. In most cases they will also include helpful software libraries and provide a scripting interface for rapid application development. Several frameworks exist that all have certain focal points, pursue a specific approach or were designed for a particular problem. Examples are VR Juggler [4], HECTOR [87], Lightning [7], Avalon [2] and *basho* [56]. The VR framework utilized for this thesis is called AVANGO NG [47]. A description can be found in Section 5.3 on page 39.

4 Hidden Markov Models

Hidden Markov Models (HMMs) had been studied long before the advent of virtual environments. Named after the Russian mathematician Andrei Andrejewitsch Markov, they were first mentioned by Leonard Baum and his colleagues in the late 1960s. However, despite several publications in the 1960s and 1970s, this statistical model had not been commonly used in the field of pattern recognition until the 1980s [70]. HMMs are mostly known for their application in speech recognition [27, 37, 51] and more recently also as a tool for analysis of biological sequences such as DNA [29]. Today HMMs are widely used in many fields, for instance robotics [50, 92], recognition of handwriting [24], fraud detection [77], gesture recognition (see Chapter 2) and many more.

When reading about systems similar to the one presented in this thesis, it is quite noticeable that many methods employ Hidden Markov Models for the task of recognizing gestures. Due to this widespread usage and their ability to model sequences of events with temporal and dynamic variations, this approach was chosen for the implementation. The theoretical basics of Hidden Markov Models covered in this section are fairly easy presuming some basic knowledge of statistics. The application of HMMs in practice, however, leads to a couple of problems which will be described in Section 5.

4.1 Definition

Since HMMs are applied to various problems of pattern recognition and machine learning, a few introductory tutorials have been published. For example [27], [29], [57] or [70] give good overviews to help readers understand the theory of Hidden Markov Models from a more practical point of view. An HMM is a doubly stochastic process with two levels. The first level is described by an underlying Markov chain with a finite set of N states: $s_1, \ldots, s_N$. The stochastic process describes transitions between those states at regularly spaced discrete times based on transition probabilities associated with each state. Calculating the probability of the system being in a certain state at a specific time $t = 1, 2, \ldots$ would require a description of the current state and all predecessor states. However, an additional restriction, the *Markov property*, simplifies this problem by considering only a finite set of predecessor states. This means, in the case of a discrete, first order Markov chain, which is most commonly used, the present state depends only on the very last predecessor state. If S_i describes the state at time $t = i$ for $i \geq 1$, then the description is simplified as follows:

$$P(S_t|S_1, S_2, \ldots, S_{t-1}) = P(S_t|S_{t-1}). \tag{4.1}$$

The Markov chain is therefore *memoryless* making the transitions time independent. Hence, the temporal behavior can be described by a set a_{ij} of the form

$$a_{ij} = P(S_t = s_j | S_{t-1} = s_i), \qquad 1 \leq i, j \leq N \tag{4.2}$$

where $a_{ij} \geq 0$ and $\sum_{j=1}^{N} a_{ij} = 1$.

On the second level an output is generated at each time t, usually referred to as emission O_t. The probability for generating a certain output at the given time depends only on the current state.

$$P(O_t | O_1, \ldots, O_{t-1}, S_1, \ldots, S_t) = P(O_t | S_t) \tag{4.3}$$

The sequence of emissions from the model is the only thing that is revealed to an observer. Therefore, the emissions are sometimes referred to as observables. The rest of the model stays hidden, hence, the name *Hidden* Markov Model. A first order HMM λ is completely defined by:

- a finite set of states $\{s_1, \ldots, s_N\}$,
- a transition probability matrix $\boldsymbol{A} = \{a_{ij} | a_{ij} = P(S_t = s_j | S_{t-1} = s_i)\}$,
- an initial state distribution $\boldsymbol{\pi} = \{\pi_i | \pi_i = P(S_1 = s_i)\}$ describing the probability of being the initial state for each state,
- and an output probability matrix with state specific probabilities
  $\boldsymbol{B} = \{b_j(o_k) | b_j(o_k) = P(O_t = o_k | S_t = s_j)\}$ for *discrete HMMs* or
  $\boldsymbol{B} = \{b_j(x) | b_j(x) = p(x | S_t = s_j)\}$ for *continuous HMMs*.

The complete HMM is usually denoted as triple $\lambda = (A, B, \pi)$. In the case of discrete HMMs the observables of the model are of a discrete set of symbols $\{o_1, \ldots, o_M\}$, sometimes also referred to as alphabet. Then $b_j(o_k)$ are discrete probability distributions. Discrete HMMs are often used to introduce the concept of the Hidden Markov Models, but they usually require a preprocessing step to quantize continuous data into a sequence of discrete observations. This additional step often renders them unfeasible for many applications. Today continuous HMMs are used for the majority of applications. For continuous models, the observations are vectors $x \in \mathbb{R}^n$ and emissions are described based on continuous density functions $b_j(x) = p(x | S_t = s_j)$. However, the majority of papers found during research use discrete HMMs for simplicity. Further, to appropriately represent the distributions in $\mathbb{R}^n$ parametric descriptions are needed. Those are known, however, for a small subclass of distributions only, e.g. the Gaussian distribution. Those kinds of distributions will usually not fit the problem, since the designated issues are always described by unimodal distributions. As a solution it is common to use *mixture densities* which further complicates the issue. Consequently, the work presented in this thesis and therefore the description of HMMs will deal with the discrete case only.

Due to the underlying structure of a Hidden Markov Model, it can be seen as a finite state machine. The state transitions are described by the probabilities a_{ij}

mentioned above. In the most general case, every a_{ij} is positive, i.e. each state can be reached directly from any other state in the chain. HMMs with this property are called *ergodic* or *fully connected* models (see Figure 4.1 (d)). This case, however, is often undesirable because the more transitions are possible, the more parameters need to be trained and the more paths through the model are possible. Therefore, it makes sense to limit the number of possible state transitions for certain applications. When trying to recognize speech, handwriting or gestures, the signals have a distinct temporal order which suggests a linear transition of states (see Figure 4.1 (a)). This is the most simple topology meeting the temporal property. It allows only transitions to the same or the next state. In the case of a four-state model, the transition matrix would be

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{pmatrix} \tag{4.4}$$

where $a_{44} = 1$. The small number of parameters decrease the complexity and increase manageability. The disadvantage of the linear HMM is that it is too restrictive. Loops allow for variation in length of the modeled signal, but all states need to be traversed. A topology used to loosen these restriction is the left-right model, which allows transitions to all predecessor states but not to any previous states (see Figure 4.1 (b)). In this case the state transitions have the property

$$a_{ij} = 0, \qquad j < i \tag{4.5}$$

In this way it is possible to "skip" parts of a signal by still accounting for its linearity. To prevent missing parts that are too long, constraints on the transition probabilities like

$$a_{ij} = 0, \qquad j > i + \Delta \tag{4.6}$$

are introduced. In speech and handwriting recognition Δ is often set to 2. This common topology is referred to as the Bakis model (shown in Figure 4.1 (c)). One problem with all of the topologies, minus the ergodic, is that start and final state need to be known. A problem which is discussed in Section 5.5. One important advantage of HMMs is that a combination of HMMs is still an HMM. Thus, even the different topologies can be combined to benefit from the different advantages. In this way the flexibility of the model can be improved while at the same time limiting the complexity. [29, pp. 67,127–128][60][70][92]

4.2 Parameter Estimation

A completely defined HMM as described in the previous section can easily be used to produce a sequence of observations. Starting with a state, chosen using the initial state distribution $\boldsymbol{\pi}$, an output can be generated and a state transition can be performed at each time step according to the probability distributions associated with the current

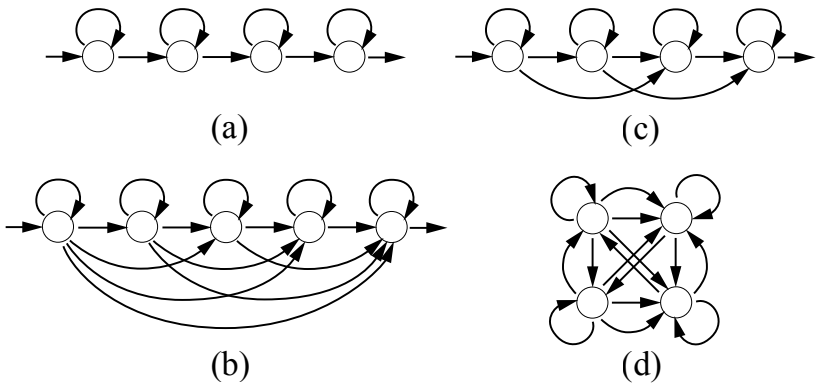


Figure 4.1: Illustration of different HMM topologies. (a) linear model, (b) left-right model, (c) Bakis model, (d) ergodic model. (Illustration from [29, p. 128])

state. However, given such a model, there are three basic problems that need to be solved in order to make HMMs useful to meaningful applications. Those are the evaluation problem, the decoding problem and the learning problem which will now be described in more detail. [70]

4.2.1 Evaluation

One essential question when dealing with Hidden Markov Models is how well a certain pattern, i.e. an observation sequence $\boldsymbol{O} = O_1, O_2, \ldots, O_T$, is described by a given model $\lambda = (A, B, \pi)$. The goal is to find out how likely it is that the model produced the sequence, $P(\boldsymbol{O}|\lambda)$. A straightforward approach to computing $P(\boldsymbol{O}|\lambda)$ is very intuitive and easy. Since an observation is generated by a hidden state, a state sequence $\boldsymbol{s} = q_1, q_2, \ldots, q_T$ of the same length as $\boldsymbol{O}$ is needed, whereas q_t is the actual state at time t. The probability of a corresponding observation sequence must be calculated along such a fixed sequence.

$$P(\boldsymbol{O}|\boldsymbol{s}, \lambda) = \prod_{t=1}^{T} b_{q_t}(O_t) \tag{4.7}$$

The probability of an arbitrary state sequence is given by the product of transition probabilities and the initial state distribution. By defining $a_{0i} := \pi_i$ and $q_0 := 0$ the equation can be simplified.

$$P(\boldsymbol{s}|\lambda) = \pi_{q_1} \prod_{t=2}^{T} a_{q_{t-1},q_t} = \prod_{t=1}^{T} a_{q_{t-1},q_t} \tag{4.8}$$

A combination of Equations (4.7) and (4.8) yields the probability that a model λ produced a given sequence of observables $\boldsymbol{O}$ along a certain state sequence $\boldsymbol{s}$.

$$P(\boldsymbol{O}, \boldsymbol{s}|\lambda) = P(\boldsymbol{O}|\boldsymbol{s}, \lambda)P(\boldsymbol{s}|\lambda) = \prod_{t=1}^{T} a_{q_{t-1},q_t} b_{q_t}(O_t) \tag{4.9}$$

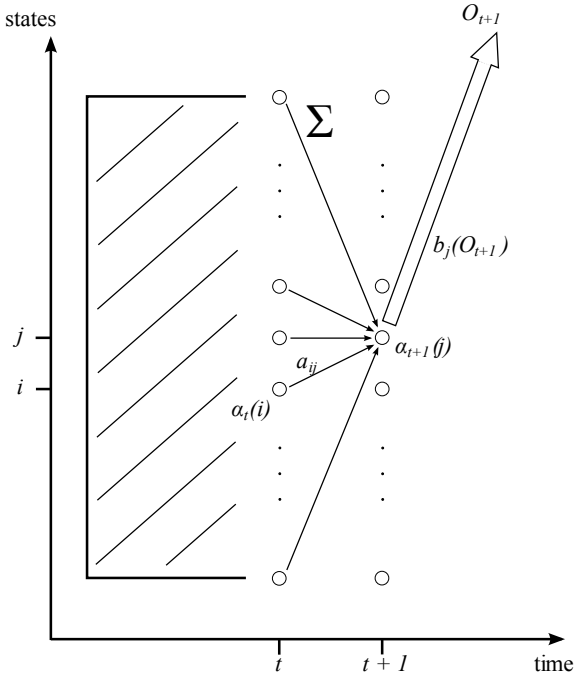


Figure 4.2: Illustration of the steps performed to compute forward variables $\alpha_t(i)$ with the *forward algorithm*. (Illustration from [29, p. 76])

Finally, the overall probability that the given sequence $\boldsymbol{O}$ was produces by the model λ is given by summing the probabilities of all possible state sequences of length T through the model.

$$P(\boldsymbol{O}|\lambda) = \sum_{\boldsymbol{s}} P(\boldsymbol{O}, \boldsymbol{s}|\lambda) = \sum_{\boldsymbol{s}} P(\boldsymbol{O}|\boldsymbol{s}, \lambda) P(\boldsymbol{s}|\lambda) \tag{4.10}$$

Although the calculation of $P(\boldsymbol{O}|\lambda)$ with Equation (4.10) is obvious, it is virtually impossible to be done this way in practice. Since there are N^T possible state sequences for which approximately $2T$ calculations have to be performed the complexity is $O(TN^T)$. However, taking advantage of the Markov property, the *forward algorithm*, which is illustrated in Figure 4.2, is a procedure to solve this problem much more efficiently. Due to the restricted memory of the model, the calculation for the next time step $t+1$ depends only on all possible states at time t which are always the N states of the model at most. Therefore, it is possible to inductively calculate a *forward variable*

$$\alpha_t(i) = P(O_1, O_2, \ldots, O_t, S_t = s_i|\lambda) \tag{4.11}$$

giving the probability of producing the partial sequence $O_1, O_2, \ldots, O_t$ and being in state s_i at time t for a model λ. At first all $\alpha_1(i)$ must be initialized for $t = 1$ using the initial state distributions π and emission probabilities for each state.

$$\alpha_1(i) = \pi_i b_i(O_1) \tag{4.12}$$

To expand a partial sequence with a new observation O_{t+1}, all possibilities of generating this sequence and ending in a certain state s_j must be taken into account. This is done by summing all $\alpha_t(i)$, because they represent the probability of generating $O_1, O_2, \ldots, O_t$, moving to the next step via the transition probabilities for getting from s_i to s_j, and the probability of observing O_{t+1}.

$$\alpha_{t+1}(j) = \sum_i \{\alpha_t(i) a_{ij}\}\, b_j(O_{t+1}) \tag{4.13}$$

After calculating all forward variables for $1 \leq t \leq T-1$, the algorithm yields N probabilities $\alpha_T(i)$ of generating $\boldsymbol{O}$ with an arbitrary state sequence stopping in state s_i. To get the complete probability of producing this sequence, all of the "final" forward variables simply have to be summed up.

$$P(\boldsymbol{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{4.14}$$

Alternatively, the probability $P(\boldsymbol{O}|\lambda)$ can also be used for classification purposes which is very useful when working with HMMs. If there is more than one HMM, which is usually the case, classifying the sequence $\boldsymbol{O}$ can be done by finding the maximum a posteriori probability $P(\lambda_j|\boldsymbol{O})$. With the Bayes formula this probability can be calculated as

$$P(\lambda_j|\boldsymbol{O}) = \max_i \frac{P(\boldsymbol{O}|\lambda_i)P(\lambda_i)}{P(\boldsymbol{O})}. \tag{4.15}$$

Since $P(\boldsymbol{O})$ is constant for one given emission sequence, only the numerator of Equation (4.15) needs to be calculated. The a priori probability $P(\lambda_i)$ is usually disregarded for reasons of simplicity or because all models are equally likely. Hence, classification is mostly done using only the probability $P(\boldsymbol{O}|\lambda_i)$, which can be efficiently computed using the described *forward algorithm*. [29, pp. 70–79][70][91]

4.2.2 Decoding

As shown, the *forward algorithm* computes $P(\boldsymbol{O}|\lambda_i)$ for all possible state sequences. In some cases, however, it is necessary to find the single state sequence that most likely produced the observations. Due to the nature of HMMs, it should be obvious that there is not an exact solution to this problem. Particularly, because there are several criteria to define an "optimal" sequence of states. Probably the most common criteria is to maximize the a posteriori probability of producing the observations given a model by finding the best state sequence $\boldsymbol{s^*}$.

$$\boldsymbol{s^*} = \operatorname*{argmax}_{\boldsymbol{s}} P(\boldsymbol{s}|\boldsymbol{O}, \lambda) \tag{4.16}$$

The probability from Equation (4.16) can be rewritten using Bayes rule:

$$P(\boldsymbol{s}|\boldsymbol{O}, \lambda) = \frac{P(\boldsymbol{O}, \boldsymbol{s}|\lambda)}{P(\boldsymbol{O}|\lambda)} \tag{4.17}$$

Since $P(\boldsymbol{O}|\lambda)$ will be constant for one observation sequence, it is enough to regard the numerator of the right hand side of Equation (4.17) to find the optimal state sequence:

$$\boldsymbol{s^*} = \underset{\boldsymbol{s}}{\operatorname{argmax}} P(\boldsymbol{s}|\boldsymbol{O}, \lambda) = \underset{\boldsymbol{s}}{\operatorname{argmax}} P(\boldsymbol{O}, \boldsymbol{s}|\lambda) \tag{4.18}$$

To solve this problem a procedure called *Viterbi algorithm* was introduced by A. J. Viterbi in 1967. This technique works almost as the *forward algorithm* mentioned in Section 4.2.1 with the main difference being that the partial probabilities from the predecessor states are not summed but maximized. After initialization each partial probability for all states and all t, $t = 1, \ldots, T-1$, is calculated according to Listing 4.1:

$$\delta_{t+1}(j) = \max_i \{\delta_t(i) a_{ij}\} b_j(O_{t+1}) \tag{4.19}$$

However, this will find only the maximum probability of producing a partial observation sequence at each time step. In order to find the optimal state sequence that produced the entire observation sequence, it is necessary to define another variable $\psi_t(j)$ that will memorize the state that led to the maximization at each time, i.e. the optimal predecessor for each $\delta_t(j)$.

$$\psi_t(j) = \underset{i}{\operatorname{argmax}}\, \delta_{t-1}(i) a_{ij} \tag{4.20}$$

After evaluating the entire observation sequence, the state that maximizes $\delta_T(i)$ marks the end of the optimal state sequence, denoted as s_T^*. Backtracing all of the other states will extract the optimal path $\boldsymbol{s^*}$.

$$s_t^* = \psi_{t+1}(s_{t+1}^*) \tag{4.21}$$

The optimal path is often referred to as *Viterbi path* since it is calculated using the *Viterbi algorithm* shown in Listing 4.1. [27, pp. 79–82][29, pp. 79–81][30][70]

Define: $\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P(O_1, O_2, \ldots, O_t, q_1, q_2, \ldots, q_{t-1}, q_t = s_i|\lambda)$

1. **Initializing**
   $\delta_1(i) := \pi_i b_i(O_1)$
   $\psi_1(i) := 0$
2. **Recursion**
   for all $t = 1, \ldots, T-1$:
   $\delta_{t+1} := \max_i \{\delta_t(i) a_{ij} b_j(O_{t+1})$
3. **Termination**
   $P(\boldsymbol{O}, \boldsymbol{s^*}|\lambda) = \max_i \delta_T(i)$
   $s_T^* := \underset{i}{\operatorname{argmax}}\, \delta_T(j)$
4. **Backtracing of optimal path**
   for all $t = T-1, \ldots, 1$:
   $s_t^* = \psi_{t+1}(s_{t+1}^*)$

Listing 4.1: *Viterbi algorithm* to determine the optimal state sequence. [29]

4.2.3 Learning

In order to obtain good recognition results, the parameters of a model must be adjusted to fit a certain pattern. A training process using observed sequences of the real phenomenon achieves this optimization. Since no analytical way of solving this problem is known, iterative procedures are normally used. The most common is the *Baum-Welch algorithm* which uses $P(\boldsymbol{O}|\lambda)$ to optimize the parameters. Using a training sequence the *Baum-Welch algorithm* reestimates the parameters of the model λ so that the new model λ' produces this sequence with a higher or equal probability.

$$P(\boldsymbol{O}|\lambda') \geq P(\boldsymbol{O}|\lambda) \tag{4.22}$$

The algorithm is based on a variable $\gamma_t(i)$ denoting the a-posteriori probability of being in state s_i at time t. To efficiently calculate these values, the *forward-backward algorithm* is used, an extension of the aforementioned *forward algorithm*. The forward variable $\alpha_t(i)$ marks the probability of being in state s_i at time t for a partial observation sequence $O_1, O_2, \ldots, O_t$. In a similar manner a *backward variable* $\beta_t(i)$ is calculated to yield the probability of producing the remaining sequence $O_{t+1}, O_{t+2}, \ldots, O_T$ starting in state s_i for a model λ.

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots, O_T|q_t = s_i, \lambda) \tag{4.23}$$

The only difference is that the *backward algorithm* must start at the end. Therefore, the backward variables are initialized for time T. Obviously, the probability of producing no further observation at this time is 1, thus

$$\beta_T(i) = 1 \tag{4.24}$$

Like the forward variables, every backward variable $\beta_t(i)$ can be calculated accounting for all successive variables $\beta_{t+1}(j)$, the output probability at time t and the according state transition probabilities.

$$\beta_t(i) = \sum_j a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \tag{4.25}$$

Just as the *forward algorithm*, this procedure calculates the probability of producing the observation sequence $P(\boldsymbol{O}|\lambda)$ by summing all backward variables at time $t = 1$ regarding the initial state distribution.

$$P(\boldsymbol{O}|\lambda) = \sum_{i=1}^{N} \pi_i b_i(O_1) \beta_1(i) \tag{4.26}$$

From the definition it should be obvious that the multiplication of $\alpha_t(i)$ and $\beta_t(i)$ is equal to the probability $P(S_t = s_i, \boldsymbol{O}|\lambda)$. With a combination of both the *forward* and the *backward algorithm* and using Bayes rule, the sought state probability $\gamma_t(i)$ can now be calculated.

$$\gamma_t(i) = P(S_t = s_i|\boldsymbol{O}, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{P(\boldsymbol{O}|\lambda)} \tag{4.27}$$

Additionally, the probabilities that a transition from s_i to s_j occurred at time t can be determined similar to Equation (4.27).

$$\begin{aligned} \gamma_t(i,j) &= P(S_t = s_i, S_{t+1} = s_j | \boldsymbol{O}, \lambda) \\ &= \frac{P(S_t = s_i, S_{t+1} = s_j, \boldsymbol{O}|\lambda)}{P(\boldsymbol{O}|\lambda)} \\ &= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(\boldsymbol{O}|\lambda)} \end{aligned} \tag{4.28}$$

With $\gamma_t(i)$ and $\gamma_t(i,j)$ it is possible to define formulas that reestimate the model parameters a_{ij}, π_i and $b_j(o_k)$.

$$\pi'_i = P(S_1 = s_i | \boldsymbol{O}, \lambda) = \gamma_1(i) \tag{4.29}$$

$$a'_{ij} = \frac{\sum_{t=1}^{T-1} P(S_t = s_i, S_{t+1} = s_j | \boldsymbol{O}, \lambda)}{\sum_{t=1}^{T-1} P(S_t = s_i | \boldsymbol{O}, \lambda)} = \frac{\sum_{t=1}^{T-1} \gamma_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{4.30}$$

$$b'_j(o_k) = \frac{\sum_{t:O_t=o_k} P(S_t = s_j | \boldsymbol{O}, \lambda)}{\sum_{t=1}^{T} P(S_t = s_j | \boldsymbol{O}, \lambda)} = \frac{\sum_{t:O_t=o_k} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{4.31}$$

The updated state transition probabilities a'_{ij} are calculated by summing all single transition possibilities for all possible time steps, namely the expected number of transitions from state s_i to state s_j. Dividing by the expected total number of transitions from state s_i normalizes the new parameter. The improved initial state probabilities π'_i are a special case of the transition probabilities accounting for the expected number of times the sequence will start at state s_i. Similarly, the emission probabilities $b'_j(k)$ can be reestimated by dividing the expected number of times being in state s_j and observing o_k by the total number of observables generated while being in state s_j. The updated parameters are the result of one iteration of the algorithm. The process is repeated using λ' in place of λ until the model satisfactory describes the training data or until no further improvements are achieved. In most cases the training data will consist of more than one observation sequence, e.g. several repetitions of a single gesture. To deal with this occurrence, the presented training algorithm does not have to be modified. The only difference is that the values for updating the parameters need to be summed over all sequences. While for the *Baum-Welch algorithm* all possible state sequences are considered, it is possible to restrict the learning to the optimal state sequence using *Viterbi training*. The criterion for optimization is the probability of producing the training sequence along the most likely path, $P(\boldsymbol{O}, \boldsymbol{s^*}|\lambda)$. This process is faster and easier to implement, but requires more training sequences than the *Baum-Welch algorithm* which extracts more information from the training data. As a result, a combination of both algorithms is used in practice. The *Viterbi training* is used to prepare a model for a subsequent training by the *Baum-Welch algorithm*. Due to the preparation phase, the *Baum-Welch algorithm* will take less time to calculate an adequate model. Another common technique is the *segmental k-means algorithm* [40] which uses the same probability as the *Viterbi training*, as well as an algorithm

for vector quantization[6] to reestimate parameters of a mixed density model. With a few alterations this procedure becomes the only formal algorithm to determine the initial parameters of HMMs. For models with mixed densities, this algorithm also converges faster than the *Baum-Welch algorithm.* [27, pp. 82–85][29, pp. 83–96][70]

4.3 Computational Consideration

Apart from practical problems such as parameter initialization, the implementation of HMM theory causes considerable issues. These need to be dealt with if an implementation is to be of high quality. When dealing with HMMs, many operations require handling probability values which quickly approach zero, especially when computing the forward and backward variables. As a result the risk of numerical inaccuracy arises. A straightforward method to avoid underflow is the introduction of scaling factors that can usually be canceled out easily. The main problem with scaling methods is that often they are error-prone and laborious.

An alternative and more common procedure is to use a logarithmic representation of small probability values. Thus, the original value p is usually transformed as follows:

$$\tilde{p} = -\log_b p \tag{4.32}$$

The accuracy of current floating point formats is sufficient for the resulting range. The choice of the base of the logarithm has no significant influence on the precision of the values. However in practice, the Euler's number e is used predominantly as the base, since virtually every standard library includes efficient implementations of the natural logarithm and its inverse function e^x. Furthermore, using the natural logarithm simplifies evaluation of the normal distribution density. The only drawback with this representation is the issue of adding two probability values, as it might require to delogarithmize and add the values and then take the logarithm again. However, there are ways to minimize the complexity of adding two numbers (see for example [29, p. 121]).

To increase efficiency of the *Baum-Welch* training, it is appropriate to apply thresholding. The reestimation results depend on the γ values only. Comparatively small γ's will have only little influence on the summations and could be disregarded. Since the value of a γ is mainly determined by the corresponding forward and backward variables they can be set to zero if they are too small compared to others. This will obviously save operations and thus reduce the computational load. Unfortunately, a fitting threshold must be selected empirically since no analytical solution exists. Yet the importance of choosing an adequate threshold must not be underestimated since it heavily influences the quality of the outcome.

A technique similar to thresholding is *flooring.* Besides very small probability values, it is possible that an event is unobservable, i.e. the probability is zero. Using the mentioned logarithmic representation, it would result in a value of infinity or

[6] In general the *k-means* algorithm is used for reasons of efficiency.

require mapping of the value onto the largest possible float, possibly causing overflow. A heuristically determined maximum $\tilde{p}_{max}$ prevents the occurrence of this problem. In terms of probabilities it means that values do not fall below a certain lower bound p_{min} which is the reason for the name of this method.

$$\tilde{p}_{max} = -\ln p_{min} \tag{4.33}$$

Additionally, flooring the emission probabilities assures that affected states will still be regarded during the reestimation process. It will also circumvent the immediate exclusion of states with low probability values while calculating the best state sequence. [29, pp. 119–125][70][91]

5 Gesture Control in the VRGeo Demonstrator

The goal of the work presented in this thesis was to develop a gesture recognition and control system. To achieve it, several ideas presented in Chapter 2 have been implemented in the system to different degrees. Although the field of OCR seemed very promising for the given task, most techniques in this field work on images of the symbols to be recognized. Converting the digital data to a pixel-like representation would have been an unnecessary effort and was disregarded. However, due to the input method, the idea of Palm's unistroke alphabet [8, 54] was adapted for the recognition system. The usage of HMMs was obvious since so many other systems employ them for the purpose of recognition. The idea for the vector quantization which is described in Section 5.4.3 was derived from the idea by Elliman and Connor [26] to subdivide a bounding sphere of a character to calculate a token sequence. The problem of time variance mentioned by Wilson and Bobick [88] was solved by sampling only equidistant points in three-dimensional space. Calculating the center of mass for all sampled points as presented in [26] and [78], also became part of the preprocessing step. The important feature of providing visual feedback to a user mentioned by Stark and Kohler [78] was also integrated into the final application. Many ideas found during research were not incorporated for various reasons. Some seemed unnecessary or too complex while others were estimated as too time consuming. However, there are various promising ideas that might be integrated at some point in the future as discussed in Chapter 7.

5.1 VRGeo Demonstrator

The recognition system developed for this thesis was meant to be included in the *VRGeo* Demonstrator. The demonstrator is an immersive VR prototype software to explore seismic data. It continues to be developed at the *Fraunhofer IAIS* for the *VRGeo* Consortium. Members of the consortium are companies from the oil and gas industry as well as their suppliers of software and related hardware. Their main interest is to utilize new VR technologies for their field of work. At the two annual meetings of the consortium, the demonstrator and new features are presented to the representatives of the members. During this session the participants (experts in the field of VR, geology and geophysics among others) are able to test the application themselves and discuss their thoughts and opinions. Afterwards the consortium will present its feedback on which the research agenda for the following term will be based.

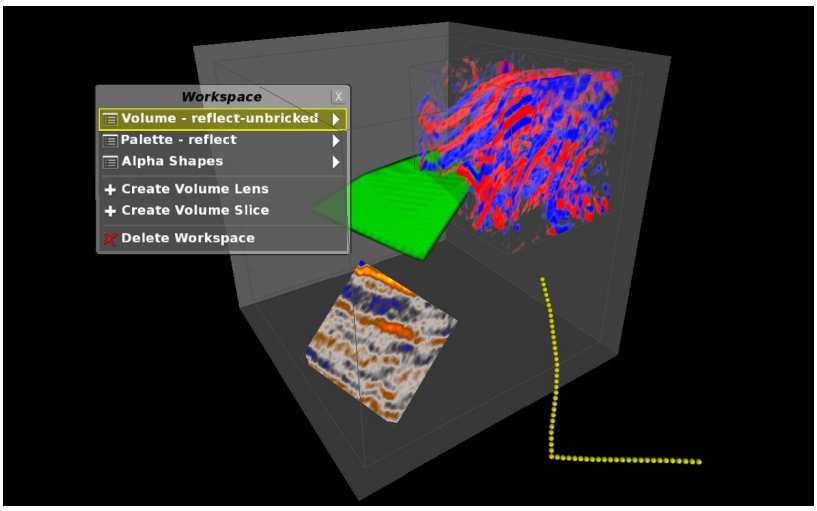
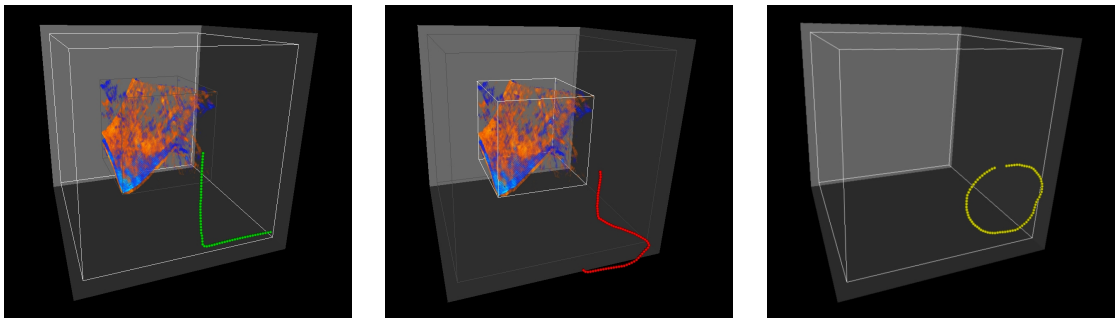


Figure 5.1: The *VRGeo* Demonstrator. The image shows a workspace (grey cube) containing a volume lens (upper right corner), a volume slice (lower left corner) and an alpha shape (center). Also visible are the 3D menu as well as a performed L gesture.

This also determines which technology will be kept as part of the demonstrator and which will not.

The *VRGeo* Demonstrator in the current version is primarily designed for the *TwoView* display (see Section 3.1.1). Two users with 3D shutter glasses and wireless 6-DOF input devices are able to work collaboratively. The glasses and input devices are tracked with an array of 4 A.R.T. infrared tracking cameras mounted at the four corners of the projection screen. With the current set up 15 individual targets can be tracked simultaneously at a frame rate of 60 Hz. A scene in the application can host workspaces into which a seismic volume data set can be loaded. Each workspace can also hold several objects: volume lenses, volume slices and alpha shapes (see Figure 5.1). Volume lenses are a cubic cutout of the volume data. A color palette is applied using 3D textures to improve the volumetric visualization. A volume slice displays the volume data loaded into the workspace along a two-dimensional intersection. Alpha shapes are geometric objects that do not have to be convex or connected. They are used to construct arbitrary shapes to model regions of uncertainty into the data. Interaction with the scene is done using a pick-ray that is emanating from the interaction device and with the help of a 3D pop-up menu [22].

The demonstrator was designed using the AVANGO VR/AR framework [81]. It had to be completely rewritten due to a conversion from AVANGO to AVANGO NG [47] (see Section 5.3). Consequently, it is implemented using the *C++* programming language and the *Python* scripting language. For the *TwoView* display, the demonstrator runs on a HP xw9400 workstation with two Dual-Core AMD Opteron 2218 processors, two NVIDIA Quadro FX 5600 and one NVIDIA Quadro G-Sync 2 graphics card hosting a CentOS 5.2 Linux.



(a) (b) (c)

Figure 5.2: Example of user feedback. (a) The intended action was to create a new volume lens. The gesture was recognized successfully and a the workspace could be referenced indicated by the green trail. (b) The intention to create a new volume slice failed as the attempted S could not be recognized. (c) The symbol was recognized correctly but the referenced object (a workspace) cannot be scaled.

Since the recognition system was to be included in the current version of the *VRGeo* Demonstrator, several data and tools could be reused. The most important advantage was the possibility to use the tracking data of the installed A.R.T. tracking cameras. These cameras provide three-dimensional coordinates of a target with submillimeter precision at a rate of 60 Hz. This highly accurate data was the starting point for all further considerations presented in this chapter. Since the input device including the tracking target is used for all interactions with the Demonstrator it had to be made clear when it would be used for the purpose of gesture input. To keep things as simple as possible, a user needs to press a certain button on the device to signal the beginning of the gesture and then perform the gesture. Releasing the button marks the end of the gesture. While the button is pressed the application takes advantage of the results of the resampling process explained in Section 5.4.1. The trail of the gesture is visualized to a user by rendering gray spheres at the position of sampled gesture points while the gesture is drawn. This way a user immediately sees what the input looks like. Additionally, after the recognition process, the trail of spheres is color coded to visualize the result of the process. If no gesture was recognized by the system, the spheres turn red. If a gesture was recognized, the corresponding action is identified and a ray is projected through the center of the gesture in the direction the input device points to. The action is then applied to the first object in the scene that is intersected by the ray. If the ray does not intersect with any object or if the action does not apply to the object found, the spheres turn yellow. Otherwise the spheres turn green to signal the success of the operation as shown in Figure 5.2. This part of the application is very important, because it gives a user the opportunity to identify errors more easily if the input did not yield the expected result. This assures a quick adaptation to the system and therefore also increases user acceptance. Reusing the available devices and data also allows a fluent integration of the gesture control in the existing workflow. The most important tasks of the demonstrator were identified

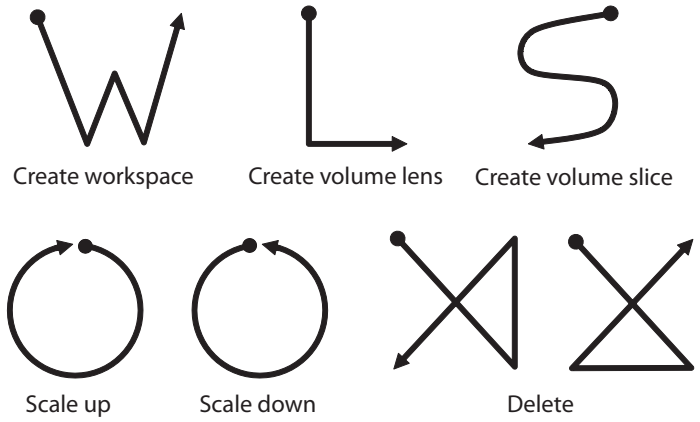


Figure 5.3: Gesture symbols used in the *VRGeo* Demonstrator.

intuitively by members of the project team. They are: scaling up/down, creating a new workspace, creating a new volume lens, creating a new volume slice and deleting. The symbols assigned to these actions were chosen so as to provide a metaphor that is plausible and simple to memorize. The symbols in the current version of the demonstrator are shown in Figure 5.3. The association of an x with the process of deleting an object seems to be widely accepted, yet it implied the problem of different drawing styles when using unistrokes. To prevent unnecessary stalling of the project's progress, two versions were defined that seemed feasible. To reiterate, since the goal was not to devise a system that is ready to use out of the box and optimized for usability, less thought was put into issues such as proper definition of the gestures and identification of the most frequently performed tasks.

5.2 Selection of Tools

This section describes which tools were chosen to implement the system and the reason for choosing them. The most important step after understanding the fundamentals of HMMs was to find a suitable library for Hidden Markov Models. It was clear that it would be impossible to implement the theory independently within the scope of this thesis. Unfortunately, the selection of available libraries is fairly small.

After a guest lecture at the Bonn-Rhein-Sieg University of Applied Sciences, Dr. Volker Krüger of the Aalborg University recommended the Hidden Markov Model Toolbox for MATLAB [62] as an excellent modeling tool for HMMs. The toolbox supports every major operation on HMMs. However, as a result of the project's specifications, it would be difficult to implement the system within a MATLAB environment. Therefore, the utilization of this Toolbox was deemed impossible.

The Hidden Markov Toolkit (HTK) by the Cambridge University Engineering Department (CUED) [15] is basically a set of tools for building and manipulating Hidden Markov Models. It is applied but not restricted to speech recognition and it is widely used. However, the recognition system had to be written in C/C++

or Python since AVANGO NG (see Section 3.4) is based on these languages. The HTK also includes source and header files, but it does not provide an API to ensure convenient usage.

The Probabilistic Networks Library (PNL) [38] by Intel includes a large variety of algorithms for computational inference and learning. As a part of Intel's Open Source Libraries, like the well-known Open Source Computer Vision Library (openCV), it is meant to provide a freely accessible software package overcoming the disadvantages of other existing packages. The scale of the package made this library very interesting for the implementation of the recognition system. The multitude of algorithms made the package appear very complex, especially since Hidden Markov Models were just a very small subset. However, this was not a reason for disregarding it. Several attempts to compile the library within the same testing environment of the *VRGeo* Demonstrator failed. Investing more time might have solved the problem, but because of a very tight schedule the decision was made to test another alternative first.

When searching for an HMM library on the Internet the first result is the General Hidden Markov Model library (GHMM) [1] developed by the Algorithmics group at the Max Planck Institute for Molecular Genetics in Berlin. As the name suggests, other than the PNL, this library is specialized on HMMs only. Primarily, this simplified handling of the API. The library is written in C, including bindings for the Python scripting language, making it ideally suited for the project. Also included in the package are several examples dealing with typical problems of HMM applications. After writing a few examples using the Python bindings, it was obvious that the incredible simplicity of operations – such as building, saving, loading and training HMMs – would be a great advantage for the implementation process. Frequent development activities and online discussions indicated a very active community. This promised good support and further encouraged the use of this library. After reviewing the findings of the research for a suitable HMM software package, the GHMM library was deemed most appropriate. The most recent version, ghmm 0.8 beta 1, was used for the implementation.

Relatively early in the project, it was agreed that the data used for recognition should preferably be reduced in dimensionality. The chosen solution was to perform a *principal components analysis* (see Section 5.4.2). Fortunately, finding a library to perform this step was unnecessary. Another project within the *VRGeo* project deals with the construction of *alpha shapes* with the help of the Computational Geometry Algorithms Library (CGAL) [17]. This library is also able to perform the principal components analysis and thus could be used for the recognition system as well.

5.3 AVANGO NG

The framework used for this thesis is called AVANGO® NG [47] which is the next generation version of the open source VR/AR framework AVANGO® [81] developed at the *Fraunhofer Institute for Intelligent Analysis and Information Systems IAIS*.

This section will focus on the main aspects of this framework and mention the major improvements in the next generation version.

The most important characteristics of AVANGO NG are the scene graph, a *field* concept similar to *Open Inventor* [86] and a scripting interface that allows rapid prototyping. Other components are a network layer for application distribution, a flexible display setup with abstraction of output devices and a device daemon to connect input devices and tracking systems. AVANGO was based on the proprietary *OpenGL Performer* scene graph toolkit for scene representation and management. For AVANGO NG this scene graph library was replaced by *OpenSceneGraph* [65], since the development of *Performer* has stalled. Additionally, the dependence on a commercial library might have caused potential users to disregard AVANGO for their work. Despite this change, the scene graph concept stayed the same. The scene is described by a hierarchy of nodes called the scene graph. Nodes can hold a variety of properties, e.g. geometry, transformations, grouping information, lights or material properties. In AVANGO NG these nodes are called *field container* and contain a collection of arbitrary *fields* which in turn encapsulate the states of AVANGO NG objects. Both fields and field containers support a streaming interface and thus network distribution. This allows running AVANGO NG applications on distant machines or in cluster systems. To propagate events within the framework, *fields* of the same type can be connected. Using these *field connections* creates an additional data-flow graph which is orthogonal to the scene graph and allows logical relations between nodes that could not be expressed with the scene graph alone. Once the value of a field changes, all connected fields will be updated before the next frame is rendered.

The majority of AVANGO NG is implemented in *C++*, but it also offers a scripting interface. In AVANGO *Scheme* was used as the scripting language. In the next generation, *Python* replaced *Scheme*, mainly because it is well-known and more widely used. Whole applications can be implemented using the scripting interface which allows rapid prototyping. One advantage, besides the time issue, is that a class implemented in *Python* does not behave any different from a class implemented in *C++*. Thus, a scripted field container can easily be reimplemented in *C++* if necessary. *Pythons* native support of object-oriented programming also simplifies application development.

Besides replacing two major dependencies, the framework as a whole was refactored to reduce inter-module dependencies and overcome flaws discovered in AVANGO. The core of AVANGO NG was decoupled from the rest of the framework allowing the encapsulation of the scene graph and an improved component model. Sometimes it is necessary to add additional information to a node to perform certain operations such as dragging an object. For the drag operation, the application needs to determine if a selected object is draggable by querying a corresponding *field*. These attributes are called *external fields* since they are not used by the node itself. Adding new *external fields* in AVANGO required changing the base classes because extra information was hardwired into them. The new model allows adding *fields* at run-time and then use them as *external fields*, thereby increasing flexibility. As another example, a large number of special *field containers* had to be defined previously to ensure correct eval-

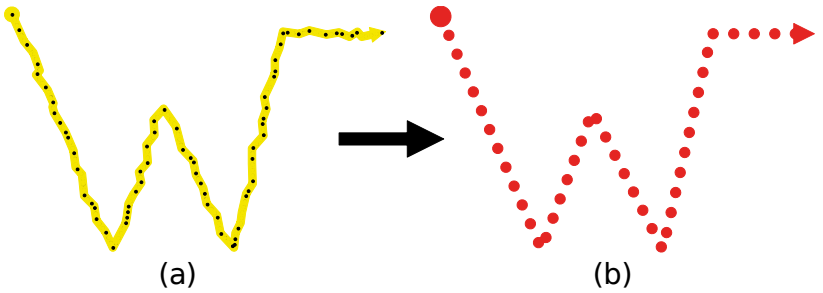


Figure 5.4: 2D example of the thresholding process to achieve equidistant sampling. (a) An exemplified input of too high precision. (b) The input after resampling.

uation order. As a result of redefining the evaluation order, special classes are no longer necessary. These are just a few improvements of AVANGO NG. Furthermore, the framework is under constant development as it is used for several current research projects at the *Fraunhofer IAIS*.

5.4 Preprocessing

As mentioned in the previous chapter, the decision was to use *discrete* Hidden Markov Models. Therefore, the target data has to be preprocessed to be appropriate for the training and recognition process. The individual steps are illustrated in this section.

5.4.1 Resampling

At first, the data is resampled to ensure time invariance. As mentioned before the tracking system used for the *VRGeo* Demonstrator delivers 60 sample vectors per second regardless of user behavior. This creates two major problems that had to be considered. First of all, because of the sampling rate and the highly precise tracking, even a user with a fairly steady hand will cause a jitter in the motion of a gesture. Therefore the variance of a repeated motion is increased unintentionally complicating the recognition. Secondly, the representation of a gesture should only depend on the order of motion trajectories, but be somewhat independent of time. This means the recognition of a gesture should not be affected by the time it takes to perform it. For example, a pause introduced at the beginning or the end of the gesture should not affect the result. Both problems were solved by a thresholding process. The incoming stream of 3D points is stored only if the distance of the current point to the last valid one exceeds a certain threshold. This step acts as a sort of low-pass filter eliminating high frequencies in the motion trajectories. At the same time it ensures an equidistant sampling. It is performed as soon as the gesture button on the input device is pressed and throughout the time it is held down. In Figure 5.4 a 2D example is illustrated to show the effect of this processing step. The number of sampled points per gesture

can now only vary with the size of the gesture. This issue was disregarded, however, because initial experiments in which gestures were resampled to have the same number of points showed no significant improvements.

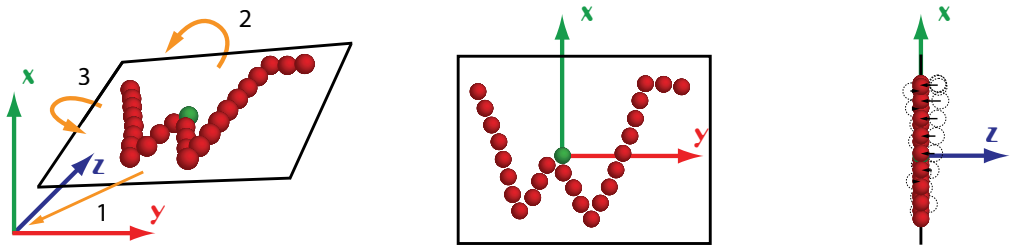
5.4.2 Dimensional Reduction

Once the gesture button is released, i.e. the gesture is complete, the result of the first preprocessing step is a list of coordinates in three-dimensional space. However, it was determined that the gestures to be recognized would be symbols such as characters or simple geometric shapes, e.g. a circle, L or W. Considering the nature of such symbols it should be obvious to the reader that they are only two-dimensional. Accordingly, it was a step to reduce the dimensionality of the resulting set of 3D points to two dimensions. The most simple solution would have been to completely disregard one axis, presumably the depth axis. Usually this will be the z-axis of a Cartesian coordinate system. If a user performs the gesture in a manner such that each sampled point lies within a single xy-plane, the mentioned technique will always capture the exact gesture. If a user only deviates slightly from this constraint, it will probably still yield fairly good results. However, for this application a user will work within in a virtual environment. Therefore, no assumption can be made about the direction in which a user points the input device while gesturing. This means that the depth axis of the gesture is not necessarily consistent with the depth axis of the application. Accordingly, a way had to be found to determine the best fitting plane in 3D space in which the gesture was performed.

The most intuitive was to run a *principal components analysis* (PCA). The PCA is usually used in data analysis to reduce complexity of multivariate data sets (see for example [34] or [36]). In most cases, the data is reduced to 2D for scatter plot visualization or to 1D for ranking. Assume the original data is a set of statistical variables $X_1, X_2, \ldots, X_n$ which are normally correlated with each other. For instance these could be test scores of students in different subjects. The idea is to decompose the data to an orthogonal set of linear combinations of all characteristics called *principal components*. Since the linear combinations are orthogonal, the resulting components will be uncorrelated. The combinations can be found by *eigenvalue* decomposition of the empirical $n \times n$ variance-covariance matrix of the data. It can be shown that the *eigenvector* corresponding to the greatest *eigenvalue* is the linear combination accounting for the greatest variance in the whole data set. This means that if a is this eigenvector, then

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \tag{5.1}$$

is the first principal component that best describes the data set. With the second largest eigenvalue, the second principal component can be found which describes the second largest variance. Continuing in this manner will lead to n principal components which all together completely describe the data. However, in most cases the components less important for the total variance are neglected. How many principal components are necessary to describe the data with satisfactory precision depends on



(a) (b) (c)

Figure 5.5: An example gesture in 3D space (a) is moved (1) and rotated (2 and 3) to be centered at the origin and aligned with the xy-plane (b) using the best fitting plane and the centroid calculated by the CGAL library. The z-coordinate of the transformed points is dropped to project them onto the xy-plane (c) which yields a two-dimensional set of sample points. The original position of the transformed points is indicated by the dotted circles.

the data and the used decision criterion. Common criteria are the *scree plot*, Kaisers's or Jolliffe's criterion (see for example [34, pp. 120–121]). For this thesis none of the criteria were important since the goal of the PCA was clear already. Unfortunately, the CGAL library used for the project does not return the linear combinations. However, it implements PCA to, among other things, determine the best fitting 3D plane and the centroid to a set of three-dimensional points. This result was ideal since finding the best fitting plane was the original intention of the dimensional reduction. For the recognition system, the centroid is used to center the gesture at the origin of 3D space. Then the sample points of the gesture are projected onto the plane calculated by the CGAL library after the plane was rotated so that it equals the xy-plane. The entire process is illustrated in Figure 5.5. [34, pp. 107–129][36]

5.4.3 Discretization

Now the sample points are two-dimensional but the coordinates are still continuous values. In order to pass the data to a discrete Hidden Markov Model in the form of an observation sequence the values need to be discrete. At the same time, the 2D points must be parametrized in such a way that they represent a gesture adequately. As presented in Chapter 2, several possibilities can be found in the literature. The approach taken here is a simple form of vector quantization. First, the position vectors of each sample are used to generate a trajectory, i.e. a chain of direction vectors (illustrated in Figure 5.6 (b)). Then the angle between each direction vector and the x-axis is used to determine a number between 1 and 16. This number represents an index into an manually created codebook of prototype vectors. Thus each direction vector can be represented by an integer value. The whole gesture is represented by a list of these values and can now be used to train a Hidden Markov Model or to query one for classification.

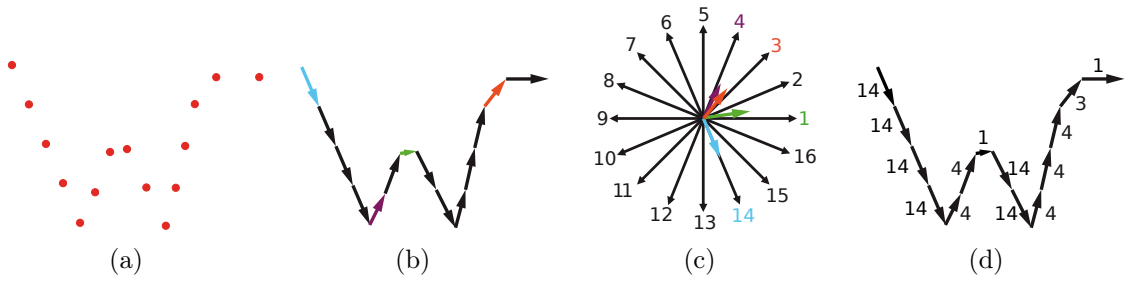


Figure 5.6: Illustration of the vector quantization process. (a) A set of 2D sample points. (b) Trajectory of direction vectors derived from the sample points. (c) Direction vectors used for quantization. The mapping of directions from the gesture to the quantization vectors is indicated. (d) The final observation sequence is a list of numbers that correspond to 1 of 16 directions. For this example the list would be: 14, 14, 14, 14, 4, 4, 1, 14, 14, 4, 4, 4, 3, 1.

5.5 HMM Initialization

This section describes how the structure and parameters of the Hidden Markov Models for the recognition system were determined. A simple decision was the overall structure of the recognition system. The approach to use one model to represent one gesture could be found in many elaborations on gesture recognition systems and was therefore adopted for this project. Despite a multitude of publications on HMMs, the rest of this part of work was one of the most challenging. Most of these publications describe the theory of HMMs in adequate depth and the applied preprocessing in detail. However, virtually every work conceals details about the choice of parameters used for the HMMs. Though HMMs are widely used, it appears that in large part finding the "right" number of hidden states or symbols is still guess work. Fink mentions that expert knowledge of the particular domain is needed to define a model structure [29, p. 81]. This means that someone with inside knowledge of the process that is to be modeled with HMMs, specifies parameters like number of states or the distribution of the emissions. This implies, however, that the expert has knowledge of his field of work and at the same time is familiar with the concept of Hidden Markov Models. Euler is a little more helpful by noting that the quality of the model can be improved by adjusting the number of states to the number of sounds in a word when dealing with speech recognition [27, p. 100]. In other papers, models with a number of states between 1 and 20 can be found. However reasons for choosing those numbers do not appear.

In parallel, a reasonable distribution of the emission symbols had to be determined, since information on this topic could not be found either. The first idea was to instinctively choose a number of states and assume a uniform distribution for all emissions, state initials and state transitions. The final models were to be shaped by the training process. Trials with 5 and 10 states respectively were not encouraging. Further research revealed that several systems use *specialized* models, i.e. each gesture is analyzed and the corresponding model is defined manually according to this anal-

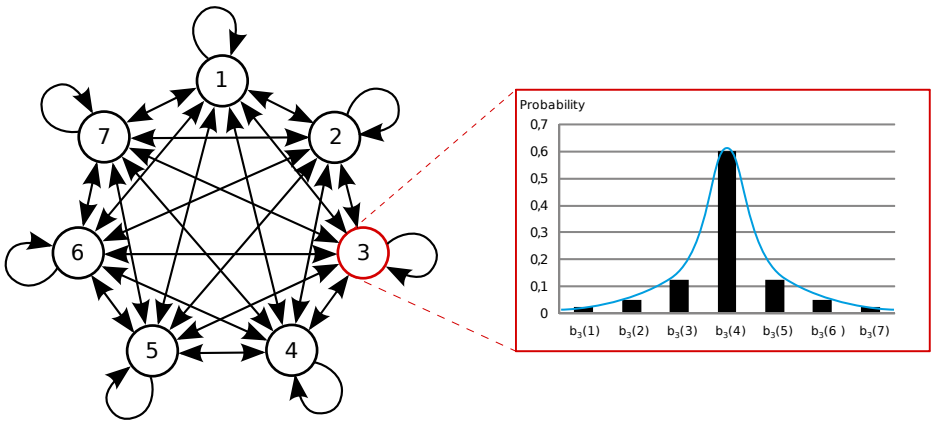


Figure 5.7: This figure displays a simplified structure of the HMM prototype used for the gesture recognition system with 7 instead of 16 output symbols. All state transitions of the ergodic model have the equal probability of $\frac{1}{number\,of\,states}$. Each state is equally likely to be the initial state with the same probability used for the state transitions. The distribution of the emissions is indicated for state 3 showing the Gaussian-like distribution.

ysis. Unfortunately, because of the concept of the recognition system for the *VRGeo* Demonstrator, this plausible approach could not be used. From the beginning it was clear that the intention was to give a user the chance to extend the system by adding more gestures. This would be impossible if a user needed to tinker with the parameter settings for each new model. The general approach had to be retained, which meant that all models had to be equal before the training. Some sort of prototype HMM had to be defined which would specify automatically based on the training.

Further experiments and research did not yield any satisfactory results. The main reason was the abstract correlation between the number of observables and the number of states. Usually the number of observables is much higher than the number of states. From this it is hard to understand the coherence between state transition probabilities and emission probability distributions. In other words, even though it is *hidden*, details of the internal process of a model must be known to the one defining it. After attempts to find a satisfactory solution to this problem were unsuccessful, a different approach was chosen. Instead of trying to find the relation between the observable and the hidden part of the model, the hidden part was disclosed. To achieve this, each state was defined to prefer one possible output symbol. This means the prototype model includes as many states as observables. Since 16 different direction vectors were chosen as possible symbols, the initial model is composed of 16 states. To account for the variability in motion of a gesture, the distribution for the emissions was modeled to represent a Gaussian-like distribution. Therefore, each state s_i of the prototype model prefers vector i with a probability of 60%. The directly preceding or following vectors $((i \pm 1) \mod 16 + 1)$ are emitted with a probability of 12.5%, $i - 2$ or $i + 2$ with a probability of 5%. The probability of emitting one of the remaining vectors is 5%. For clarification, output probabilities for state s_3 of a simplified model are illustrated in Figure 5.7.

In speech recognition it is common to use left-right or Bakis models (see Section 4.1) because they effectively model signals with temporal order. The same idea applies to gesture recognition. However, to effectively use these topologies, the general path through the model must be known in advance, including its start and final state. In this case it would require knowledge about the general movement of the gesture which would have been used for defining specialized HMMs. However, because of the general approach explained previously, this was ruled impossible. Hence, the generic ergodic topology was used as seen in Figure 5.7. Since the direction at the beginning of the gesture is totally arbitrary, any of the 16 states could be the first. Therefore, the initial state probabilities were uniformly distributed. The same applies to the transition probabilities. Both would then be adapted to a certain gesture by the training process. After collecting all ideas about the parameters and the topology, it was possible to construct a prototype HMM which would serve as a basis for all gesture models.

6 Training and Results

So far this thesis has covered the necessary theories and their application to practice. The result is a system that is capable of transforming raw tracking data to a sequence of discrete observation symbols. Such sequences could be used to train the described HMM prototype. In this chapter the concepts and processes involved in the training phase will be covered before discussing the results obtained while testing the system. However, it is worth mentioning that the evaluation presented here is not a thorough statistical analysis but rather a straightforward test using a set of prerecorded training samples. Besides presenting the test results, the introduction of an additional *noise* model is explained alongside its positive and negative consequences.

6.1 Training the Hidden Markov Models

The result of the ideas presented in the previous chapter was an HMM prototype. This prototype was now to be trained to represent a certain gesture. Similar to the number of states, information about the number of training samples differ considerably in the relevant literature. Quantities range from just two samples per gesture [42] to 100 [91]. As a result of extensive testing during development, a direct proportionality between the number of samples used for training and the recognition quality was observed, concurring with the results found in [57]. Consequently, for the gesture recognition system presented here 130 samples of each gesture were recorded from 4 different users, whereas one sample is one recorded repetition of a gesture. Of those 130 samples, 100 were used for training and the remaining 30 for testing. The recognition system was presented at the June meeting of the *VRGeo* Consortium to their members. Since no training module for customization had been planned, the system had to be user independent. Therefore, it was necessary to use samples of different users, because too many training sets of a single user would result in overfitting of the models. In that case the system would recognize gestures of users other than the one who trained the model with very low probability. The recording of samples took place within an environment similar to the *VRGeo* Demonstrator setup using the same input device and display system. This was important so as to avoid introducing unnecessary complications to the process due to unknown factors.

The training was performed off-line, i.e. without user interaction and before integration into the *VRGeo* Demonstrator. For each of the seven gestures, the General Hidden Markov Model library was used to create a prototype described in Section 5.5. Then the *Baum-Welch algorithm* was applied to reestimate the parameters of the prototype using the set of thirty preprocessed training sequences for each gesture. To

store the resulting model for later reference the GHMM library provides a function that is capable of writing all important parameters and properties to an XML type format which was then saved to the hard drive. A text file serving as a flat database was updated with information on the currently trained gesture, namely a reference name and the name of the model file. The results are seven XML files representing one HMM each and a database with seven entries. In this way the database can be modified easily with any text editor. Therefore, adding new gestures is very simple and excluding them from the system only requires the deletion of the corresponding two lines in the text file. Another advantage is that the model files themselves do not have to be deleted, so if a model is to be used again at a later time, no training is necessary. The name of the file and model just have to be reentered into the database. Although the construction of the database was performed off-line, it is implemented in a way that it can easily be reused for an on-line training module later on.

For recognition, the database and the model files were made accessible to the recognition module of the demonstrator. The module uses the database to create HMMs from the provided model files with the available GHMM library functions. After the user finished the gesture and the data preprocessed according to the procedures described in the previous chapter, the model which most likely produced the current emission sequence is determined by finding the maximum likelihood value. The GHMM uses the *forward algorithm* (see Section 4.2.1) to calculate the logarithmized probability $\log P(\boldsymbol{O}|\lambda)$ for each model. This representation is due to the computational issues mentioned in Section 4.3. The results are compared and the model which maximizes the probability is chosen to describe the currently observed gesture. Unlike the negative logarithmic scale introduced before, the results are not negated by the GHMM library. If the probability were negated, of course, the lowest result would have to be considered. Depending on many factors, e.g. sloppiness, type or especially size of the gesture and missing references it was impossible to determine at which point the calculated probability would legitimatize the recognition of a gesture. It is noticeable from the definition of the *forward algorithm* that the resulting probability will decrease monotonously with increasing size of the observation sequence. However, as mentioned in the previous chapter, resampling the sequences to achieve a constant length of n samples did not seem to attenuate the problem much, making it hard to find an adequate threshold. Actually, a sort of thresholding is already performed by the GHMM library. Using the *Python* bindings, probabilities that are too small or zero, i.e. non-observable, are returned as a value of "$-infinity$". Thus, setting a low enough threshold will only assure that none of the trained gestures was performed if the threshold was not exceeded after calculating the probabilities for all models, i.e. if every model returned "$-inf$". As soon as this is not the case, the choice of an arbitrary threshold becomes problematic. If it is too high, a correct gesture might be rejected. If it is too low, a wrong gesture might be recognized. The possibility of performing experiments to empirically determine a threshold was dismissed for two reasons. Firstly, the time available before the *VRGeo* June meeting was limited. Secondly, since the recognition system should eventually contain a training module for customized gestures, such an experiment would have made not much

sense. The threshold determined as a result for the current seven gestures might be totally worthless for a different set. As a consequence, an eighth model was added to the database. This model is simply the prototype model without training, which means all states are equally likely to be the initial state and all transition probabilities are identical. It describes a very general case, which was the intention of finding a prototype model. The idea was to use it to filter any *background noise* which does not qualify as a gesture, namely every arbitrary movement which is not one of the trained gestures.

6.2 Evaluation of the Recognition System

To test the performance and robustness of the implemented recognition system, the total number of 210 raw test samples were passed to the recognition module. The number of correct and false classifications were counted for each gesture to calculate a *recognition rate* and an *error rate*.

$$recognition\,rate = \frac{number\,of\,correct\,classifications}{total\,number\,of\,test\,samples} \cdot 100\,\% \tag{6.1}$$

$$error\,rate = \frac{number\,of\,incorrect\,classifications}{total\,number\,of\,test\,samples} \cdot 100\,\% \tag{6.2}$$

If a gesture was classified as noise, it was neither recognized correctly nor incorrectly. To investigate the proportional effect of recognition rate and the number of training samples, the system was retrained with an increasing number of samples starting at 20 and ending at 100. This was done in five steps and for each step the testing procedure was repeated. To ensure a flexible and efficient testing process, an additional module was implemented which automatizes the procedure of retraining the existing models. The recognition rate is particularly important when judging the quality of the devised system. A study on handwriting recognition by Mary LaLomia [48] identified a rate of 97% as acceptable. Since the process of handwriting is somewhat similar to the gesture input for this recognition system, similar rates will presumably have to be achieved to ensure user acceptance. Others suggest recognition software should have recognition rates of nearly 100% to not be discarded by a user [42, 54]. As mentioned before, the goal of this thesis was to create a proof of concept to show that gesture control can be introduced successfully to the *VRGeo* prototype software. Accordingly, achieving recognition rates of 97% or higher was not expected. Suprisingly, analyzing the results of the system showed very good recognition rates, up to 98.57% when training the models with the highest number of training samples available. As seen in Figure 6.1, the rates were relatively high for any tested number of samples. The monotonous increase of rates also show that the proportional relation between recognition rate and number of training samples holds true for the implemented system. Thus, the remaining 30 samples used for testing could be incorporated into the training to further improve the final system.

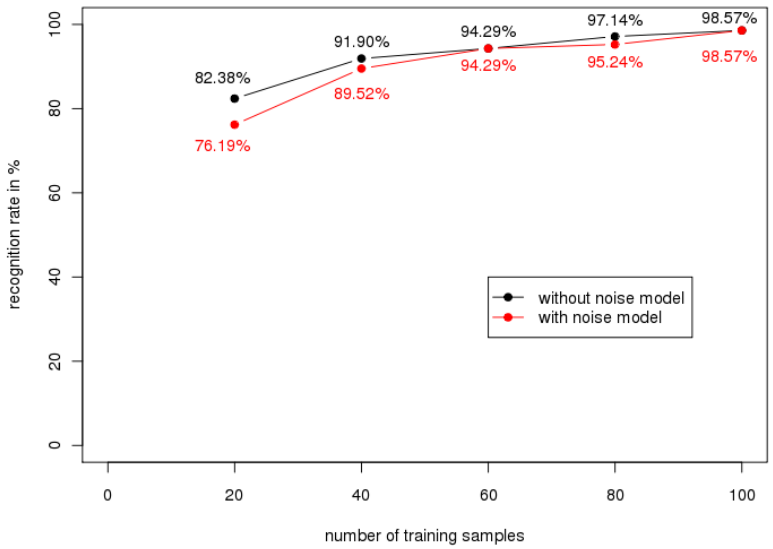


Figure 6.1: This plot illustrates the percentage of correctly recognized gestures of a total of 210 test samples of the trained gestures with (red) and without (black) the additional noise model.

Even more surprising was the astonishing 0% error rate for all five configurations. Everytime a gesture was not recognized correctly it was classified as noise. While this result approved the usage of the noise model, it also implied the question whether the additional model worked too well by intercepting too many movements. To trace this thought, the same experiment was repeated excluding the noise model from the recognition process, i.e. as soon as at least one model does not return "$-inf$" a gesture will be recognized. Indeed, the obtained results indicate that in a few cases the noise model falsely intercepts a gesture that would otherwise have been classified correctly by one of the trained models. In the experiments, this led to slightly higher recognition rates when training with 20, 40 and 80 samples. However, the main purpose of the noise model was to solve the thresholding problem and prevent the recognition of wrong gestures. Hence, two more experiments were conducted to test whether the addition of the noise model was justifiable. For this purpose a set of seven new gestures were defined as shown in Figure 6.2. Those new symbols are similar to the ones that were used to train the HMMs. They were designed so that a user accidentally gesturing these "wrong" symbols while performing a task in the demonstrator would easily be imaginable. Since rotational invariance is not yet a feature of the system, it was obvious to choose the circles and start at the bottom instead of starting at the top. Due to the similarity of the symbols to the Roman letters we use in our everyday life, it is plausible that a user could try to draw a symbol in cursive instead of using the printed version. The L and W symbols are ideal candidates for such a mix up. Similarly, those two symbols were used to introduce a case where some

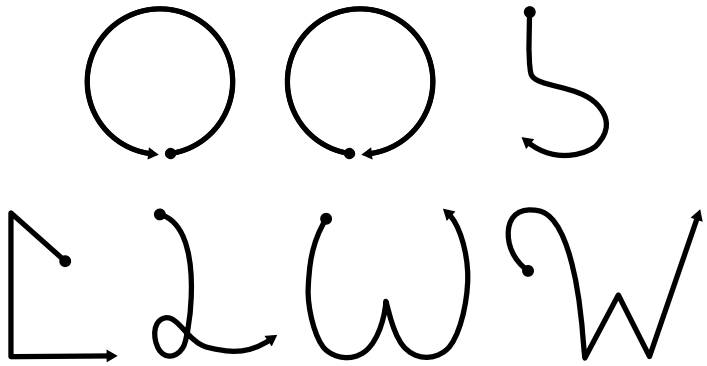


Figure 6.2: Symbols that were used to test the effects of the additional HMM meant to filter background noise. The symbols were chosen because they are similar to the original gestures and could easily be used by the user on accident.

pre-movement of the gesture accidentally became part of the input. Lastly, a sloppy version of an S was also considered. The X symbols were not considered because they were recognized to such a degree that it was virtually impossible to gesture an X-like symbol that would be misclassified. The similar gestures were recorded and passed to the module for recognition. Again a gesture was not classified at all, only if all models returned "$-inf$". The results displayed in Figure 6.3 show that just as in the case of the recognition rate, the error rate would increase with the number of training samples. After training the models with only 20 training samples, the error rate was only 6.67%, because most of the time no gesture was recognized. However, when using 100 training samples as much as 145 of 210 test samples were misclassified. Compared to that, the maximum error rate was only 4.28% for 40 training samples while allowing the classification as noise. With all 100 training samples, the error rate was 2.38% which is 66.67% lower than without the noise model. As a conclusion, the advantage of significantly reducing the misclassification of "wrong" gestures with the noise model clearly surpasses the effect of rejecting slightly more correct gestures.

Despite the promising results of the experiments, one problem could not be solved to this date. Due to the structure of the HMMs and the defined gestures used, the system also recognizes partial gestures. Partial gesture input could be the result of a user deciding to discontinue a gesture after starting it or it could be an arbitrary motion that was recorded on accident. After the training process, the initial state probabilities corresponding to the first general direction of a gesture will be very high, all others will be quite low or even zero. The trained model will also have high state transition probabilities from one state to itself and to the one that corresponds to the following direction. Obviously, the transition probability for the state corresponding to the last direction will only be high for a transition to itself. Now, as mentioned before, the probability calculated by the *forward algorithm* will decrease proportional to the length of the observation sequence. Therefore, any motion trajectory that starts out similar to one of the trained gestures will always result in a higher likelihood than gesturing the entire symbol. Another problem is that the transitions from one state to itself will usually be the highest, especially in the case of symbols containing straight

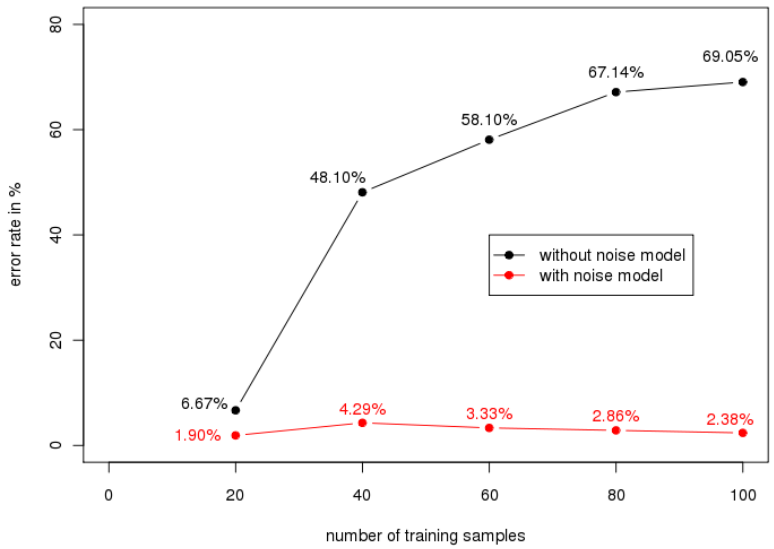


Figure 6.3: This image shows the significant differences in classification errors when utilizing the additional noise model (red) and when excluding it (black) while trying to recognize the set of "wrong" gestures.

lines, to ensure that the model stays in that state long enough. If the gestures are resampled to be of a constant size, gestures that consist of only one direction existent in a trained symbol will still yield high results. These cases should ideally be intercepted by the noise model. However, since the obtained result of such partial gestures will be higher than the one from a whole gesture, it will usually also be higher than the likelihood calculated from the noise model. Every attempt to eliminate this problem either failed or put constraints on the system that erased the general structure. For this reason, this issue remains to be dealt with later on (see Chapter 7).

The gesture recognition system was presented as part of the *VRGeo* Demonstrator to the participants of the 2008 *VRGeo* June meeting. Everybody was given the chance to test the new feature themselves. Although the success of using the gesture control varied from person to person, the overall feedback was very positive. Many interesting ideas were submitted to further integrate the gesture control into the demonstrator. As suspected before the meeting, one of the main suggestions was the creation of a user interface for training custom gestures. Fortunately, the current implementation should easily allow an adaption to include the requested interface. The consortium also agreed upon limiting the number of possible functions to be mapped to customized gestures, since it is always important to keep the memory load of users as low as possible when designing interfaces. Whenever gestures are performed too rapidly even for the deployed tracking system, interpolation problems are likely to occur, e.g. resulting in gestures as the sloppy S seen in Figure 6.2. To avoid this problem, it was suggested to combine the tracking with the data obtained by the built-in accelerometer

of the utilized 6-DOF input device. Another predictable request was the integration of gesture control and a speech control system. A system applying speech recognition was implemented and investigated independently from this thesis. Both interaction modalities are very natural ways of communication and complement each other. In fact one of the first interfaces to utilize speech and gesture input was the "Put That There" system developed by Bolt [9] dating as far back as 1980. Since then the promising idea of combining both modalities have been the subject of many studies [5, 10, 44, 49, 85].

7 Conclusion and Future Work

In this thesis the development of a gesture control interface for virtual environments based on Hidden Markov Models was presented. The design focused on understanding and applying the theory of HMMs to obtain a working system that could serve as a basis for future development. At the same time it was important to devise a system that justifies further research. For this reason, a fluent integration of the recognition software into an existing workflow within an VR application was emphasized. To assure user acceptance, visual feedback was included to improve the understanding of the process and the results produced by the recognition system. A trail of spheres along the path of the symbol while gesturing allows the user to immediately see the input. The results are presented with the help of the three traffic light colors, so anybody is able to easily understand the chosen color coding theme. The trail of spheres will turn red if recognition failed, yellow if a gesture was recognized but no according object could be referenced, and green if the process succeeded. The underlying concepts of the system were described as well to allow the reader to retrace the thoughts and ideas that led to the current interface. The crucial preprocessing phase was then covered in detail. It was elaborated how the raw tracking data is converted to assure an equidistant and time-invariant sampling which is the basis for a dimensional reduction using *principal components analysis*. The subsequent simple vector quantization produces the discrete observation symbols used to train and recognize gestures. The crucial part of initializing the parameters of the HMMs used for training were discussed to explain how this common problem was solved within the context of this thesis. The process of training and recognition was realized using the General Hidden Markov Model library in accordance to the presented theory of the HMM. Further, it was assured that the devised recognition software in its current form could easily be adapted for use in other applications. The planned integration into the introduced VR application for interactive exploration of seismic data and the presentation at the *VRGeo* June meeting was used to evaluate the recognition system. The new interaction technique was tested by VR and geophysics experts along with other new features over a period of several hours. The feedback delivered by the participants and the results obtained from a separate study were both highly positive. The recognition rates obtained while testing the system were unexpectedly high. At the same time, the deployed HMM to filter out background noise kept misclassifications at a very low rate. The goal was to create a system that would serve as a proof of concept. Overall the implemented gesture control interface by far exceeded the required results.

Despite the encouraging feedback, there are still many ideas for further improvements of the system, the foremost being the integration of a training module to allow

customized gestures. This addition, as well as the the combination of gesture and speech input, are part of the current research agenda for the *VRGeo* December meeting in 2008. The main problem to be dealt with when developing a training module is the number of training samples. It is impractical to ask a user to record 100 training samples for a gesture before it can be used for system control. Kela et al. [42] introduced noise to copies of the original training data achieving good recognition rates with only two training samples (see Chapter 2). Using a similar approach could be the basis for an acceptable training interface.

One of the major issues remains the problem of recognizing partial gestures described in the previous chapter. In speech recognition, it is a common practice to use one HMM for each phoneme – the smallest sound in a spoken language with a distinguishable meaning. A similar approach could be adapted by describing certain partial gestures with one HMM. These HMMs could then be combined to form one whole gesture, avoiding the problem with partial gestures. Yet another concept that could be borrowed from speech recognition is the use of Bakis HMMs. As described in Section 4.1, this topology is not deployed in the current system because it is not general enough to allow the addition of arbitrary symbols. However, having a final state would be a simple solution to prevent the recognition of partial gestures. Hence, a possibility would be to keep the current HMM prototype but further process a trained model through an algorithm to extract a corresponding Bakis model. This could not only solve the issue of partial gestures, but also improve recognition results.

Integrating these ideas into the developed recognition software will show whether the desired improvements will be achieved. The promising results of the current version indicate that the additional modality can be a valuable supplement to other interaction techniques. Thus, investing more time and effort to perfect the implemented system will definitively lead to further advances within the *VRGeo* project.

A Media Content

The following material can be found on the enclosed CD:

- Abstract
- Bachelor's Thesis
- References
  - Internet Screenshots
  - Publications
- Software
  - Data
    - HMM XML Files
    - Test Data
      - Trained Gestures
      - Wrong Gestures
    - Training Data (grouped by number of samples)
      - 20
      - 40
      - 60
      - 80
      - 100
  - Source Code
    - C++ Files
    - Python Modules
- Themenspezifikation

Bibliography

[1] Algorithmics group at the Max Planck Institute for Molecular Genetics. GHMM Library. `http://www.ghmm.org`, 2008. [Online; accessed August-06-2008].

[2] Behr, J. *Avalon: Ein skalierbares Rahmensystem für dynamische Mixed-Reality Anwendungen*. PhD thesis, TU Darmstadt, 2006.

[3] Bertelson, P., and Aschersleben, G. Automatic visual bias of perceived auditory location. *Psychonomic Bulletin & Review 5*, 3 (1998), 482–489.

[4] Bierbaum, A., Just, C., Hartling, P., Meinert, K., Baker, A., and Cruz-Neira, C. VR Juggler: A virtual platform for virtual reality application development. *Proceedings of the IEEE Virtual Reality* (2001), 89–96.

[5] Billinghurst, M., Savage, J., Oppenheimer, P., and Edmond, C. The expert surgical assistant: An intelligent virtual environment with multimodal input. In *Proceedings of Medicine Meets Virtual Reality IV* (1995).

[6] Birn, J. *Digital Lighting and Rendering*, 2nd ed. New Riders, 2006.

[7] Blach, R., Landauer, J., Rösch, A., and Simon, A. A highly flexible virtual reality system. *Future Generation Computer Systems 14*, 3–4 (1998), 167–178.

[8] Blickenstorfer, C. H. Graffiti: Wow! *Pen Computing Magazine January* (1995), 30–31.

[9] Bolt, R. A. "Put-that-there": Voice and gesture at the graphics interface. *SIGGRAPH Comput. Graph. 14*, 3 (1980), 262–270.

[10] Bolt, R. A., and Herranz, E. Two-handed gesture in multi-modal natural dialog. In *UIST '92: Proceedings of the 5th annual ACM symposium on User interface software and technology* (New York, NY, USA, 1992), ACM, pp. 7–14.

[11] Bowman, D. A., Kruijff, E., LaViola, J. J., and Poupyrev, I. *3D User Interfaces: Theory and Practice*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2005.

[12] Burdea, G. C., and Coiffet, P. *Virtual Reality Technology*, 2nd ed. Wiley-IEEE Press, July 2003.

[13] Buxton, W., Sniderman, R., Reeves, W., Patel, S., and Baecker, R. The evolution of the SSSP score editing tools. *Computer Music Journal 3(4)* (1979), 14–25.

[14] Bystrom, K.-E., Barfield, W., and Hendrix, C. A conceptual model of the sense of presence in virtual environments. *Presence: Teleoper. Virtual Environ. 8*, 2 (1999), 241–244.

[15] Cambridge University Engineering Department. HTK3. `http://htk.eng.cam.ac.uk/`, 2007. [Online; accessed August-13-2008].

[16] Castellucci, S. J., and MacKenzie, I. S. Graffiti vs. unistrokes: an empirical comparison. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 2008), ACM, pp. 305–308.

[17] Cgal. Computational Geometry Algorithms Library. `http://www.cgal.org`, 2008. [Online; accessed August-06-2008].

[18] Colman, A. M. *A Dictionary of Psychology.* Oxford University Press, 2001.

[19] Cruz-Neira, C., Sandin, D. J., and DeFanti, T. A. 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* (New York, NY, USA, 1993), ACM, pp. 135–142.

[20] Czernuszenko, M., Pape, D., Sandin, D., DeFanti, T., Dawe, G. L., and Brown, M. D. The ImmersaDesk and Infinity Wall projection-based virtual reality displays. *SIGGRAPH Comput. Graph. 31*, 2 (1997), 46–49.

[21] d'Angelo, D. Interaction techniques for assembly based modeling in virtual environments. Master's thesis, Bonn-Rhein-Sieg University of Applied Sciences, 2008.

[22] Dressler, A. Benutzergerechte Menügestaltung für Virtuelle Umgebungen im Expertenkontext. Master's thesis, Technische Universität Ilmenau, 2007.

[23] Eisenstein, J., Ghandeharizadeh, S., Golubchik, L., Shahabi, C., Yan, D., and Zimmermann, R. Device independence and extensibility in gesture recognition. In *VR '03: Proceedings of the IEEE Virtual Reality 2003* (Washington, DC, USA, 2003), IEEE Computer Society, p. 207.

[24] El-Yacoubi, A., Sabourin, R., Suen, C. Y., and Gilloux, M. An HMM-based approach for off-line unconstrained handwritten word modeling and recognition. *IEEE Trans. Pattern Anal. Mach. Intell. 21*, 8 (1999), 752–760.

[25] Elen, R. Ambisonics: The surround alternative. `http://www.ambisonic.net/pdf/ambidvd2001.pdf`. [Online; accessed September-08-2008].

[26] Elliman, D., and Connor, P. Orientation and scale invariant symbol recognition using a Hidden Markov Model. *Image Processing and its Applications, 1992., International Conference on* (April 7–9 1992), 331–334.

[27] Euler, S. *Grundkurs Spracherkennung*. Vieweg, 2006.

[28] Fels, S. S., and Hinton, G. E. Glove-Talk: A neural network interface between a data-glove and a speech synthesizer. *IEEE Transactions on Neural Networks 4*, 1 (January 1993), 2–8.

[29] Fink, G. A. *Mustererkennung mit Markov-Modellen*. Leitfäden der Informatik. B. G. Teubner, Stuttgart – Leipzig – Wiesbaden, 2003.

[30] Forney, G.D., J. The viterbi algorithm. *Proceedings of the IEEE 61*, 3 (March 1973), 268–278.

[31] Fraunhofer IAIS. VRGeo – Virtual reality research for the geosciences. `http://www.vrgeo.org`, 2008. [Online; accessed August-06-2008].

[32] Fröhlich, B., Blach, R., Stefani, O., Hochstrate, J., Hoffmann, J., Klüger, K., and Bues, M. Implementing multi-viewer stereo displays. In *WSCG (Full Papers)* (2005), pp. 139–146.

[33] Goldberg, D., and Richardson, C. Touch-typing with a stylus. In *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems* (New York, NY, USA, 1993), ACM, pp. 80–87.

[34] Handl, A. *Multivariate Analysemethoden: Theorie und Praxis multivariater Verfahren unter besonderer Berücksichtigung von S-PLUS*. Springer-Verlag, 2002.

[35] Hetmann, F., Herpers, R., and Heiden, W. The Immersion Square – immersive VR with standard components. In *Proc. Virtual Environment on a PC Cluster Workshop* (Protvino, Russia, 2002).

[36] Hotelling, H. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology 24* (1933), 417–441.

[37] Huang, X. Phoneme classification using semicontinuous Hidden Markov Models. *IEEE Transactions on Signal Processing 40*, 5 (May 1992), 1062–1067.

[38] Intel Corporation. Intel's Open-Source Probabilistic Networks Library (PNL). `http://www.intel.com/technology/computing/pnl/`. [Online; accessed August-08-2008].

[39] Jorke, H., and Fritz, M. INFITEC – A new stereoscopic visualisation tool by wavelength multiplex imaging. In *Proceedings Electronic Displays* (Wiesbaden, September 2003).

[40] Juang, B.-H., and Rabiner, L. The segmental k-means algorithm for estimating parameters of Hidden Markov Models. *Acoustics, Speech and Signal Processing, IEEE Transactions on 38*, 9 (September 1990), 1639–1641.

[41] Kay, S. Can detectability be improved by adding noise? *Signal Processing Letters, IEEE 7(1)* (January 2000), 8–10.

[42] Kela, J., Korpipää, P., Mäntyjärvi, J., Kallio, S., Savino, G., Jozzo, L., and Marca, S. D. Accelerometer-based gesture control for a design environment. *Personal Ubiquitous Comput. 10*, 5 (July 2006), 285–299.

[43] Kim, G. *Designing Virtual Reality Systems: The Structured Approach*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[44] Koons, D. B., Sparrell, C. J., and Thorisson, K. R. Integrating simultaneous input from speech, gaze, and hand gestures. 257–276.

[45] Kraemer, A. Two speakers are better than 5.1. *Spectrum, IEEE 38*, 5 (May 2001), 70–74.

[46] Krüger, W., Bohn, C.-A., Fröhlich, B., Schüth, H., Strauss, W., and Wesche, G. The Responsive Workbench: A virtual work environment. *Computer 28*, 7 (1995), 42–48.

[47] Kuck, R., Wind, J., Riege, K., and Bogen, M. Improving the AVANGO VR/AR framework – lessons learned. In *5. Workshop Virtuelle und Erweiterte Realität der Fachgruppe VR/AR* (Magdeburg, 2008).

[48] LaLomia, M. User acceptance of handwritten recognition accuracy. In *CHI '94: Conference companion on Human factors in computing systems* (New York, NY, USA, 1994), ACM, pp. 107–108.

[49] LaViola, J. J. Msvt: A virtual reality-based multimodal scientific visualization tool. In *Proc. IASTED Int. Conf. on Computer Graphics and Imaging* (2000), pp. 1–7.

[50] Lee, C., and Xu, Y. Online, interactive learning of gestures for human/robot interfaces. In *1996 IEEE International conference on Robotics and Automation* (Pittsburgh, PA, 1996), vol. 4, pp. 2982–2987.

[51] Lee, K.-F., Hon, H.-W., and Reddy, R. An overview of the SPHINX speech recognition system. *IEEE Transactions on Acoustics, Speech and Signal Processing 38*, 1 (January 1990), 35–45. see also IEEE Transactions on Signal Processing.

[52] Linde, Y., Buzo, A., and Gray, R. An algorithm for vector quantizer design. *Communications, IEEE Transactions on [legacy, pre - 1988] 28*, 1 (Jan 1980), 84–95.

[53] Lipscomb, J. S. A trainable gesture recognizer. *Pattern Recognition 24*, 9 (1991), 895–907.

[54] MacKenzie, I. S., and Zhang, S. X. The immediate usability of Graffiti. In *Proceedings of the conference on Graphics interface '97* (Toronto, Ont., Canada, Canada, 1997), Canadian Information Processing Society, pp. 129–137.

[55] Mandel, T. *The Elements of User Interface Design.* John Wiley & Sons, Inc., 1997.

[56] Mannuss, F., and Hinkenjann, A. Independent scene element representation in the *basho* virtual environment framework. In *Workshop on virtual and Augmented Reality of teh GI-Fachgruppe VR/AR.* Shaker Verlag, Aachen, 2005.

[57] Mäntylä, V.-M. Discrete Hidden Markov Models with application to isolated user-dependent hand gesture recognition. VTT Publications 449, 2001.

[58] Marcel, S., Bernier, O., Viallet, J.-E., and Collobert, D. Hand gesture recognition using input-output Hidden Markov Models. In *FG '00: Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition 2000* (Washington, DC, USA, 2000), IEEE Computer Society, p. 456.

[59] Mine, M. R. Virtual environment interaction techniques. Tech. rep., UNC Chapel Hill CS Dept, 1995.

[60] Mitra, S., and Acharya, T. Gesture recognition: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 37*, 3 (May 2007), 311–324.

[61] Mori, S., Suen, C., and Yamamoto, K. Historical review of OCR research and development. *Proceedings of the IEEE 80*, 7 (Jul 1992), 1029–1058.

[62] Murphy, K. Hidden Markov Model (HMM) Tollbox for Matlab. `http://www.cs.ubc.ca/~murphyk/Software/HMM/hmm.html`, 2005. [Online; accessed August-06-2008].

[63] Nam, Y., and Wohn, K. Recognition of space-time hand-gestures using Hidden Markov Model, 1996. ACM Symposium on Virtual Reality Software and Technology, pages 51–58, Hong Kong, 1996.

[64] O'Hagan, R., Zelinsky, A., and Rougeaux, S. Visual gesture interfaces for virtual environments. *Interacting with Computers 14*, 3 (2002), 231–250.

[65] OpenSceneGraph. `http://www.openscenegraph.org/`. [Online; accessed August-26-2008].

[66] Pavlović, V. I., Sharma, R., and Huang, T. S. Gestural interface to a visual computing environment for molecular biologists. In *2nd International Conference on Automatic Face and Gesture Recognition (FG '96)* (Killington, VT, October 1996), IEEE Computer Society, pp. 30–35.

[67] Pavlovic, V. I., Sharma, R., and Huang, T. S. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Trans. Pattern Anal. Mach. Intell. 19*, 7 (1997), 677–695.

[68] Poddar, I., Sethi, Y., Ozyildiz, E., and Sharma, R. Toward natural gesture/speech HCI: A case study of weather narration, 1998. Proc. Workshops on Perceptual User Interfaces, pages 1–6, November, 1998.

[69] Pulkki, V. Virtual sound source positioning using vector base amplitude panning. *Journal of the Audio Engineering Society 45*, 6 (June 1997), 456–466.

[70] Rabiner, L. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE 77*, 2 (Feb 1989), 257–286.

[71] Riege, K., Holtkämper, T., Wesche, G., and Fröhlich, B. The bent pick ray: An extended pointing technique for multi-user interaction. *3D User Interfaces, 2006. 3DUI 2006. IEEE Symposium on* (March 2006), 62–65.

[72] Rubine, D. H. *The automatic recognition of gesture.* PhD thesis, Carnegie Mellon University, 1991.

[73] Sherman, W. R., and Craig, A. B. *Understanding Virtual Reality: Interface, Application, and Design.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[74] Simon, A., and Göbel, M. The i-Cone – A panoramic display system for virtual environments. In *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications* (Washington, DC, USA, 2002), IEEE Computer Society, p. 3.

[75] Simon, A., Smith, R. C., and Pawlicki, R. R. OmniStereo for panoramic virtual environment display systems. In *VR '04: Proceedings of the IEEE Virtual Reality 2004* (Washington, DC, USA, 2004), IEEE Computer Society, p. 67.

[76] Sproull, R. F. *Principles of Interactive Computer Graphics (2nd ed.).* McGraw-Hill, Inc., New York, NY, USA, 1979.

[77] Srivastava, A., Kundu, A., Sural, S., and Majumdar, A. Credit card fraud detection using Hidden Markov Model. *IEEE Transactions on Dependable and Secure Computing 5*, 1 (2008), 37–48.

[78] Stark, M., Kohler, M., and ZYKLOP, P. Video-based gesture recognition for human computer interaction. Tech. Rep. 593/1995, Universität Dortmund, Universität Dortmund, 44221 Dortmund, GERMANY., 1995.

[79] Susnik, R., Sodnik, J., Umek, A., and Tomazic, S. Spatial sound generation using HRTF created by the use of recursive filters. *EUROCON 2003. Computer as a Tool. The IEEE Region 8 1* (Sept. 2003), 449–453 vol.1.

[80] Theile, G. Wave field synthesis – A promising spatial audio rendering concept. In *Proc. of the 7th Int. Conference on Digital Audio Effects (DAFx'04)* (Naples, Italy, October 5-8 2004).

[81] Tramberend, H. Avocado: A distributed virtual reality framework. *Virtual Reality, 1999. Proceedings., IEEE* (Mar 1999), 14–21.

[82] Tuulari, E., and Ylisaukko-oja, A. SoapBox: A platform for ubiquitous computing research and applications. In *Pervasive '02: Proceedings of the First International Conference on Pervasive Computing* (London, UK, 2002), Springer-Verlag, pp. 125–138.

[83] Uchino, S., Abe, N., Tanaka, K., Yagi, T., Taki, H., and He, S. VR interaction in real-time between avatar with voice and gesture recognition system. In *AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 959–964.

[84] Vince, J. *Introduction to Virtual Reality.* Springer-Verlag, February 2004.

[85] Weimer, D., and Ganapathy, S. K. Interaction techniques using hand tracking and speech recognition. 109–126.

[86] Wernecke, J. *The Inventor Mentor: Programming Object-Oriented 3d Graphics with Open Inventor, Release 2.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[87] Wetzstein, G., Göllner, M., Beck, S., Weiszig, F., Derkau, S., Springer, J. P., and Fröhlich, B. HECTOR – Scripting-based VR system design. *SIGGRAPH '07: ACM SIGGRAPH 2007 posters* (2007), 143.

[88] Wilson, A. D., and Bobick, A. F. Using configuration states for the representation and recognition of gesture. Tech. Rep. 308, Massachusetts Institute of Technology, 1994.

[89] Wilson, A. D., and Bobick, A. F. Hidden Markov Models for modeling and recognizing gesture under variation. In *Hidden Markov models: applications in computer vision.* World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2002, pp. 123–160.

[90] Wozniewski, M., Settel, Z., and Cooperstock, J. R. A framework for immersive spatial audio performance. In *NIME '06: Proceedings of the 2006 conference on New interfaces for musical expression* (Paris, France, 2006), IRCAM — Centre Pompidou, pp. 144–149.

[91] Yang, J., and Xu, Y. Hidden Markov Model for gesture recognition. Tech. Rep. CMU-RI-TR-94-10, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 1994.

[92] Yang, J., Xu, Y., and Chen, M. Hidden Markov Model approach to skill learning and its application to telerobotics. Tech. Rep. CMU-RI-TR-93-01, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 1993.

[93] Zeltzer, D., Pioch, N., and Aviles, W. Training the officer of the deck. *Computer Graphics and Applications, IEEE 15*, 6 (Nov 1995), 6–9.

Declaration of Authenticity

I hereby declare, that the work presented in this thesis is solely my work and that to the best of my knowledge this work is original, except where indicated by references to other authors. No part of this thesis has been submitted for any other degree or diploma.

Sven Seele
Brandstr. 17, 53639 Königswinter