Pontifícia Universidade Católica
do Rio de Janeiro

Siniša Kolarić

Towards direct spatial manipulation of virtual 3D objects using visual tracking and gesture recognition of unmarked hands

MSc Thesis

Thesis presented to the post-graduate program in Computer Science of the Department of Computer Science, PUC-Rio as partial fullfillment of the requirements for the degree of Master in Computer Science.

Adviser : Prof. Marcelo Gattass
Co–Adviser: Prof. Alberto Barbosa Raposo

Rio de Janeiro
March 2008

Siniša Kolarić

Towards direct spatial manipulation of virtual 3D objects using visual tracking and gesture recognition of unmarked hands

Thesis presented to the post-graduate program in Computer Science of the Department of Computer Science, PUC-Rio as partial fullfillment of the requirements for the degree of Master in Computer Science. Approved by the following commision:

Rio de Janeiro, March 28, 2008

Siniša Kolarić

Siniša Kolarić received his BSc degree in mathematics with a minor in computer science from the University of Zagreb, Croatia. He also concurrently studied theoretical physics for three years at the same university. Later on he worked in academia and industry for Croatian, USA and German organizations. Since 2006 he has been a graduate student at PUC-Rio and a researcher at Tecgraf/PUC-Rio. His scientific interests include computer-aided design, computational geometry and topology, solid modeling, 3D user interfaces and real-time interactive rendering.

Bibliographic data

Kolarić, Siniša

Towards direct spatial manipulation of virtual 3D objects using visual tracking and gesture recognition of unmarked hands / Siniša Kolarić; adviser: Marcelo Gattass; co–adviser: Alberto Barbosa Raposo. — 2008.
117 f.: il. (col.) ; 30 cm

Dissertação (Mestrado em Informática) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2008.
Inclui bibliografia.

1. Informática — Teses. 2. Manipulação direta espacial de objetos virtuais 3D. 3. Realidade aumentada. 4. Realidade mista. 5. Dispositivos de entrada 3D. 6. Técnicas de interação 3D. 7. Visão por computador. 8. Detecção de mãos. 9. Rastreamento de mãos. 10. Reconhecimento de gestos manuais. I. Gattass, Marcelo. II. Raposo, Alberto Barbosa. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

Katarina Kolarić née Bubek (1948–2007)

In memory of my mom.

Acknowledgments

I would foremostly like to thank my wife Ana Lúcia who has patiently allowed me to study at the expense of household chores, holidays, parent visits and many other little things. Although he is currently too little to understand, I would also like to thank our Raul for moments of pure joy that just a happy two-year old can provide.

Big thanks to my colleagues Pablo Carneiro Elias and Thiago de Almeida Bastos who initially helped me to make my ways around PUC-Rio and amortize the culture shock which I experienced when I arrived at the campus. They made my stay here much more easier than if I had tried to discover everything all by myself. Thanks especially for including me into your study group during the first year of my graduate studies at PUC — guys you've really been of much help. I hope that I have helped you at least half as much as you have helped me.

Next I would like to thank Prof. Marcelo Gattass who made me a member of the crew at Tecgraf and this way allowed me to have access to all the resources that just an institute can provide, as well as giving me an academic home. Further, thanks for great classes that actually motivated me to adopt a computer-vision approach to the problem described in this MSc thesis.

I would also like to thank Prof. Alberto Raposo, the coordinator of Virtual Reality group at Tecgraf, for creating a great environment to work, and doing everything possible to provide all the needed resources for this work to take place.

Thanks to all the members of the Virtual Reality group and to all the people Tecgraf is composed of — no wonder that Tecgraf has been doing so well lately — just a group of very talented people can achieve such an impressive string of successes.

Special thanks to Rosane and Cosme at the library of the Department of Computer Science, for their patience and expertise, and who sometimes had to endure my less then stellar record in devolving books on time.

Finally, thanks to PUC-Rio in general — I find this university to be a great place, in a great setting, and a real place of excellence. Being here is being part of something special.

Abstract

Kolarić, Siniša; Gattass, Marcelo; Raposo, Alberto Barbosa. **Towards direct spatial manipulation of virtual 3D objects using visual tracking and gesture recognition of unmarked hands**. Rio de Janeiro, 2008. 117p. MSc Thesis — Department of Computer Science, Pontifical Catholic University of Rio de Janeiro.

The need to perform spatial manipulations (like selection, translation, rotation, and scaling) of virtual 3D objects is common to many types of software applications, including computer-aided design (CAD), computer-aided modeling (CAM) and scientific and engineering visualization applications. In this work, a prototype application for manipulation of 3D virtual objects using free-hand 3D movements of bare (that is, unmarked, uninstrumented) hands, as well as using one-handed and two-handed manipulation gestures, is demonstrated. The user moves his hands in the work volume situated immediately above the desktop, and the system effectively integrates both hands (their centroids) into the virtual environment corresponding to this work volume. The hands are being detected and their posture recognized using the Viola-Jones detection method, and the hand posture recognition thus obtained is then used for switching between manipulation modes. Full 3D tracking of up to two hands is obtained by a combination of 2D "flocks-of-KLT-features" tracking and 3D reconstruction based on stereo triangulation.

Keywords

Direct manipulation of virtual 3D objects. Augmented reality. Mixed reality. 3D input devices. 3D interaction techniques. Computer vision. Hand detection. Hand tracking. Hand gesture recognition.

Resumo

Kolarić, Siniša; Gattass, Marcelo; Raposo, Alberto Barbosa. **Rumo à manipulação direta espacial de objetos virtuais 3D usando rastreamento baseado em visão e no reconhecimento de gestos de mãos sem marcadores**. Rio de Janeiro, 2008. 117p. Dissertação de mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A necessidade de executar manipulações espaciais (como seleção, deslocamento, rotação, e escalamento) de objetos virtuais 3D é comum a muitos tipos de aplicações do software, inclusive aplicações de *computer-aided design* (CAD), *computer-aided modeling* (CAM) e aplicações de visualização científica e de engenharia. Neste trabalho é apresentado um protótipo de aplicação para manipulação de objetos virtuais 3D utilizando movimentos livres de mãos e sem o uso de marcadores, podendo-se fazer gestos com uma ou duas mãos. O usuário move as mãos no volume de trabalho situado imediatamente acima da mesa, e o sistema integra ambas as mãos (seus centróides) no ambiente virtual que corresponde a este volume de trabalho. As mãos são detectadas e seus gestos reconhecidos usando o método de detecção de Viola-Jones. Tal reconhecimento de gestos é assim usado para ligar e desligar modalidades da manipulação. O rastreamento 3D de até duas mãos é então obtido por uma combinação de rastreamento 2D chamado "*flocks-of-KLT-features*" e reconstrução 3D baseada em triangulação estéreo.

Palavras–chave

Manipulação direta espacial de objetos virtuais 3D. Realidade aumentada. Realidade mista. Dispositivos de entrada 3D. Técnicas de interação 3D. Visão por computador. Detecção de mãos. Rastreamento de mãos. Reconhecimento de gestos manuais.

Summary

List of figures

List of tables

People don't understand 3D. They experience it.

Ivan E. Sutherland, *American computer scientist*

1 Introduction

The author's original intention, a couple of years ago, was to develop a kind of an intuitive, dataglove-based interface for Computer-Aided Design (CAD) applications. The idea was to interact with 3D geometry *directly*, i.e. using hands, just as we interact with physical 3D objects in the real world. However, while undoubtedly there exist application areas where datagloves are the best and optimal choice, they (datagloves, sometimes also called *cyber-gloves*) continue to be expensive, invasive, and essentially a niche technology.

Subsequently, it was in 2006 when the author came into contact with advanced computer-vision techniques (through the graduate-level course "Visão Computacional e Realidade Aumentada", held by professor Marcelo Gattass), that the idea to use vision-based hand tracking for the same type of a CAD interface, instead of using datagloves, was born.

However, using computer vision (CV) techniques to detect and track human hands is difficult. Although in recent years many advances in the field of vision-based hand tracking (and tracking of articulated structures in general) have taken place, a lot of theoretical and practical problems remain. For example, while detecting an arbitrarily oriented and illuminated human hand in an digital image reliably, robustly and quickly is a difficult problem, it is simply a non-issue with datagloves. Furthermore, tracking a hand using CV techniques is even harder, while datagloves can do the same tracking with almost exact precision. On the other hand, CV techniques adopted in this work don't require the user to don any device, and offer a complete freedom of movements.

That said, and as I have already mentioned at the beginning of this preface, the central theme and objective of this dissertation is actually the *manipulation of 3D objects using hands*, and computer vision is "merely" our vehicle to achieve that end. Put differently, the main motivation for doing this work was to try to map manipulation operations as we know them in the real, physical world (when we manipulate real objects), to a set of corresponding manipulation operations in the virtual environments, so that we can manipulate virtual 3D objects.

1.1 Historical context

As of time of writing (second half of the first decade of the 21st century), the field of computing is as alive and active as ever. Just in the last couple of years, we've witnessed the meteoric rise of Google, Inc. ("organizing the world's information and making it universally accessible and useful"), the widest possible dissemination of mobile computing platforms, and last but not least a slow but steady switch (actually, a sea change) to many-core and multi-core computing, which will soon have deep repercutions on how we conceptualize, develop and use computer programs.

Taking a look at the hardware interfaces of our good old standard Personal Computer (PC), apparently we cannot find the same level of dynamism. The peripherals we use to interact with our PCs practically haven't changed since the original IBM PC was ushered into the IT scene in 1981 — granted the data buses have become wider and faster, processors tick at 3 GHz instead of at 4.77 MHz, RAM and disk sizes are much more plentiful and the operating systems that drive this hardware are much more complex and capable — but the keyboard stayed almost the same as the one featured by the original IBM PC, and the mouse is conceptually equivalent to the one Douglas Engelbart invented and perfected in the 1960s. Furthermore, the technologies that were predicted to revolutionize human-computer interfaces, like for example speech recognition, haven't come to realize their full potential.

Yet, exactly in the last couple of years preceding this work, we have witnessed some interesting developments in the field of HCI[1] (specifically, launches of commercial products, which were of course preceeded by years and even decades of academic and corporate research); characteristically, all these developments try to provide more *natural* ways for users to interface with computers, like for example *touch* and *hand gestures.*

For starters, in 2007 Microsoft Inc. introduced[2] the *Microsoft Surface*^{tm}, a computerized table whose tabletop (a touch-sensitive display) can detect user's touches and recognize physical objects by means of five infra-red cameras situated beneath the surface. The device itself is built around the principles of **direct interaction** (the user manipulates virtual objects using hands and/or fingers), and **multi-touch interaction** (the user can apply one, several or all his fingers to interact with the device; also, many users can interact with the device at the same time).

Further, also in 2007, Apple Inc. launched a commercially successful

[1]HCI is the acronym for "Human-Computer Interaction".
[2]www.microsoft.com/surface/

product line which includes the iPod[3] and the iPhone[4] whose interfaces also feature a touchscreen able to detect touching and dragging finger gestures. Similarly to Microsoft's Surface, it's possible to stretch a photo by placing two fingers on two opposite corners of the image, then spreading the fingers thus enlarging or shrinking the image.

In an somewhat earlier development, Nintendo Inc. introduced in 2005 the gaming console *Wii*[5] and the associated *Wiimote*, which actually acts as an accelerometer and 3D position tracker; in conjuction with the associated software, this system can recognize certain hand gestures. This way it's possible, for example, to simulate hitting the tennis ball by doing the equivalent, *natural* hand movement and gesture.

Finally, we have earlier devices that also support direct manipulation, like MERL's DiamondTouch[6] [1], a multi-user, touch-and-gesture-activated screen for supporting small group collaboration, and the Responsive Workbench [2], a virtual work environment.

As a conclusion, there seems to exist a certain momentum towards more natural and intuitive ways to interact with computers, although only time will tell how successful this push for more intuitive and "natural" interfaces will ultimately be.

1.2 The motivation

The motivation for this work is simply to try to use our own hands to interact with 3D geometry, and also due to the author's deep insatisfaction with the current state of affairs in the field of 3D user interfaces. While the mouse (and a number of specialized devices like 3D mice, SpaceBall^{TM} and SpaceNavigator^{TM} by 3Dconnexion Inc., and similar devices), have proved their value in various 3D application contexts along the last *several* decades, this work is an attempt to offer an arguably more *natural* and *intuitive* method to interact with a 3D computer model, especially having certain types of user communities in mind (for example, architectural and graphic designers, sculptors, and artists in general).

[3]www.apple.com/ipod/
[4]www.apple.com/iphone/
[5]www.nintendo.com/wii/
[6]www.merl.com/projects/DiamondTouch/

1.3 The scope covered by this dissertation

The title of this MSc dissertation is **Direct spatial manipulation of virtual 3D objects using vision-based tracking and gesture recognition of unmarked hands**, which implies the following:

- **Direct spatial manipulation** — the expression "direct manipulation" (without the adjective "spatial" or "3D") refers to the *technique of making user interfaces more intuitive by representing the objects of interest visually and letting the user manipulate them directly with an input device like a mouse* [3]. Consequently, "direct spatial manipulation" or "direct 3D manipulation" can be considered to be a specialization of direct manipulation, in the following way:
  - we deal with the manipulation of virtual 3D geometric objects, as dinstinguished for example from manipulation of 2D icons.
  - we use free-form hand movements for spatial input, and
  - there is a minimal (or equal to zero) spatial displacement between the user's physical hand (and of its virtual representation) and the manipulated virtual 3D object.
- **Virtual 3D objects** — here the fact that we manipulate virtual (computational) 3D models, instead of physical objects, is emphasized.
- **Vision-based tracking of (unmarked) hands** — "tracking", in our context, refers to the process and techniques for future position prediction of a target object. "Hand tracking", therefore, refers to the tracking of human hand. Consequently, tracking of *unmarked* (i.e. uninstrumented, unadorned, bare) hands refers to hand tracking which does not try to instrument the hands in any way, like for example, by placing a marker on the hand. Finally, we perform tracking using passive computer vision techniques, that is, we do not consider active computer vision techniques like for example projecting a pattern onto the object of interest.
- **Vision-based gesture recognition of (unmarked) hands** — again, we use use passive computer vision techniques to recognize various hand gestures (in our case, static gestures, that is, views of hand postures) which modulate the movements of human hands in the workspace.

Therefore, according to the definitions above, this dissertation describes an approach to direct spatial manipulation of virtual 3D objects, using passive computer vision techniques to detect and track user's hands in the workspace, as well as recognize hand gestures made by the user in the workspace.

1.4 The structure of this MSc thesis

This MSc thesis consists of two parts: the first part describes related work, and the second part describes the prototype software application.

The first part, **Related Work**, describes prior work done in all the areas that are relevant to this MSc thesis, and consists of the following chapters:

- Chapter 2 describes the anatomy and biomechanical properties of the human hand, as well as gives an overview of existing biomechanical models of the human hand.
- Chapter 3 describes one-handed and two-handed gestures for manipulation, as well as gives the theoretical framework for hand gestures and hand gesture recognition.
- Chapter 4 describes interaction techniques for direct 3D manipulation.
- Chapter 5 gives an overview of computer vision topics for hand detection, recognition and tracking.
- Finally, Appendix A gives a timeline of research in manipulation of virtual geometrical objects.

The second part, **Prototype Application**, consists of the following chapters and appendices:

- Chapter 6 describes all the aspects of the prototype application.
- Chapter 7 gives conclusions and future work.
- Appendix B describes the Viola-Jones detection method, which is used in the prototype for hand detection.
- Appendix C describes KLT features, which are used in the prototype for hand tracking.
- Appendix D describes the Hartley-Sturm triangulation method, which is used in the prototype to perform 3D reconstruction of the tracked hand's position in workspace.

2
Human hand

Since this work deals with *direct manipulation*, i.e. manipulation using hands, obviously human hands are of crucial importance for this exposition. In order to approach the research and development of a 3D user interface as the one proposed in this work, we must be familiar with the anatomy, biomechanical properties and biomechanical models of human hand to a sufficient degree.

2.1
Introduction

The human hand is a complex mechanical manipulator, able to perform both delicate manipulations (e.g. a clocksmith trying to repair a watch) and powerful manipulations (a gardener breaking and turning over earth with a spade). Besides being a tool to achieve a goal (for example, "move the teacup nearer"), hands are also a source of *tactile feedback*, obtained through the sense of *touch*. Each hand is controlled by the opposing brain hemisphere — for example, for right-handed people the right hand is controlled by the left side of the brain.

2.2
Human hand anatomy

The human hand (Figure 2.1, adopted from [4]) is a highly articulate object, consisting of 27 bones. (Note that the two bones at the bottom, *radius* and *ulna*, do not belong to the hand, but to the arm.) The bones of the human hand can be grouped as follows:

1. **Wrist bones (carpals, *carpus* in Latin)** — these bones include 8 bones called *carpal bones*, which are located between a) the two elbow bones (*radius* and *ulna*) and b) the palm (*metacarpus*).

2. **Palm bones (metacarpals, *metacarpus* in Latin)** — these bones include 5 bones called *metacarpal* bones (or simply *metacarpals*).

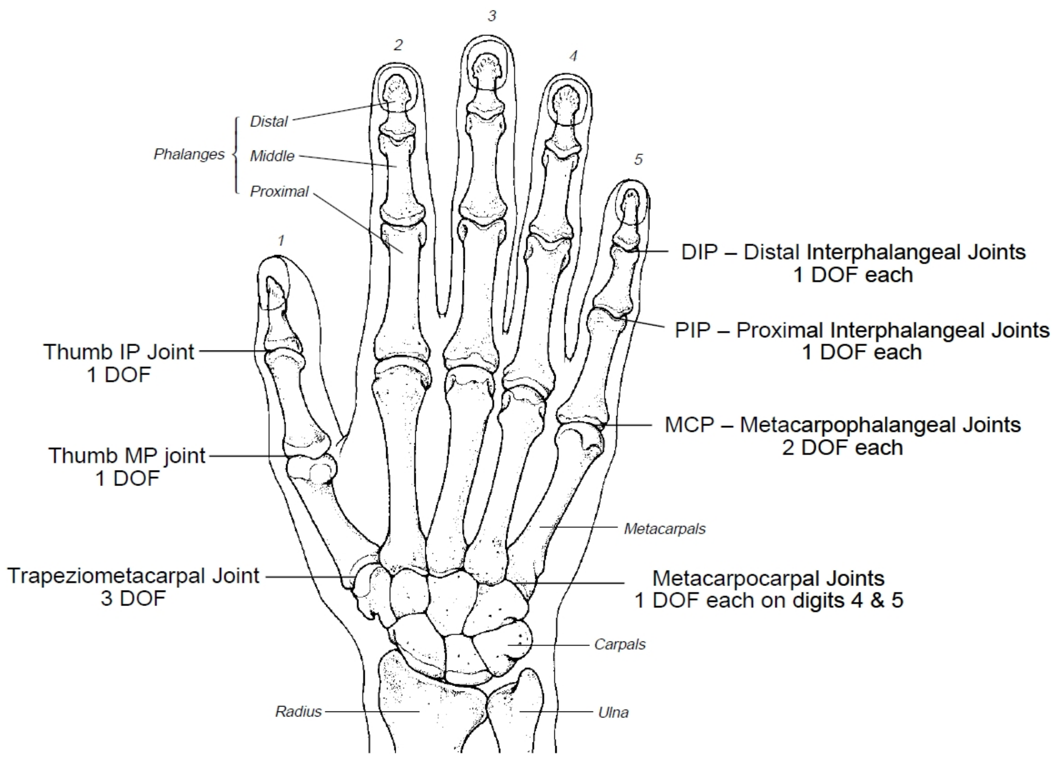

Figure 2.1: A drawing of a human hand, with joins and bones emphasized

3. **Finger bones (phalanges, also *phalanges* in Latin)** — these bones include 14 bones of all the fingers (thumb, index finger, middle finger, ring finger/annualry and little finger). All fingers except the thumb have three phalanges:

   (a) *proximal* phalange,

   (b) *intermediate* phalange, and

   (c) *distal* phalange.

   The **thumb** has just two phalanges (the intermediate phalange is missing):

   (a) *proximal*, and

   (b) *distal*.

For the illustration of muscles and tendons (cords or bands of inelastic tissue connecting a muscle with its bony attachment) of the human hand, refer to Figure 2.2, adopted from [4].

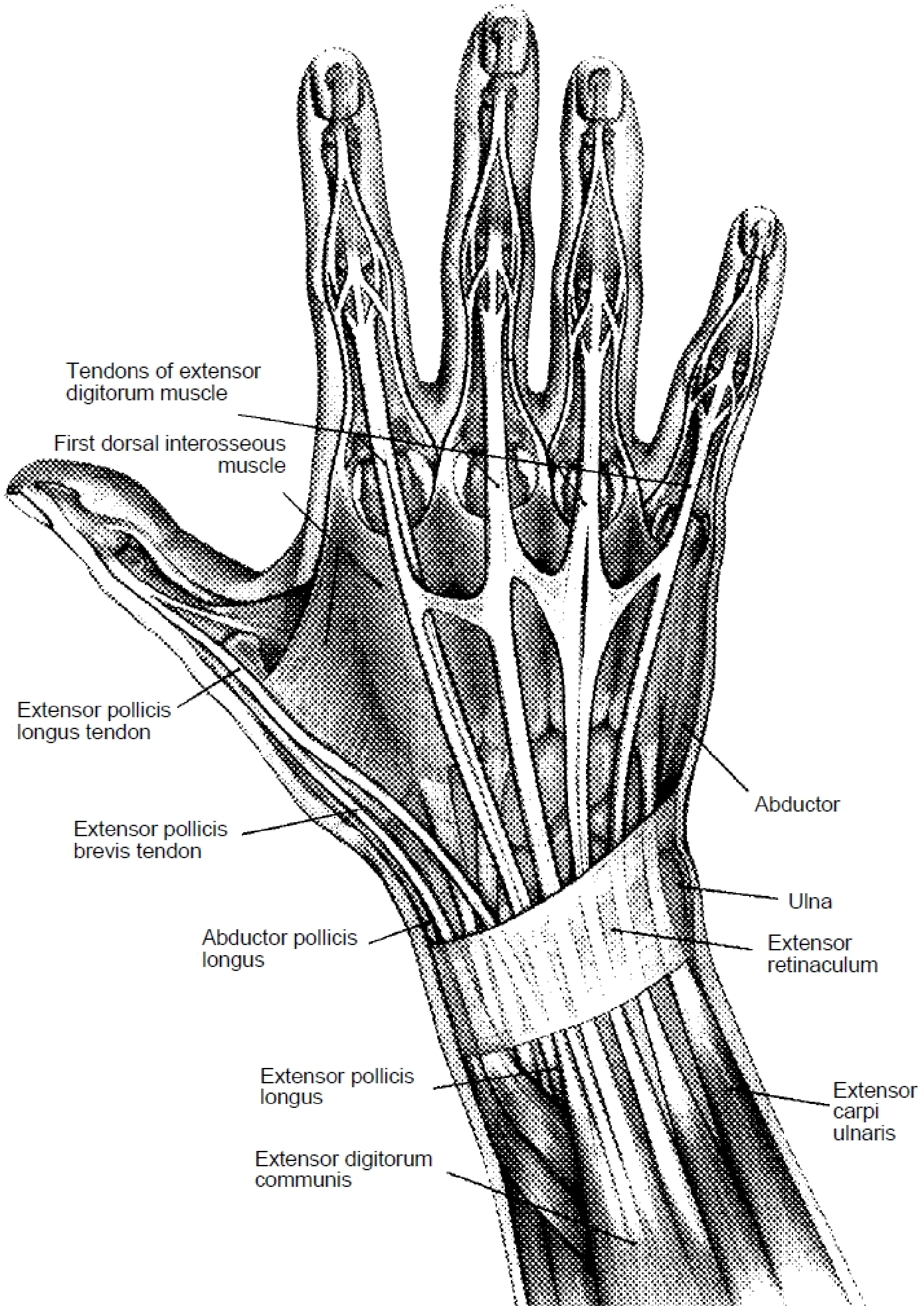
Tendons of extensor digitorum muscle
First dorsal interosseous muscle
Extensor pollicis longus tendon
Extensor pollicis brevis tendon
Abductor pollicis longus
Extensor pollicis longus
Extensor digitorum communis
Abductor
Ulna
Extensor retinaculum
Extensor carpi ulnaris

Figure 2.2: Muscles and tendons of the human hand

2.3 Human hand modeling

Hand modeling can be defined in our context as the construction and use of a computer-based 3D biomechanical model of human hand. A hand model is

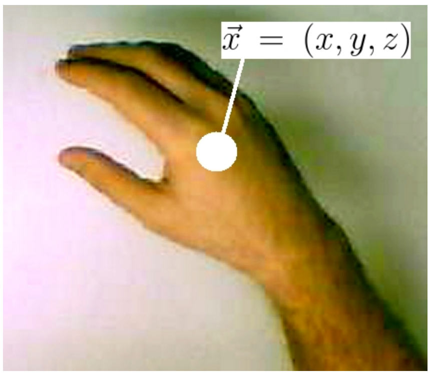

Figure 2.3: 3-d.o.f. hand model

therefore an approximation of the real human hand which, as has been shown in Section 2.2, has 27 bones, of which 19 belong to the palm (5 of 19) and fingers (14 of 19), and the remaining 8 belong to the wrist. The bones thus form a system with a certain total count of degrees of freedom (d.o.f.), and can be abstracted and modelled as such.

Various models, with various d.o.f. have been proposed in the literature. The simplest possible hand model must be the one with just three d.o.f. (i.e. one 3D position $\vec{x} = (x, y, z)$). In this case, this 3D position then designates a characteristic 3D point for a hand, for example the hand's current mass center.

The highest number of d.o.f. for a hand model model is theoretically unlimited, however in practice the highest number of d.o.f. ranges from 27 to 33 d.o.f.

Of course, it depends on the application which hand model we will choose. For some applications, just 3 d.o.f. is sufficient, however for other application only models with high d.o.f. will suffice.

2.3.1 3 d.o.f. hand model

This model (Figure 2.3) can be considered the simplest possible 3D hand model, because in this model the human hand is represented by one single 3D point $\vec{x} = (x, y, z)$, and has therefore just 3 degrees of freedom. The criteria for deriving $\vec{x}$ is diverse, and can range from the mass centroid (barycentre) all the way to the mean of a number of "good features to track" (see Appendix C on page 113).

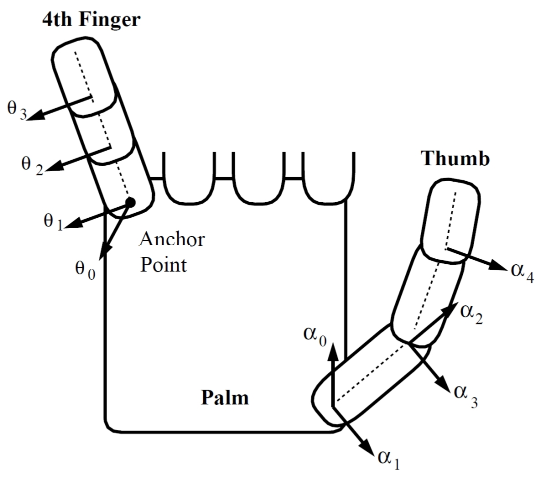

Figure 2.4: Rehg-Kanade 27 d.o.f. hand model

2.3.2 27 d.o.f. hand models

Rehg-Kanade model

Figure 2.4 depicts the Rehg-Kanade hand model, described in [5]. Here the hand is modelled as a collection of five kinematic chains (four fingers + thumb) attached to a base (the palm):

- the palm is modelled as a rigid block with 6 d.o.f. — 3 for position and 3 for orientation.
- each of the four fingers are planar mechanisms with 4 d.o.f. — 3 d.o.f. determine the finger's configuration within the plane, while the 4th d.o.f. is reserved for abduction (i.e. finger movements away from the central axis).
- the thumb is modelled as a 5-d.o.f. mechanism, using the approach described in [6]

Wu-Huang model

The Wu-Huang model [7] (see Figure 2.5) is similar to the Rehg-Kanade model, with the same number of d.o.f. and the same distribution of d.o.f.

2.3.3 33 d.o.f. hand models

Model by Nirei et al

In [8] the hand is modelled as a collection of 21 segments and 20 joints, thus giving us a 33-d.o.f. hand model (see Figure 2.6):

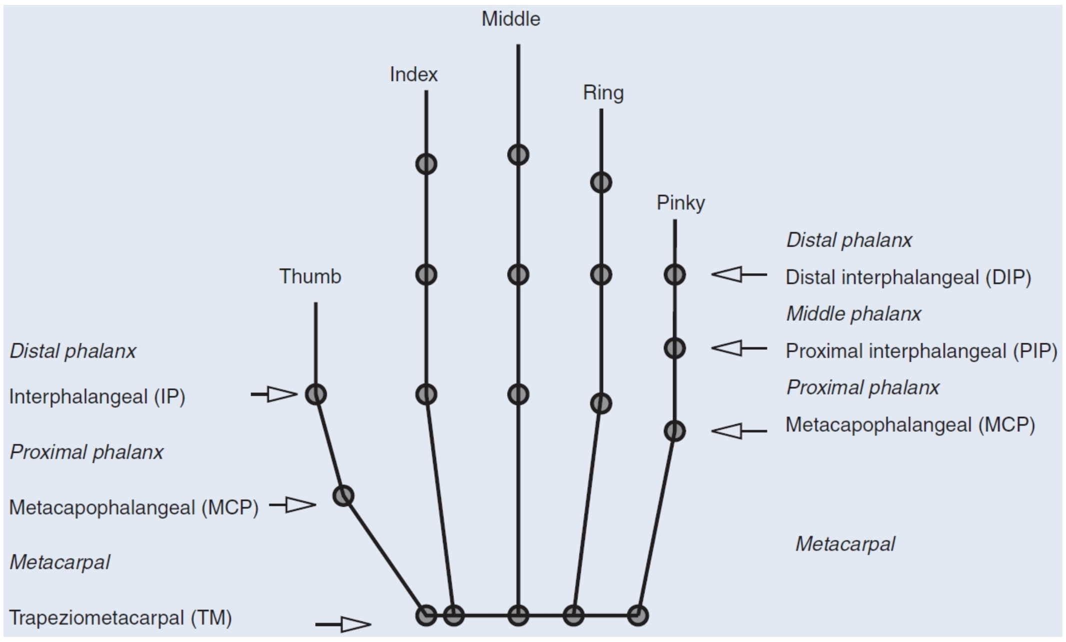

Figure 2.5: Wu-Huang 27 d.o.f. hand model

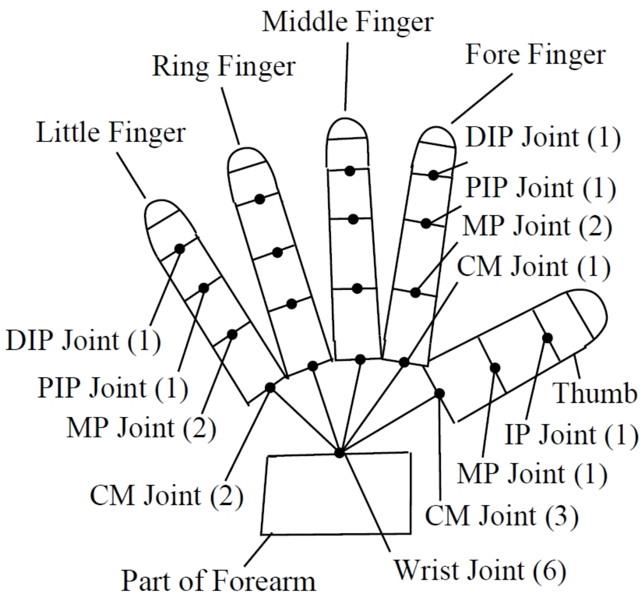

Figure 2.6: 33 d.o.f. hand model by Nirei et al

3 Hand gestures for manipulation

The previous chapter demonstrated how we can parametrize individual parts of the human hand, using biomechanical structures with various total numbers of degrees of freedom. Given this data (the trajectory of hand and its parts in space and time), we can recognize gestures made by the hand.

3.1 One-handed and gestures in general

As per Figure 3.1 adopted from [9], any hand movement can be classified as 1) gesture or 2) unintentional movement [10], [11], [9].

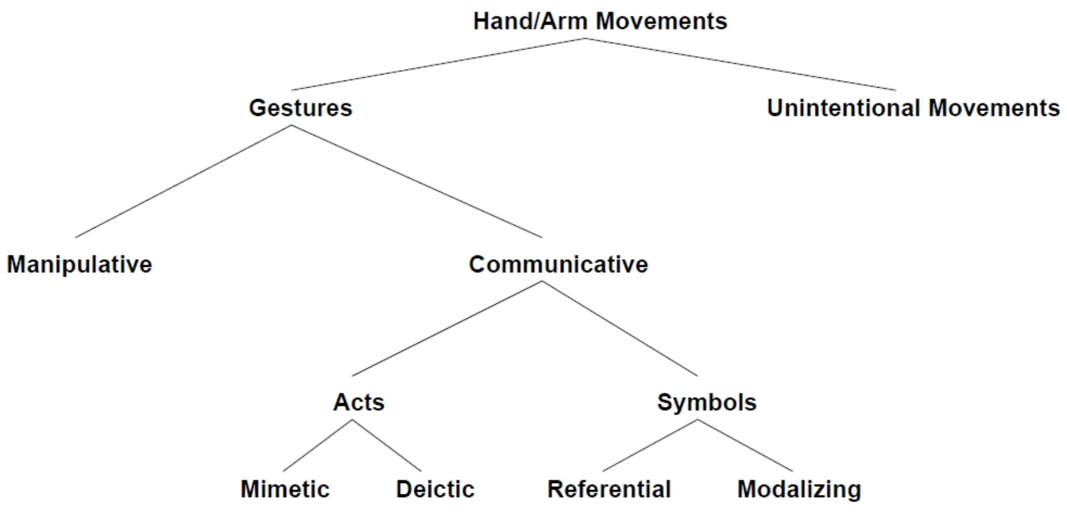

Figure 3.1: Taxonomy of gestures. In this work mostly the manipulative gestures (see extreme left of the figure) will be considered

There are two major classes of gestures:

1. **Manipulative gestures** — in this work, we are mainly interested in these gestures. Manipulative gestures act directly on objects in the real or virtual environment, like grabbing, moving, touching, rotating and stretching an object. For example [11], a pianist‘s hand movements are meant to touch the piano keys. Differently from an orchestral conductor’s hand motions, the pianist’s hand and finger motions are not meant to

communicate with anyone, but to simply touch (*manipulate*) the piano keys in order to produce music.

Some manipulative gestures are amenable to be processed by computer vision techniques, in order to manipulate 3D virtual objects. For example, a closed fist can be interpreted as a grabbing of a virtual object.

2. **Communicative gestures** — meant for visual interpretation, not for acting on objects. These gestures try to convey a message, an information. For example, an orchestral conductor's hand motions are intended to communicate temporal, affective and interpretative information to the orchestra, NOT to act on an object.

   Communicative gestures are further subdivided into (see Figure 3.1):

   – **Acts** — the movements performed relate directly to the intended interpretation.
     – **Mimetic gestures** — imitate some actions and can be considered pantomimes. These are characterized by their "iconicity". For example, a smoker going through the motion of "lighting-up" with a cigarette in his mouth indicates that he needs a light. Such gestures are usually generated on-the-fly without predetermined convention. The more novel the pantomime, the more exaggerated the motion would be to convey its intent.
     – **Deictic gestures** — point at something, and as such are very interesting for HCI. Subdivided into: **specific** — to select a particular object or location, **generic** — elicit the identity of a class of object by picking one of its members, and **metonymic** — when pointing at an object in order to signify some entity related to it.

   – **Symbols** — a type of motion short-hand.
     – **Referential gestures** — designated objects or concepts. For example, circular motion of index finger may be a referent for a wheel. Or, rubbing the index finger and the thumb in a circular fashion, is referential to money.
     – **Modalizing gestures** — serve in conjuction with some other means of communication (for example speech) to indicate the opinion of the communicator. For example, at a party, one might say to another, "Have you seen your husband?" (holding her hands apart to indicate that he is overweight). The resulting chuckle would not be understandable if one listened only to an audio transcript of the exchange.

3.2 Two-handed gestures

The seminal work for two-handed gestures is Guiard's work [12]. Guiard classifies manual tasks into the following three categories:

1. **Unimanual tasks** — to perform these tasks only one hand is necessary. Examples include combing one's hair, or dropping an object.

2. **Bimanual tasks** — two hands are necessary to perform these tasks. Can be divided into:

   – **Bimanual symmetric tasks** — both hands have an equal role in performing an activity/task which can be either 1) in the same phase (for example, rowing) or 2) out of phase (for example, climbing a mountain, or boxing).
   – **Bimanual asymmetric tasks** — for these tasks, in right-handed people, motion produced by the right hand tends to be articulated with motion produced by the left. A complex coordination between both hands is required, as in for example playing a guitar, or writing something on a paper with a pen.

Section 6.4 on page 58 shows how:

– manipulation operations `OP_SELECT`, `OP_DESELECT`and `OP_TRANSLATE`were implemented in this work, in order to perform *unimanual* tasks of selection, deselection, and translation of virtual 3D objects.

– manipulation operation `OP_SCALE`was implemented in this work in order to perform *bimanual-symmetric* task of scaling virtual 3D objects.

– manipulation operation `OP_ROTATE`was implemented in this work in order to perform *bimanual-asymmetric* task of rotating virtual 3D objects.

3.3 Modeling of hand gestures

In this section, the theoretical background on hand gesture modeling (spatial and temporal) will be given, based on exposition in [9]. Hand gesture modeling can be:

– **spatial**, and

– **temporal**.

3.3.1 Spatial modeling of gestures

Hand and arm movements trace trajectories in 3D space, therefore an useful gesture model must include a formal parametrization of the paths that all bones of the hand (finger bones, palm bones ...) and the arm (humerus, ulma, radius ...) trace in the 3D space.

Definition 1 *A complete gesture model for HCI is one whose parameters belong to the parameter space S for one-handed gestures defined as:*

$$S = \{\vec{x} : \textit{position of all hand and arm joints, and fingertips, in 3D space}\}$$

Note that the dimensionality of S is relatively high (at least $23 \times 3 = 69$), because we have 23 hand and arm joints, and fingertips, for one hand.

Definition 2 *Let $\vec{h}(t) \in S$ be a vector that describes the hand pose within a 3D Euclidean space at time t in the parameter space S. A* **hand gesture** *is then represented by a trajectory in the parameter space S over a suitably defined time interval $I = (a, b) \subset R$.*

The time interval I is also called "gesture interval".

3.3.2 Temporal modeling of gestures

Human gestures happen in time. In order to differentiate gestures from unintentional hand and arm movements, we have to determine the gesture interval I. Kendon [13] calls this gesture interval I a "gesture phrase", consisting of the following three phases:

1. **Preparation** — a preparatory movement that sets the hand in motion from some resting position.
2. **Nucleus (Peak, Stroke)** — has some definite form and enhanced dynamic qualities.
3. **Retraction** — here hand either 1) returns to the resting position or 2) repositions for the new gesture phase.

The only exception to this "preparation-nucleus-retraction" rule are the so-called "beats" that are related to the rhythmic structure of the speech.

Definition 3 *The following set of rules determines the temporal segmentation of gestures:*

1. *Gesture interval consists of three phases: preparation, stroke and retraction.*
2. *Hand pose during the stroke follows a classifiable path in the parameter space S.*
3. *Gestures are confined to a specified spatial volume (workspace).*
4. *Repetitive hand movements are gestures.*
5. *Manipulative hand gestures have longer gesture interval lengths than communicative gestures.*

3.4 Hand gesture recognition

Gesture recognition attempts to classify the trajectory of arm and hand in the parameter space S as a member of some meaningful subset of the parameter subset S. See Figure 3.2 adopted from [9].

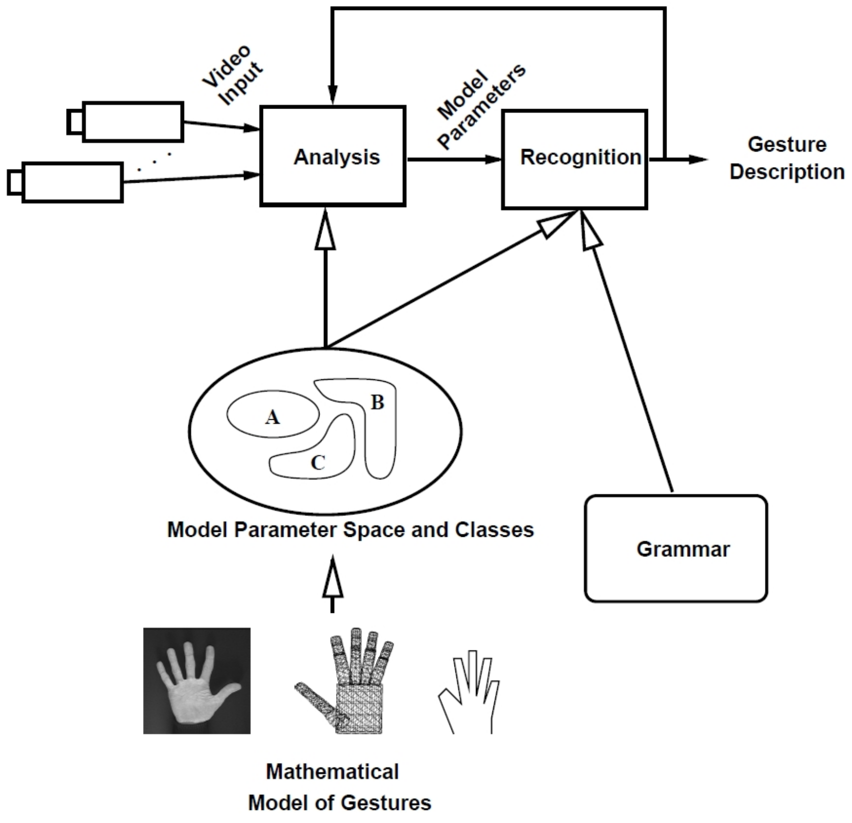

Figure 3.2: Gesture interpretation. The Recognition phase has the hand pose (Model Parameters), the database of all defined gestures (classes of trajectories) and a Grammar (serving to influence the gesture recognition depending on the current working context) as input parameters

Relevant classification methods include:

- **Hidden Markov models** — here a gesture being modelled is assumed to be a Markov process with unknown parameters, and the goal then is to determine these unknown parameters from the estimated hand pose.
- **K-means algorithm** — this algorithm classifies input gestures into clusters, where each cluster is defined by its object attributes.
- **Neural networks** — here a network of artificial neurons, which has previously been trained, tries to classify a hand trajectory in S as a gesture.

4
Interaction techniques for direct 3D manipulation

Interaction techniques can be defined as [14] *the methods used to accomplish a given task via the given interface.*

Interaction techniques include both hardware and software components. Software components translate (map) input information captured by these input devices into system actions, which in their turn exercise an effect on the geometry of the 3D scene.

Captured input can include information such as the path followed by the hand moving through space, or a button pressed. This input is then transformed into a desired manipulation action, such as selecting, translating, rotating or scaling a virtual 3D object.

4.1
Selecting virtual 3D objects

Selecting a virtual 3D object is *the task of acquiring or identifying a particular object from the entire set of objects available* [14]. The parameters of the selection task include:

1. Vector **User** $\longrightarrow$ **Target 3D object** (i.e. distance and direction)
2. **Size** of the target 3D object
3. **Density of 3D objects** around the target 3D object
4. **Number of target 3D objects** to be selected
5. **Occlusion** of the target 3D object.

Further, each selection operation consists of the following three sub-operations (or subtasks), which decompose even further:

1. **User indicates the 3D object** (that is, user points at the 3D object):
   - by **occluding** the object
   - by **touching** the object — can be through a list, voice selection, automatic or iconic objects.

   - by **pointing** at the object — can be 2D, through 3D gaze, or 3D hand.
   - by **selecting** the object indirectly

2. **User confirms the selection** (that is, user triggers the selection):
   - by triggering some **event**, for example by pressing a button
   - by doing a **hand gesture**, for example by touching mutually two fingertips
   - by saying a **voice command**, for example "SELECT!"
   - by **no explicit command** — in this case it suffices to move the pointer into the object.

3. **User obtains feedback** from the system whether the 3D object has been selected or not through:
   - **text/symbolic** feedback, for example by printing out the message "OBJECT X SELECTED" to standard output (console)
   - **aural (audio)** feedback, by playing back a sound
   - **visual** feedback, for example by coloring the object in a distinct color
   - **force/tactile** feedback, for example by shaking the haptic hand.

4.2 Translating virtual 3D objects

Translating a virtual 3D object is *the task of changing the 3D position of an object* [14]. Before we can translate an object it has to be selected (see Section 4.1). The parameters of the translation task include:

1. Vector **User** $\longrightarrow$ **Initial position** (i.e. distance and direction)
2. Vector **User** $\longrightarrow$ **Target position** (i.e. distance and direction)
3. **Precision** required for translation.

Given these parameters, the translation vector **Initial position** $\longrightarrow$ **Target position** is then the difference between the second and the first vector above, constrained by the given precision **Precision**.

Since the target 3D object has been selected, now the user's hand motion can be mapped directly to the user's virtual hand motion in the VE, which therefore also translates the selected 3D object. Various mappings between the user's real hand and virtual hand are possible:

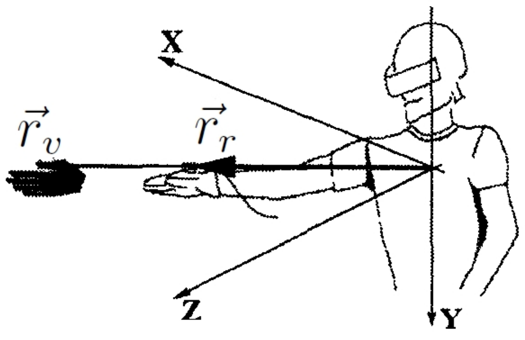

Figure 4.1: Go-go technique extends the hand non-linearly for $\vec{r} > \vec{r}_r$

- **Classical (simple) virtual hand**. Here the relation between the the virtual hand's position $\vec{p}_v = (x_v, y_v, z_v)$ and the real hand's position $\vec{p}_r = (x_r, y_r, z_r)$ is a simple one:

$$\vec{p}_v = \alpha \vec{p}_r$$

  where α is a scaling factor between the real and virtual coordinate systems.

- **Go-Go technique** by Poupyrev *et al.* [15], see Figure 4.1. Instead of two Cartesian coordinate systems, here we use two 3D polar coordinate systems, one of the VE and the other one of the physical world, and both have the origin at the same point — in the centre of the user (for example, in the centre of the user's head or at the centre of the user's torso). Here a 3D point is determined by the coordinate tuple (r, ϕ, θ) which in turn defines the radius vector $\vec{r}$.

  If the vector $\vec{r}_r = (r_r, \phi_r, \theta_r)$ points from the origin to the hand in the physical world, then the corresponding radius vector $\vec{r}_v = (r_v, \phi_v, \theta_v) = (r_v, \phi_r, \theta_r)$ (note that the angles stay the same) in the VE coordinate system is

$$r_v = \begin{cases} r_r & \text{if } r_r \leq D \\ r_r + \alpha(r_r - D)^2 & \text{otherwise} \end{cases}$$

  where α is a scaling factor between the real and virtual coordinate systems and D is some threshold distance. The formula above expresses the fact that when the physical hand is close to the body, its movements correspond linearly to the virtual hand's movements. However if the user extends his hand beyond distance D, the arm's length grows at quadratic rate, thus enabling the user to translate the object to remote positions.

- **Other translation techniques**. Besides these two mappings between the user's real hand and virtual hand, which enable us to translate a 3D object, other translation techniques are available however are not relevant for this exposition. These techniques include World-in-Miniature [16] where the user can manipulate iconic (shrank, downsized) representations of 3D objects, "Stretch Go-Go" [17] and ray-casting techniques where the translation is not hand-centered, but relative to the hand-object axis.

4.3 Rotating virtual 3D objects

Rotating a virtual 3D object is *the task of changing the orientation of an object* [14]. Before we can rotate an object it has to be selected (see Section 4.1). The parameters of the rotation task include:

1. **Distance** to target 3D object
2. **Initial rotation**
3. **Final rotation** or **Amount of rotation**
4. **Precision** required for rotation.

Given these parameters, we rotate the select object by either the angle **Final rotation** – **Initial rotation** or the angle **Amount of rotation**, starting at the angle **Initial rotation**, and respecting the given precision **Precision**.

Since the target 3D object has been selected, now the user's hand orientation can be mapped directly to the user's virtual hand orientation in the VE, which therefore also rotates the selected 3D object. For this, we use the same mapping as discussed in Section 4.2.

4.4 Scaling virtual 3D objects

Scaling a virtual 3D object is *the task of adjusting an object, according to a scale*. Before we can scale an object it has to be selected (see Section 4.1). Mine [18] lists two types of scaling:

– **Uniform scaling** — here the selected object is scaled by the same factor along all three extents (dimensions) equally.
– **Non-uniform scaling** — here the selected object is scaled along each dimension separately.

Mine [18] also lists two key parameters for a scaling operation:

– **Scaling factor** — this is the real number which multiplies one (in the case of non-uniform scaling), or all three dimensions (in the case of uniform scaling) of the object being scaled. The scaling factor can be:

  - Hand-controlled
  - Controlled through a physical control
  - Controlled through a virtual control

- **Center of scaling** — determines the behaviour of the scaling operation. It is the point which all objects move towards when you scale down and all points move away from when scaling up. Center of scaling can be on/in the:

  - **Object** — for object centered scaling, the center of scaling is defined to be the center of the selected object.
  - **Hand** — in hand centered scaling, all selected objects will scale about the current location of the hand.
  - **User-defined point** — alternately, objects can scale about some user defined center of scaling. User defined centers of scaling can be specified via direct interaction (e.g. the user can grab some icon representing the center of scaling and move it about) or by using some remote agent which the user moves (via remote control) to specify the desired center of scaling.

5
Computer vision for hand recognition

5.1
Cameras

A *camera*, in its everyday meaning, is a device for taking *photographs*, which in their turn are 2D representations of a 3D scene in the form of either 1) a raster image file, 2) a printout or 3) a transparent slide. In this work, we're mainly interested in digital cameras. They are capable to produce raster image files (see Section 5.2 for more on digital images), suitable for computer processing.

Real cameras can be modeled using various mathematical models. In mathematical terms, a *camera model* (frequently called *camera* as well) is defined as a mapping between the 3D (Euclidean) world being observed and the resulting 2D image:

$$\text{camera: } \text{3D scene} \longrightarrow \text{2D image}$$

For the purposes of this exposition, one mathematical model in particular satisfies our needs: the **pinhole camera** model.

5.1.1
Pinhole camera model

The pinhole camera (Figure 5.1) consists of a hollow box, whose side has been perforated by a small hole, called *pinhole*. (Alternative name for pinhole is *optical center*, designated by $\vec{C}$ in Figure 5.1). Light emanating from the scene enters the pinhole and gets projected onto the inner surface (*screen*) opposite to the hole. The screen has a light-sensitive surface (either an CCD or CMOS chip in the case of digital cameras, and photographic film in the case of analog cameras) which enables the camera to record the 3D scene.

Due to the geometry of the image-creating process (which can be easily understood looking at Figure 5.1), the projected image is reversed (i.e. upside-down). Also, the smaller the hole is, the projected image is sharper due to the

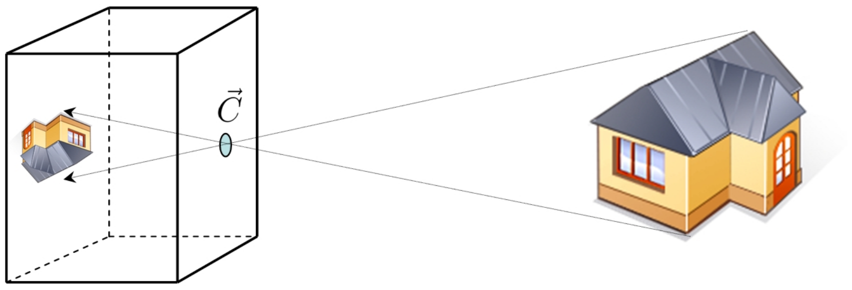

Figure 5.1: Pinhole camera, with pinhole (i.e. optical center) at $\vec{C}$

smaller number of light rays falling onto one specific spot in the image plane; and vice versa, light rays emanating from one particular position in 3D scene fall on just one spot on the screen. On the other hand, very small pinholes lead to the aberration of the image because light entering the box starts to suffer the phenomenon of *wave difraction*. Also, too small a hole permits just a very low amount of light energy to enter the box, which leads to exposure times which are simply too long for many purposes.

Since the projected image is upside down, we sometimes replace the screen with another (hypothetical) screen between the optical centre $\vec{C}$ and the 3D scene, thus creating a *virtual image* which has the same orientation as the 3D scene:

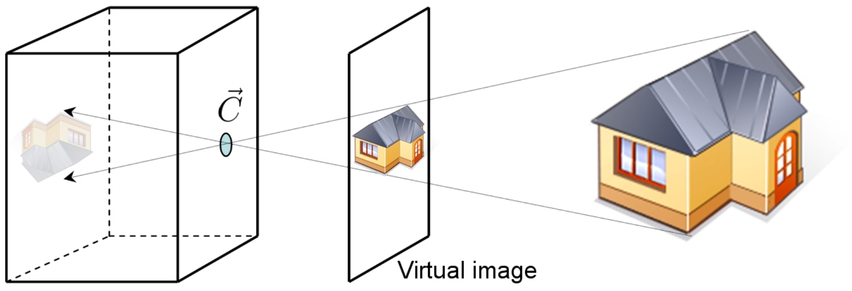

Figure 5.2: Pinhole camera with screen in the front of $\vec{C}$

5.2 Digital images

Digital images we refer to in this work are the so-called *intensity images*, two-dimensional discrete arrays of picture elements (also called pixels) with M rows and N columns, where each of the $M \times N$ pixels measure the amount of visible electromagnetic energy (i.e. light) that fell onto that position at the moment the image was taken.

An image can also be considered a planar (2D) coordinate system, where the origin is fixed at the upper-left corner, and where each 2D point (u, v) is

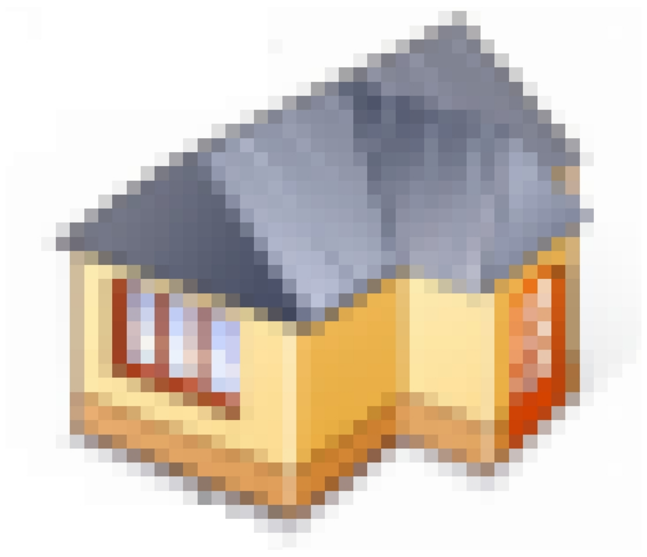

Figure 5.3: A digital image consisting of 48 × 43 pixels

defined by its horizontal distance u from the origin and its vertical distance v from the origin.

5.3 Mono vision

This section deals with mono-vision, which is a vision obtained using just one camera. We'll determine mathematically how a specific 3D point P with the associated position vector $\vec{X} = (X, Y, Z)$ gets projected into a specific 2D pixel point $\vec{u} = (u, v)$ in the raster image.

5.3.1 Relevant coordinate systems

There are four coordinate systems (Figure 5.4) involved in the computation of the 2D raster image of a 3D scene:

1. **World coordinate system (WCS)** — this is our global, absolute 3D system. In this work, WCS is fixed on the table, so that $+x$ points to the right, $+y$ away from the user and $+z$ up. We designate the origin of WCS by O.

2. **Camera coordinate system (CCS)** — this 3D coordinate system is fixed so that its origin O_c is at the camera's optical centre. No imagine that you're peeping through the camera's optical finder. In this case, $+x$ is to the right, $+y$ is up, and $+z$ points to the user.

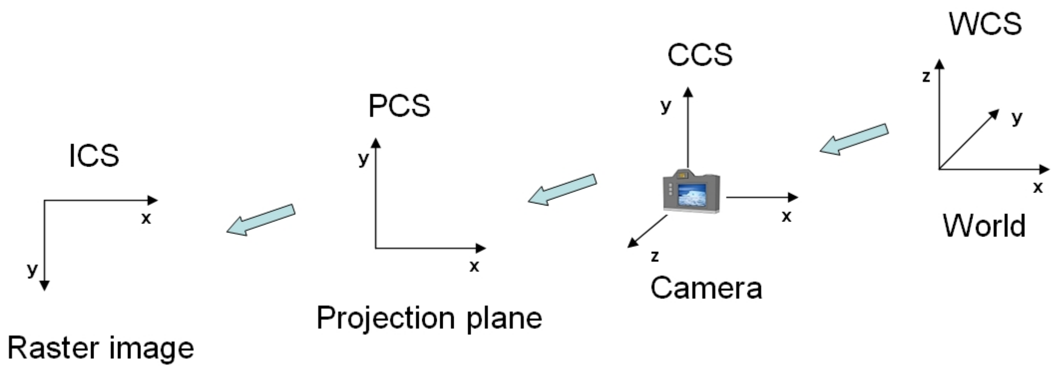

Figure 5.4: Coordinate systems in the world–camera–projection–raster image chain.

3. **Projection plane coordinate system (PCS)** — this is a 2D coordinate system embedded into the projection plane. The origin O_p is fixed at the orthogonal projection of the camera's optical centre onto the projection plane, $+x$ axis points right, and $+y$ points up.

4. **Raster image coordinate system (ICS)** — this is a final 2D coordinate system, which expresses the position of a point in PCS relative to the grid defined by the rectangular array of pixels. In this work, the origin $o = (o_x, o_y)$ of ICS is fixed at the upper left corner of the image produced in PCS. The $+x$ points therefore to the right, and $+y$ points down.

5.3.2 Transformations between coordinate systems

As we have seen, we need to deal with four coordinate systems: WCS, CCS, PCS and ICS. Therefore, we have to consider three coordinate transformations between (three of) them, in order to understand mathematically how the 2D raster image forms from the 3D scene:

1. CCS $\longleftarrow$ WCS

2. PCS $\longleftarrow$ CCS

3. ICS $\longleftarrow$ PCS

Schematically:

$$\boxed{\text{Raster image (2D)}} \longleftarrow \boxed{\text{Projection plane (2D)}} \longleftarrow \boxed{\text{Camera (3D)}} \longleftarrow \boxed{\text{World (3D)}}$$

or equivalently:

$$\text{ICS} \longleftarrow \text{PCS} \longleftarrow \text{CCS} \longleftarrow \text{WCS}$$

or, using coordinates:

$$(u, v) \longleftarrow (u_p, u_p) \longleftarrow (X_c, Y_c, Z_c) \longleftarrow (X, Y, Z)$$

CCS $\longleftarrow$ WCS

Here, 3D world (i.e. expressed in WCS) coordinates (X, Y, Z) are being re-computed as 3D camera coordinates (X_c, Y_c, Z_c) (i.e. CCS coordinates). Let P be a 3D point, $\vec{X} = (X, Y, Z)$ its representation in WCS, and $\vec{X}_c = (X_c, Y_c, Z_c)$ be the presentation (i.e. coordinates) of P in CCS. It holds:

$$\vec{X}_c \longleftarrow \vec{X}$$

$$\vec{X}_c = R\vec{X} + \vec{t}$$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

where R is an 3×3 rotation matrix that rotates WCS relative to CCS (i.e. its columns are coordinates of the unitary vectors that make up a base in WCS, relative to the vector base of CCS), and $\vec{t} = (t_1, t_2, t_3)^\tau$ is the vector $\overrightarrow{O_cO}$ (the vector from the camera's origin to the world's origin). See Figure 5.5 that depicts the transition from WCS to CCS.

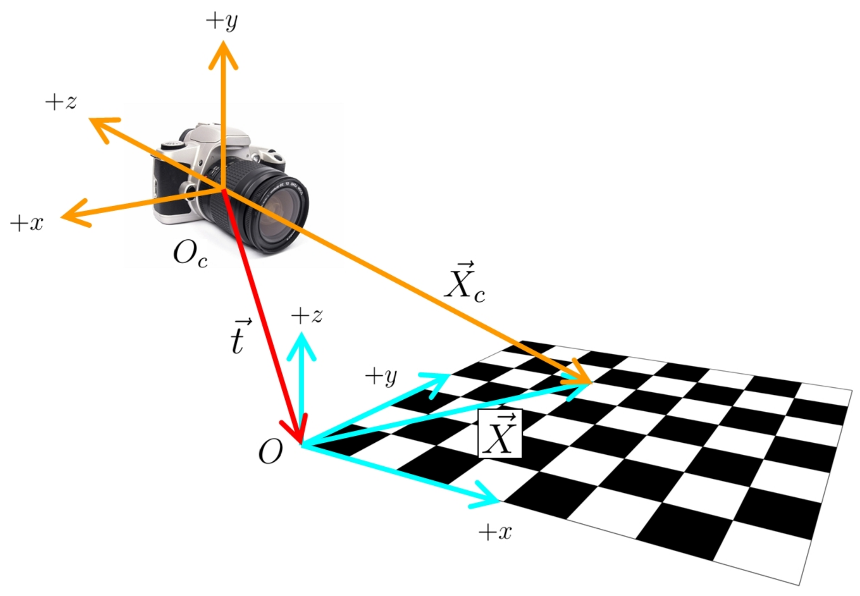

Figure 5.5: Going from 3D world to 3D camera coordinates (CCS $\longleftarrow$ WCS)

The rotation matrix R and translation vector $\vec{t}$ are also called *extrinsic parameters* of the camera. Extrinsic parameters determine the camera's location and orientation relative to WCS.

PCS $\longleftarrow\!\!\dashv$ CCS

Having now calculated CCS coordinates $\vec{X}_c = (X_c, Y_c, Z_c)$, we can compute 2D coordinates $\vec{u}_p = (u_p, v_p)$ of the projection of $\vec{X}_c$ onto the camera's projection plane. We'll use the pinhole camera model.

$$\vec{u}_p \longleftarrow\!\!\dashv \vec{X}_c$$

$$(u_p, v_p) = \left(f\frac{X_c}{Z_c},\ f\frac{Y_c}{Z_c} \right) \tag{5-1}$$

For the computation in Eq. 5-1 to take place, we of course must know the value of f (focal length of the camera, expressed in meters). Parameter f is one of the so-called *intrinsic parameters* of the camera.

ICS $\longleftarrow\!\!\dashv$ PCS

Finally, the projected 2D point $\vec{u}_p = (u_p, v_p)$ can now be expressed relative to the pixel array grid as a 2D point $\vec{u} = (u, v)$. Note that $\vec{u}_p = (u_p, v_p)$ is expressed in meters, while $\vec{u} = (u, v)$ is expressed in pixels.

$$\vec{u} \longleftarrow\!\!\dashv \vec{u}_p$$

$$(u, v) = \left(-\frac{u_p}{s_x} + o_x, -\frac{v_p}{s_y} + o_y \right)$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\frac{u_p}{s_x} + o_x \\ -\frac{v_p}{s_y} + o_y \end{bmatrix}$$

For this computation, we must know the values of s_x and s_y (dimensions of one sensor element in the CCD/CMOS chip, expressed in meters), and o_x and o_y (translation values for the raster image origin). Parameters s_x, s_y, o_x, o_y also belong to the set of intrinsic parameters of the camera.

Composing transformations PCS $\longleftarrow\!\!\dashv$ CCS and ICS $\longleftarrow\!\!\dashv$ PCS

Composing transformations PCS $\longleftarrow\!\!\dashv$ CCS and ICS $\longleftarrow\!\!\dashv$ PCS we obtain the transformation ICS $\longleftarrow\!\!\dashv$ CCS:

$$\text{ICS} \longleftarrow\!\!\dashv \text{PCS} \longleftarrow\!\!\dashv \text{CCS}$$

that is, we go from the 3D camera system CCS to the 2D raster image system directly:

$$\text{ICS} \longmapsfrom \text{CCS}$$
$$(u,v) = \left(-\frac{f}{s_x}\cdot\frac{X_c}{Z_c} + o_x, -\frac{f}{s_y}\cdot\frac{Y_c}{Z_c} + o_y\right)$$

If we now define $f_x = \frac{f}{s_x}$ and $f_y = \frac{f}{s_y}$ we obtain

$$(u,v) = \left(-f_x\cdot\frac{X_c}{Z_c} + o_x, -f_y\cdot\frac{Y_c}{Z_c} + o_y\right)$$

When working with vision setups, we rarely get to know focal length f of the camera exactly (only when we have the camera manufacturer data). Instead, using calibration techniques (see Section 5.3.3) we usually obtain just f_x and f_y (focal lengths expressed in pixels). Using the same calibration techniques we also obtain o_x and o_y (expressed in pixels as well), which are coordinates of the image center (principal point), which is the intersection between the image plane and the optical axis (line through O_c, perpendicular to the image plane).

Considering this, in this work we frequently ignore PCS and go straight from CCS (3D camera system) to ICS (2D raster image system). Thus the pipeline looks like this:

$$\boxed{\text{Raster image (2D)}} \longmapsfrom \boxed{\text{Camera (3D)}} \longmapsfrom \boxed{\text{World (3D)}}$$

or equivalently:

$$\text{ICS} \longmapsfrom \text{CCS} \longmapsfrom \text{WCS}$$

or, using coordinates:

$$(u,v) \longmapsfrom (X_c, X_c, Z_c) \longmapsfrom (X, Y, Z)$$

5.3.3 Mono-camera calibration

The (mono) camera calibration is a process where all the parameters of a camera are being determined. The parameters include *intrinsic* camera parameters and *extrinsic* camera parameters.

– **Intrinsic camera parameters**:

  – $f_x \in \mathbb{R}, f_y \in \mathbb{R}$: two focal lengths in the x- and y-direction (in pixels)
  – $o_x \in \mathbb{N}, o_y \in \mathbb{N}$: two integer x- and y-coordinates of the image center (in pixels). Note that when we are working with subpixel precision, we have $o_x \in \mathbb{R}, o_y \in \mathbb{R}$ (in pixels, but we use real numbers here)

  - $\alpha = \frac{s_y}{s_x}$: pixel aspect ratio (pixel deformation)(dimensionless)
  - k_1, k_2: two radial distortion coefficients (dimensionless)

- **Extrinsic camera parameters**:

  - rotation matrix R (dimensionless) and
  - translation vector $\vec{t}$ (in meters).

While there exist many calibration techniques, i.e. methods to extract both the extrinsic and intrinsic camera parameters listed above, we focus on the method by Zhang [19].

5.3.4 Zhang's camera calibration method

Zhang's camera calibration method [19] enables the user to obtain intrinsic and extrinsic camera parameters taking several (at least two) snapshots of a planar pattern (for example, a checkerboard consisting of a grid of alternating black and white squares, or any other planar pattern with easily distinguishable features). For example, Figure 5.5 shows a camera observing an 8×7 checkerboard pattern.

The method first finds an initial solution using a closed-form expression. This solution is then refined using the Levenberg-Marquardt algorithm, which is a nonlinear minimization method.

5.4 Stereo vision

Stereo vision, or *stereopsis*, is the main mechanism through which humans perceive spatial depth. In stereopsis, a 3D point P gets projected into two different locations on both eyes' retinas. The difference between these two locations, called *stereo disparity*, is then processed by the brain which, in conjuction with other factors (like for example eye-to-eye distance), estimates the depth coordinate of the point P.

In the computational context, stereo vision is obtained using two cameras. We say that two cameras, when employed to compute stereo, make up a *stereo rig*. Just like a in the case of mono vision, stereo vision has its own geometry. Figure 5.6 shows the geometry of a stereo rig.

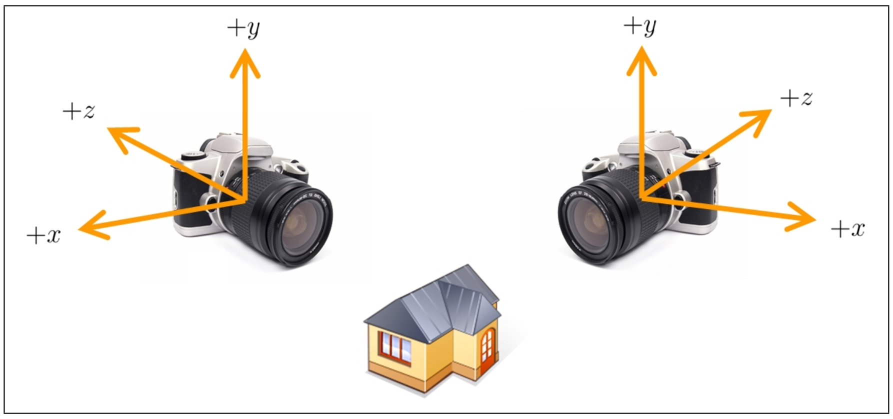

Figure 5.6: Stereo rig. If the two cameras take a snapshot at the same instant, the two photos make a stereo pair of photos.

5.4.1 Stereo 3D reconstruction

The expression "stereo 3D reconstruction" refers to the process of determining the 3D structure and 3D position of an observed object, given a number N of correspondences $\{(\vec{u}_1, \vec{u}'_1), (\vec{u}_2, \vec{u}'_2), \ldots, (\vec{u}_N, \vec{u}'_N)\}$ in the left and right image of the stereo input stream.

Reconstruction based on triangulation

Triangulation is a method of determining the position of a fixed point using another two fixed points a known distance apart. Therefore, triangulation is a stereo 3D reconstruction of just one corresponding pair. Having one correspondence $(\vec{u}_1, \vec{u}'_1)$ of one observed (photographed) point feature P, we can use triangulation to determine the 3D location (X, Y, Z) of this feature.

Supposing that a point P is visible in both images, and that we know the pixel coordinates $\vec{u}$ and $\vec{u}'$ of both projections of the point P in both images, we can easily compute two rays in space — one ray passing through the left camera's optical centre and $\vec{u}$, and the second ray through the right camera's optical centre and $\vec{u}'$. The triangulation problem is then equivalent to finding the intersection of these two rays in space.

However, there exist several problems with this approach. If we knew $\vec{u}, \vec{u}'$ and camera's parameters exactly, of course we would be able to recover P easily, using formulas of elementary vector algebra. The problems are the following:

- **Floating-point arithmetics errors** — since we use floating-point arithmetics (because all the underlying parameters are actually continuous physical values), we induce numerical errors to the triangulation process. As a consequence of this, the two rays can never intersect exactly.

– **Measurement errors** — we can never measure $\vec{u}, \vec{u}'$ and determine camera's parameters exactly. As a consequence, we can never compute the intersection of two rays exactly.

Due to the aforementioned problems, which leads to the fact that the two rays *do not cross* in space, we must find other solutions to determining 3D point $\vec{X}$ using triangulation. An overview of available triangulation methods can be found in [20]. In this work, we will mention (and later implement) two triangulation methods: a simple one, called **Mid-point triangulation method**, and the optimal one (under the assumption of Gaussian noise), called **Polynomial triangulation method**.

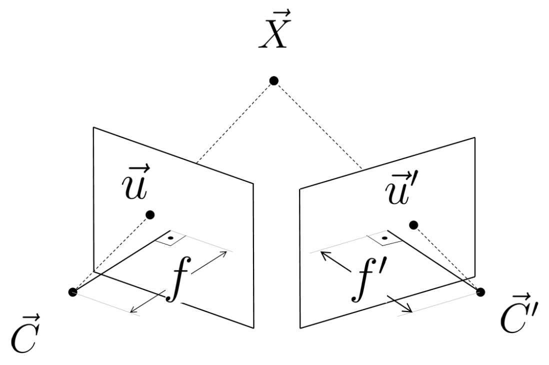

Figure 5.7: Triangulation. Knowing 3D positions of optical centers $\vec{C}, \vec{C}'$, focal lengths f, f' and 2D positions $\vec{u}, \vec{u}'$, we can determine 3D position $\vec{X}$ using various triangulation methods (for example, *mid-point* and *polynomial* triangulation methods).

Mid-point triangulation method This is probably the simplest and fastest possible triangulation method, however with serious shortcomings (see [20]).

Suppose that the corresponding (matched) 2D points are $\vec{u} = (u, v)$ and $\vec{u}' = (u', v')$, i.e. $\vec{u}, \vec{u}'$ are images of an 3D point $\vec{X} = (X, Y, Z)$ in the left and right image. The point $\vec{X}$ lies at the interesection of two rays: first ray from C through $\vec{u}$ and the second ray from C' through $\vec{u}'$ (see Figure 5.8). However, since the two rays do not actually meet in space due to the aforementioned issues, we can only *approximate* the intersection of two rays. In this method, mid-point method, we approximate this intersection with the point that lies at the *minimum distance to both rays.*

To extract $\vec{X}$, suppose first that we work in the left camera's coordinate system. Let r, r' be two rays:

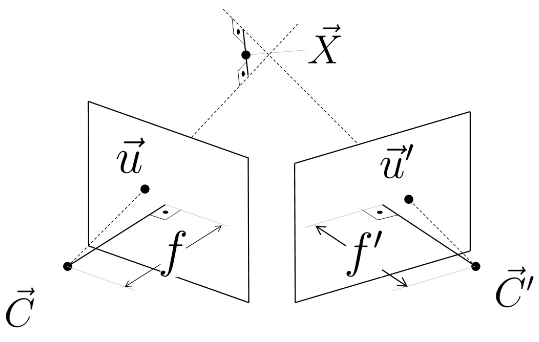

Figure 5.8: Mid-point triangulation method, which finds the point $\vec{X}$ as the point that lies at the minimum distance to both rays: first ray from C through $\vec{u}$, and the second ray from C' through $\vec{u}'$.

- $r = \{\alpha \cdot \vec{u} \mid \alpha \in \mathbb{R}\}$ the ray through C and $\vec{u}$, and
- $r' = \{\hat{\vec{t}} + \beta \cdot \hat{R}^\tau \cdot \vec{u}' \mid \beta \in \mathbb{R}\}$ the ray through C' and $\vec{u}'$.

...where $\hat{R} = R'R^\tau$, and $\hat{\vec{t}} = \vec{t} - \hat{R}^\tau \vec{t}'$. Let $\vec{a}$ be a vector orthogonal to both r and r'. Point $\vec{X}$ is now in the middle of the segment parallel to $\vec{a}$ that joins both rays r and r'.

Now let $\vec{Y} = \alpha_0 \cdot \vec{u}$ one endpoint of the segment, and $\vec{Z} = \hat{\vec{t}} + \beta_0 \cdot \hat{R}^\tau \cdot \vec{u}'$ the other endpoint of the segment. Parameters α_0 and β_0 are now computed solving the following system of linear equations:

$$\alpha_0 \vec{u} - \beta_0 R^\tau \vec{u}' + \gamma(\vec{u} \times (R^\tau \vec{u}')) = \vec{t}$$

The desired 3D point $\vec{X}$ is now simply

$$\vec{X} = \frac{\vec{Y} + \vec{Z}}{2}$$

Polynomial triangulation method This method [20] gives an optimal global solution to the triangulation problem. Further, the algorithm employed is non-iterative and simple in concept, has low computation requirements and has superior performance compared with other methods.

Formulated as a least-squares minimization problem, the method computes image points $\hat{u}, \hat{u}'$ such that

$$d(u, \hat{u})^2 + d(u', \hat{u}')^2 \to \min, \qquad \hat{u}'^\tau F \hat{u} = 0$$

where $d(*, *)$ is the Euclidean distance function and F the fundamental matrix of the stereo rig. Assuming Gaussian error distribution (tracking of u, u' is noisy because of digitization errors), the points $\hat{u}, \hat{u}'$ are the most likely values for true image correspondences. Since the corresponding rays through $\hat{u}, \hat{u}'$ meet *exactly* in 3D space, we can now find easily x (the global position of the hand in the workspace) using other triangulation methods, for example the mid-point triangulation described above in Section 5.4.1.

5.5 Color spaces

Since we are interested in visual hand recognition, and human hand is of course covered with skin, we also have to consider the color of human skin when trying to detect a hand in an image. According to [21], human skin color can be conveniently represented and processed in the following color spaces:

– **Basic color spaces** (RGB, normalized RGB, CIE-XYZ) — these are the so-called "default" color spaces, because they are ubiquitous and their properties are well know and defined.

– **Perceptual color spaces** (HSI/HSV/HSL, TSL) — the HSI/HSV/HSL model models "perceptual" qualities like *hue*, *saturation* and *intensity* (also called *brightness*, *lightness* or *value*). The TSL model quantifies *tint* (hue with white added), *saturation* and *lightness*.

– **Orthogonal color spaces** (YCbCr, YIQ, YUV, YES) — these reduce the redundancy present in the RGB model, and model the color with as statistically independent components as possible. Luminance and chrominance components are separated, therefore these spaces are the favorable choice for skin detection.

– **Perceptually uniform color spaces** (CIE-*Lab*, CIE-*Luv*) — in these spaces, the luminance L and the chroma ab or uv are obtained through a non-linear mapping of XYZ coordinates. The advantage of these spaces is that they can represent color in a perceptually uniform way. The downside is that computing CIE-*Lab*, CIE-*Luv* colors is computationally expensive.

– **Other color spaces** — color ratios like R/G and $R/G + R/B + G/B$.

5.6 Human skin modeling

Human skin *detection* can be viewed as a two-class classification problem — given an image I in color space C, and given a pixel $I(x, y)$ in image I at position (x, y) with color $c \in C$, a detector f_C outputs 1 if the pixel $I(x, y)$ is a skin pixel, and 0 otherwise (a non-skin pixel). The domain $D(f)$ of classifier

f is therefore the color space C, and its range $R(f)$ is the set $\{0, 1\}$:

$$f_C : C \to \{0, 1\}$$

$$f_C(c) = \begin{cases} 1 & \text{if } c \text{ deemed a skin color} \\ 0 & \text{otherwise} \end{cases}$$

Again according to [21], human skin detection methods can be classified as:

- **Explicit skin-color space thresholding** — skin colors of different individuals cluster in a small region in color space. This method thus simply marks this region; if a pixel falls within this region, it is deemed a skin pixel. Has good skin detection rates, at the expense of high false positives. Simple and fast, but with many limitations (e.g. illumination must be controlled, threshold values differ for color spaces and different illumination levels, is less accurate in case of shadows, and in case the background contains objects with colors similar to skin color). Because of the limitations, this approach is usually complemented with a dynamic adaptation approach.
- **Histogram model with naive Bayes classifiers** — here a 2D or 3D color histogram is used to represent skin tones. This method is stable, unaffected by occlusions and changes in view, and can be used to differentiate a large number of objects. Slightly better than GMM or MLP (see below). However, needs a very large training set, and has high storage requirements.
- **Gaussian classifiers (SGM, GMM)** — Again, since skin colors of different individuals cluster in a small region in color space, we can model this skin distribution by a multivariate normal Gaussian distribution (this is the so-called Single Gaussian Model — SGM) or by a sum of individual Gaussians (the so-called Gaussian Mixture Models — GMM). This approach generalizes well, with less training data, and has a small storage requirements. Slightly inferior to histograms with naive Bayes classifiers, however GMMs are popular because they can generalize very well with less training data.
- **Elliptical boundary model** — Has performance slightly better than GMM, and the computational complexity is as simple as training a SGM. The downside is that it performs binary classifications only.
- **Multi-layer perceptron (MLP)** — a type of feed-forward neural network. Outperforms (together with Bayesian classifiers) Gaussian models

and Explicit skin-color space thresholding. Has very low storage requirements.

- **Self-organizing map (SOM)** — a type of neural network. Consistently better than GMMs.
- **Maximum entropy (MaxEnt)** — a statistical method for estimating probability distributions from data.
- **Bayesian network (BN)** — directed acyclic graphs that allow efficient and effective representation of the joint probability density functions.

5.7 Image features

Detecting image features in an image is a low-level but fundamental task, prerequisite for almost any higher-level computer-vision algorithm, like for example camera calibration, line detection or tracking. The term *local feature* designates a *local* property of the image, for example an edge, cross, closed curve (ellipse, or circle), KLT feature or SIFT feature (more will be said about KLT and SIFT in the text that follows).

If a feature is located in a region of an image, we call this region a “local interest region”. And taking this local interest region into consideration, we are then able to compute a “local descriptor”. Many types of descriptors have been proposed so far in the literature. As a rule, local descriptors must be invariant to image scale and rotation. In further text, terms “local descriptor” and “local feature” are considered synonymous.

From the viewpoint of vision-based tracking (see also Section 5.11 on page 53), features represent “hooks” onto which we can latch and observe their displacements from frame to frame. By tracking these features, therefore, we can track the objects to whom these features belong to. See Figure 5.9.

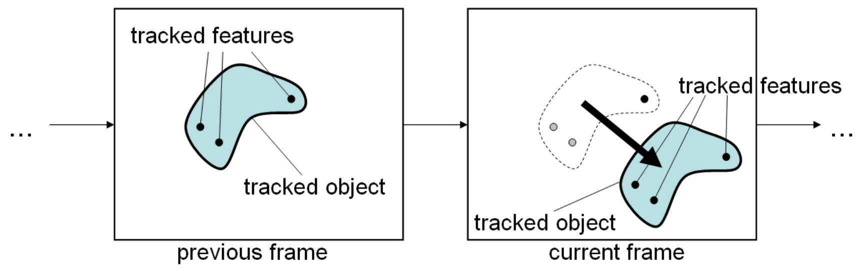

Figure 5.9: Tracking an object by tracking its features

The positive aspects of local features are:

1. **Abundance** — even tiny objects can give rise to a large number of features.
2. **Computationally cheap** — local features are generally cheap to compute, which leads, in general, to real-time tracking performance.
3. **Robustness** — features are robust to: occlusion, noise, small changes in viewpoint, and changes in illumination.

Some representative classes of features, listed from the oldest to the newest work, include:

– **Corners (Harris detector)** — the basic idea is that shifting a small window (for example, of 9×9 pixels) around a pixel should result in large change in intensity, in any direction [22].
– **KLT features** — Kanada-Lucas-Tomasi (KLT) features, also called "Good features to track", are "good" in a well-defined formal, mathematical sense. In this approach, an image feature is deemed a KLT feature only if it can be tracked in a simple, fast and accurate fashion. See Appendix C on page 113 and reference papers [23] [24] [25] .
– **SIFT features** — SIFT stands for "Scale Invariant Features Transform". This method maximizes difference of Gaussians over space and scale [26]. Note that this method has been patented.
– **SURF features** — SURF stands for "Speed Up Robust Features", and is a scale- and rotation-invariant interest point detector and descriptor. It approximates or outperforms earlier methods with respect to repeatability, distinctiveness and robustness, but can be computed and compared much faster [27].

Note that there exist many more types of local features. For the performance evaluation of various types of features, please refer to [28], which shows that SIFT outperforms all methods. However note that the publication date of this performance evaluation [28] is in 2005, while the SURF paper [27] was published one year later (in 2006) and claims performance superior to SIFT.

5.8 Hand detection

This is the process whereby a hand is detected (localized) in an image, using computer vision techniques. Basically, this process tries to answer the question "Is there a hand in this image?". (Is the answer is affirmative, then this process also returns the location of the hand in the image.) There exist various approaches to detecting and localizing a hand in an image:

- **Human skin detection** — human skin has a characteristic color signature which can be used to detect it (the skin) in an image.

  Pros: skin color is surprisingly uniform (race - white/yellow/red/black, or being suntanned doesn't matter, since the hue does not change), so color-based detection is achievable.

  Cons: other skin-colored object (like for example user's face, or other users' hands and/or faces, and so on) can enter into the image, and thus confuse the detection process. Workarounds would include 1) restricting the work area so that it contains hands only, or 2) to use hand's salient and discriminating features to dinstinguish it from other skin-colored objects.

  Other potential problem is insufficient illumination, whereby too dark a workspace prevents efficient detection of human skin's characteristic hue. Potential workaround would include using supplementary infrared cameras.

- **Motion detection** — human hands usually move at greater speeds compared with other object in the scene.

  Pros: when the background is static, motion-based detection is a practical and fast method to detect hands.

  Cons: does not work when either the camera or parts of the background move, which is frequently the case.

- **Classifier-based detection** — this is a machine-learning approach. A classifier is a mapping $f : X \to Y$ from a feature space X to a discrete set of labels Y. Classifiers can be seen as decision systems which accept values of some features or characteristics of a situation as input and produce as an output a discrete label related to the input values.

  One representative of this approach is the Viola-Jones method [29], which also uses AdaBoost [30] in order to construct so-called "strong classifiers" from a combination of "weak classifiers". Viola-Jones method can be applied to any type of object; effectiveness at detecting specifically hands has also been investigated [31].

Pros: fast, high detection accuracy, very large and very complex set of features possible (although in this work we are interested in hands only), robust (works under wide range of conditions: variations of illumination, scale, pose and camera).

Cons: a learning process (sometimes a very prolonged one) is needed. However for hands this learning process can be significantly reduced [31].

– **Hybrid detection methods** — here two or more approaches are being combined, in order to increase the robustness and reliability of hand detection. For example, skin detection can be combined with motion detection. Or, Viola-Jones method can be combined with skin detection [31], and so on.

  Pros: increased detection rate.

  Cons: increased processing load.

Since Viola-Jones detection method has been used in the prototype application (see Chapter 6, page 55), technicalities are given in detail in Appendix B on page 106.

5.9 Hand segmentation

After the hand has been detected (localized) in the image, the hand must be segmented i.e. the background must be extracted from the region containing hand's image. In other words, pixels belonging to the hand must be separated from all the remaining pixels. Approaches and techniques:

– **Thresholding** — here a special auxiliary black & white image is created for each frame, whereby pixels belonging to the hand are white, and all the rest are black. Now combining this auxiliary image with the original image (utilizing the binary operation AND), the pixels belonging to the hand are being filtered out as they are, while all the remaining pixels are filtered out/set to 0 (black).

– **Morphological functions** — functions like "dilate" and "erode" that work on black & white images (like the auxiliary one obtained through thresholding). Dilate operation causes objects to grow in size (get "thicker"), and erosion causes elements to shrink (get "thinner"). These functions serve to "improve" the hand region ultimately segmented from the image.

– **Blobs** — here contiguous regions of the image are being identified and labelled. The region ("blob") which contains the location of detected hand is now the region depicting the hand.

5.10 Hand pose estimation

Now, given the extracted (segmented) hand region (blob), we can proceed with estimating the parameters determining the hand's posture in 3D space. See Figure 5.10 for a classification of pose estimation approaches.

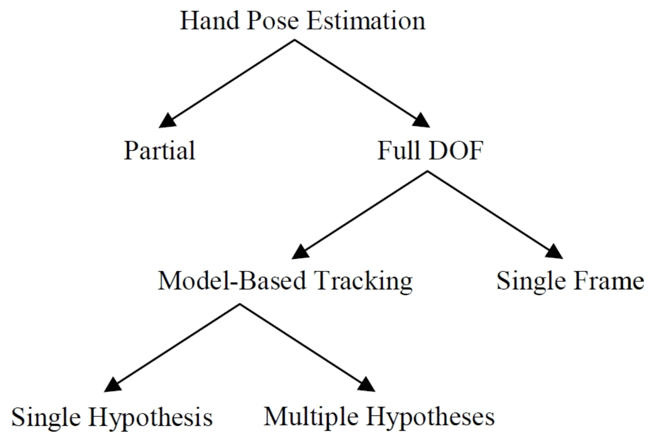

Figure 5.10: Taxonomy of hand pose estimation approaches

Partial hand pose estimation

Partial hand pose estimation relies on hand models with reduced number of d.o.f., therefore it does not try to recover the full set of 26 d.o.f. As such it relies on extracting *appearance-based* features like fingertips, orientation of fingers, global position of the hand, contours (sillhouettes), contour centroids, and curvature in order to reconstruct *directly* (i.e. without the help of a virtual 3D hand model) the pose of the partial model.

Full d.o.f. hand pose estimation

Differently from the partial hand pose estimation, the full d.o.f. hand pose estimation does not try to compute pose directly from extracted features, but instead *uses the extracted features in order to execute a search in the parameter space, so that a certain type of error can be minimized.*

- **Model-based pose estimation** — During the search, the 3D hand model (collocated into a certain position and certain orientation in virtual 3D space) is being projected onto a 2D image plane, and features extracted from this image plane are then compared with features extracted from the source video image. Can be further subdivided into:
  - **Single-hypothesis model-based pose estimation** — this one is based on restricted (local) search and retains only one, the best, estimate at each input image. The search is being done either a) optimization methods, or b) applying physical forces on the 3D hand model.
  - **Multiple-hypotheses model-based pose estimation** — retains several hypotheses about the hand pose. If one estimate gives a too big an error, the next best one is used. A big majority of this type of pose estimation utilizes Bayesian filtering or derivation thereof, specifically a) particle filters, b) tree-based filters, c) Bayesian networks, d) template database search, e) other approaches.
- **Single-frame pose estimation** — in this approach just one image (or, in the case of multiple-camera setups, the set of images taken at the same instant) is being used. Therefore, past parameter states are not being used in order to restrict the scope of the search — the whole (global) parameter space is being searched. An obvious advantage is that no analysis of past states is necessary, and disadvantage that the complete (and therefore computationally more expensive) search in parameter space must be made, although there may be no real need for that. Approaches:
  - **Classifiers** — please refer to Section 5.8. Classifiers can also work for pose estimation, not just detection, because classes of training data can be tagged by a full, predetermined set of pose data. So when a classifier detects a hand pattern in the image, we will automatically know the pose too. For this to work, a virtual 3D hand model must be used in order to produce training data (because it's practically impossible to determine pose data from pre-existing photos of real hands).
  - **Database indexing** — a large number of training samples (i.e. images of the hand model projected in every possible way) can be saved into a database, and then indexed in a special way. The database is then searched for in order to retrieve the nearest neighbour of the input image.

- **2D-3D mapping** — this is another learning approach. In a nutshell, here certain features are being computed from the (2D) input image (like moments of the hand contour, invariant to the rotation and scaling), and the mapping from this set of 2D features to the set of 3D poses is then being taught to a special machine-learning architecture named “Special Mapping Architecture”.
- **Inverse kinematics** — here the position of fingertips is being used in order to compute joint angles. This reduces the problem to a typical inverse kinematics problem, known for example in the field of robotics. The difficulties in this approach are a) to detect fingertips reliably, and 2) to solve the inverse kinematics problem (i.e. computing the joint angles) correctly.

5.11 Hand tracking

Tracking is the process of estimating the position of a tracked object, taking its previous position into consideration.

Since the hand is an articulate 3D structure, we can choose to track the hand either in its original 3D space (in this case we say that we employ *model-based* hand tracking), or in the 2D projection image plane (in this case we employ the so-called *appearance-based* tracking). We can also talk about *hybrid tracking*, a recent mode of tracking which combines elements of model- and appearance-based tracking.

5.11.1 Appearance-based hand tracking

Various tracking methods in this class differ in what *cues* they use for tracking — some, for example, use just one cue like skin color or hand motion, and another use a combination of cues (for example, skin color *and* motion). For example, in Camshift [32] just the hand’s color is being used as a cue; in CONDENSATION [33], hand contours + hand motion, and in ICONDENSATION [34] hand contours + hand motion + skin color; in “Flock of Features” [35] a combination skin color + KLT features [25], [24] (see Section 5.11 for more on KLT features).

For example, KLT tracking takes advantage of the fact that images in a video sequence are usually similar to each other. Due to the small time interval between the frames, objects being tracked haven’t had the time to travel large distances, or the shift (translation vector) between an object’s images in the previous and current image is small. This fact leads to an algorithm which extracts this translation vector thus tracking the object. For technical details on tracking based on KLT features, please refer to Appendix C on page 113.

5.11.2 Model-based hand tracking

Here the back-projection of a predefined 3D parametric hand model is being matched against the input video frame. At each frame, extracted features are being compared with the current 3D model, and the matching error computed; if the error is too large, the 3D model is adjusted in the attempt to decrease the error — if the error is still too big, we repeat the model adjustment, otherwise we found a good matching and the tracking was successful. Examples include the classic DigitEyes system [5], where a 27-d.o.f. hand model is being tracked.

5.11.3 Hybrid hand tracking

Here elements of both the model-based and appearance-based tracking are combined in an effort to get the best of two worlds. Shimada *et al* in [36] and Athitsos and Sclaroff in [37] synthesize a large number of 2D views of a software 3D hand model, and tag each of these views by the corresponding, exact hand pose vector. After this preprocessing step, appearance-based matching methods are used to process real images.

6 Prototype application

As a constituent part of this work, a prototype application which demonstrates the chosen approach to the manipulation of virtual 3D objects, has been developed.

6.1 Requirements

The basic requirements for the prototype application were as follows:

- the application must run at interactive rates.
- the application must implement the following basic set of manipulation operations: SELECT, DESELECT, TRANSLATE, ROTATE, SCALE.
- the hardware part must be based on commercial, off-the-shelf components (standard PC, inexpensive low-resolution web cameras).
- the system shall use passive stereo vision in order to reconstruct 3D position of the hand.
- the system shall use the 3 d.o.f. hand model (for each hand, therefore effectively creating a 6 d.o.f. input device), as described in Section 2.3.1 on page 20.

6.2 Constraints, assumptions and restrictions

The constraints on the system were as follows: the workplace (Figure 6.1) consists of a standard office cubicle equipped with a personal computer with two cameras (i.e. a stereo pair) connected. Cameras are fixed at the top of the cubicle and are directed down, at a certain angle, relative to the surface of the desktop. A stereo pair of cameras enables us, due to the phenomenon called *stereo disparity*, to estimate 3D positions of various hand features, thus offering us a way to integrate our hands into the VE.

Accordingly, we define the *workspace* as the intersection of the two visual cones defined by the respective cameras' field of view — the user must move his

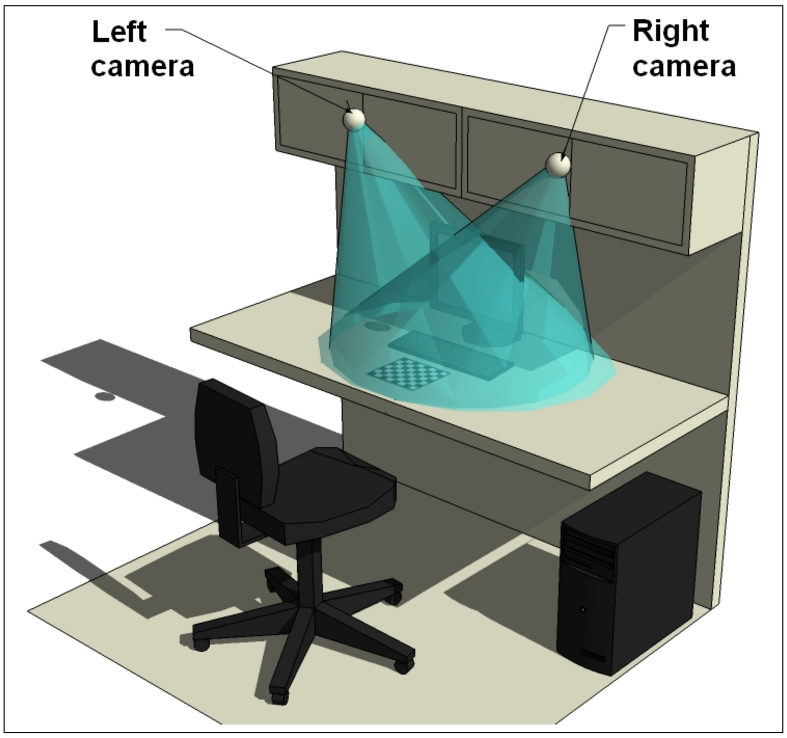

Figure 6.1: The user's workplace

hands in this working space, in order for the system to register hand movements and gestures. If a hand exits the workspace, the system stops tracking the hand.

Also:

- Cameras are fixed, oriented downwards and towards the desktop surface. and the background (desktop surface) does not change in time.
- Palms must always be approximately co-planar with the desk surface, in order for the system to successfully detect hand postures.
- Illumination cannot be too weak, because in this case the pixel segmentation process, based on human skin color, would fail.
- The hands must move approximately in the far-distance field of the cameras, because if they move too near the cameras, the perspective distortion gets large and 3D triangulation fails.

6.3 Hand postures defined

Figure 6.2 depicts the three hand postures we use in our application. Note that the image depicts the right hand only, but both the left and the right hand can assume these postures. (The left hand assumes postures which are simply mirrored relative to the vertical axis.) Also please note that the hand postures shown in the image are inclined at an angle, which approximates the natural hand inclination when it is being tracked in the workspace. We now define the following hand postures (also called *hand states*), from which all the manipulation operations are being defined, as follows:

1. `HAND_POSTURE_OPEN` — flat palm with all fingers spread apart
2. `HAND_POSTURE_POINTING` — all fingers closed, except the index finger
3. `HAND_POSTURE_FIST` — all fingers closed

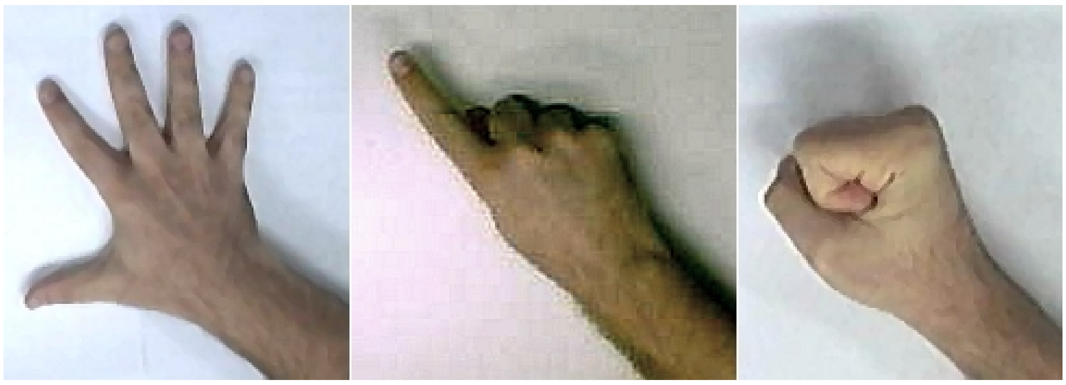

Figure 6.2: Three hand postures utilized by the system: `HAND_POSTURE_OPEN` (left), `HAND_POSTURE_POINTING` (middle) and `HAND_POSTURE_FIST` (right)

As a matter of convenience, we defined one more posture, `HAND_POSTURE_UNKNOWN`, which designates any hand posture that is not recognized by the system.

The criteria used when choosing postures and the mapping between postures and manipulation operations were:

1. the "naturalness" of the posture, that is, the similarity between similar hand movements and gestures when manipulating real, physical objects, and
2. sufficient degree of inter-posture "otherness", in other words the appearances of postures had to be sufficiently different in order to achieve better visual separability of postures, thus allowing better detection performance.

6.4 Manipulation operations implemented

This section gives the list of implemented hand gestures for the manipulation of 3D virtual objects.

Using the hand postures defined in Section 6.3, we now define *direct 3D manipulation operations.* Manipulation operations can be either one-handed or two-handed:

1. `OP_SELECT` (one-handed) — selects an object. Based on the posture `HAND_POSTURE_POINTING`.

2. `OP_DESELECT` (one-handed) — deselects an object. Based on the posture `HAND_POSTURE_POINTING`.

3. `OP_TRANSLATE` (one-handed) — translates (moves) selected objects. Based on the posture `HAND_POSTURE_FIST`.

4. `OP_ROTATE` (two-handed) — Rotates selected object. Based on two `HAND_POSTURE_POINTING` hand postures.

5. `OP_SCALE` (two-handed) — scales objects. Based on two `HAND_POSTURE_FIST` hand postures.

Therefore, we have two two-handed spatial operations: `OP_ROTATE` and `OP_SCALE` — these use both hands, the left and the right one, at the same time. The remaining three spatial operations (`OP_SELECT`, `OP_DESELECT` and `OP_TRANSLATE`) are one-handed — must be performed by just one hand, either just by the left or just by the right hand. Each of these operations will now be described in detail.

6.4.1 Selecting and deselecting objects

Operation `OP_SELECT` selects an object in the scene, while operation `OP_DESELECT` deselects an (already selected) 3D object. In order to (de)select a 3D object, the user extends the index finger of *one and exactly one* hand (thus changing that hand's state into `HAND_POSTURE_POINTING`), and moves the hand into the object. (We emphasized "one and exactly one" because *two* tracked hands which are in the `HAND_POSTURE_POINTING` state perform the operation `OP_ROTATE`, see Section 6.4.3.) As soon as the application detects that the tracked hand's centroid entered the interior of the object, while the hand is in state `HAND_POSTURE_POINTING`, the objects gets (de)selected. The user can now move her hand out of the object; the object stays (de)selected.

In WIMP terms, operation `OP_SELECT` is equivalent to moving the mouse pointer onto a screen object (for example, onto an icon) and then pressing the mouse button, thus selecting the icon. Similarly, operation `OP_DESELECT` is equivalent to moving the mouse pointer onto an already selected screen object (for example, onto an already selected icon) and then pressing the mouse button, which deselects the icon.

6.4.2 Translating objects

Operation `OP_TRANSLATE` translates (moves) all the currently selected objects (Figure 6.3). For this to work, *one and exactly one* hand must be in the `HAND_POSTURE_FIST` state. (We say "exactly one" because if both tracked hands are in the `HAND_POSTURE_FIST` state, we will be performing the `OP_SCALE` operation, see Section 6.4.4.)

The operation initiates at the moment the user changes the state of one (and exactly one) of her hands into `HAND_POSTURE_FIST`. The position of that hand's centroid is then the starting point of the translation vector. User moves the hand about, and the selected objects move along too. When the user decides the translation is just right, she changes the hand's state into `HAND_POSTURE_OPEN` thus terminating the `OP_TRANSLATE` operation.

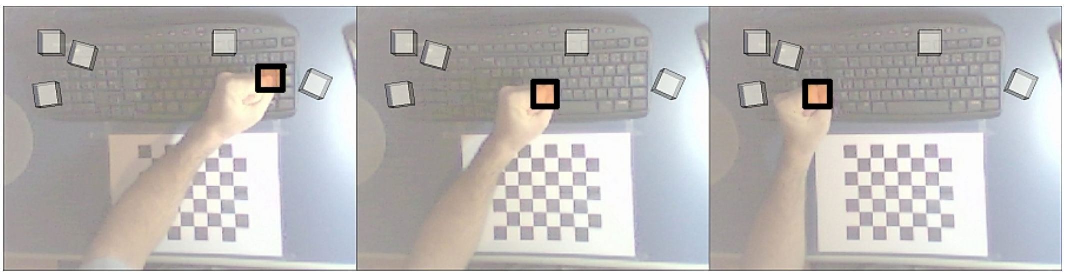

Figure 6.3: `OP_TRANSLATE` operation, based on one `HAND_POSTURE_FIST` posture

6.4.3 Rotating objects

Operation `OP_ROTATE` rotates all the currently selected objects (Figure 6.4). It is a *bimanual-asymmetric* operation (see Section 3.2 on page 25). At the moment when the application detects that *both* hands have their index finger extended (thus entering into the `HAND_POSTURE_POINTING` state), the hands' respective positions (A for the left, and B for the right hand) get memorized and defined as the vector $\vec{u} = B - A$. If subsequent positions of both the left and the right hand are C and D respectively, and if define $\vec{v} = D - C$, then the current rotation axis is the vectorial cross-product $\vec{u} \times \vec{v}$, and the rotation angle is equal to the angle between vectors $\vec{u}$ and $\vec{v}$.

The user can now move her hands about, and the selected objects rotate around their (local) origins in real time. When the user decides to stop the rotation, she changes both hands' state into HAND_POSTURE_OPEN thus terminating the OP_ROTATE operation.

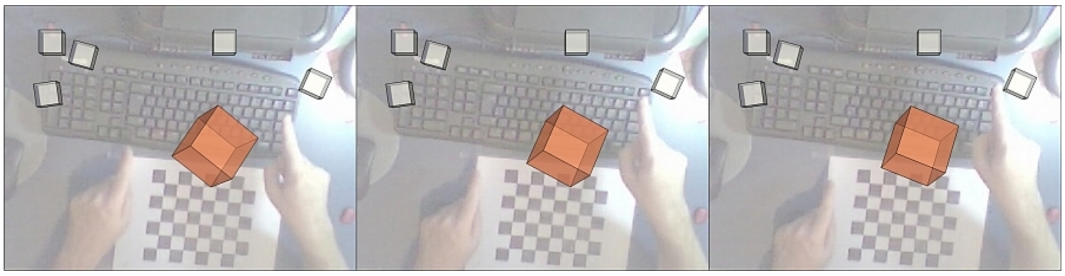

Figure 6.4: The two-handed OP_ROTATE operation is based on two HAND_POSTURE_POINTING postures. An example of a CCW rotation shown

6.4.4 Scaling objects

Operation OP_SCALE scales all the currently selected objects (Figure 6.5). It is a *bimanual-symmetric* operation (see Section 3.2 on page 25). At the moment both hands enter into the HAND_POSTURE_FIST state, the locations of both hands get memorized and defined as two points A, B. If subsequent positions of both the left and the right hand are C and D respectively, then the current scaling factor is defined simply as the ratio $\frac{|C-D|}{|A-B|}$, which is a ratio of lengths: first length defined by points A and B, and the second length defined by points C and D.

Therefore, during the scaling operation, the user can move her hands, and the selected objects scale in real-time around their (local) origins, in the directions defined by their local coordinate systems. When the user decides to stop the scaling, she changes one (or both) hand's state into HAND_POSTURE_OPEN thus terminating the OP_SCALE operation.

f

6.5 Control flow

Before anything, the stereo rig must be calibrated i.e. its parameters (intrinsic and extrinsic) determined. Only by knowing these camera parameters,

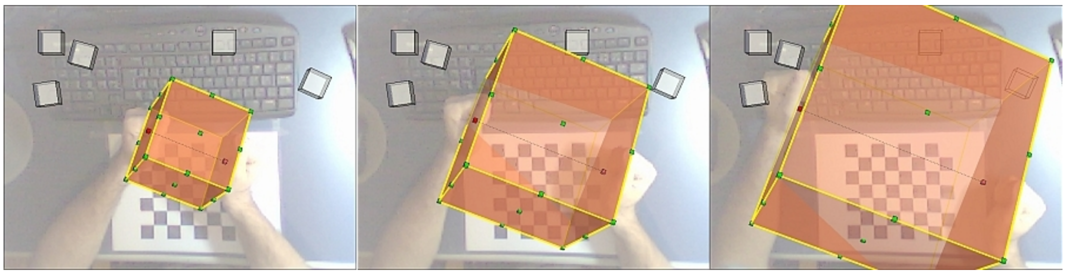

Figure 6.5: The two-handed `OP_SCALE` operation is based on two `HAND_POSTURE_FIST` postures

can CV techniques (specifically, the triangulation technique adopted) reconstruct 3D position of our hands in the workspace.

The set K of intrinsic parameters for a single camera includes two focal lengths (f_x, f_y), the principal point (o_x, o_y) and four distortion parameters (k_1, k_2, k_3, k_4):

$$K = \{(f_x, f_y), (o_x, o_y), (k_1, k_2, k_3, k_4)\}$$

We determined these intrinsic parameters using the Zhang's method [19]. For the calibration pattern we used a 8 × 7 checkerboard pattern.

Having determined two sets K_L, K_R of intrinsic parameters (for the left and right camera), we can proceed to determining extrinsic parameters (orientation and distance of a camera relative to the pattern). For this, we now fix the 8 × 7 checkerboard on the desk surface, and orient cameras so that they both have the pattern in their field of view.

Knowing both cameras' intrinsic parameters, we can now compute the extrinsic parameters (rotation matrices R_L and R_R, and translation vectors $\vec{T}_L$ and $\vec{T}_R$) of both cameras, relative to the pattern we've just fixed on the desk's surface. The extrinsic parameters R and $\vec{T}$ of the stereo rig are then simply

$$R = R_R R_L^{\tau} \qquad \vec{T} = \vec{T}_L - R^{\tau}\vec{T}_R$$

These parameters R and $\vec{T}$ now completely determine the geometry of our stereo rig and allow us to perform absolute, Euclidean 3D reconstruction of the hand's 3D position in the workspace shown in Figure 6.1.

Another important concept is the fundamental matrix F, a 3×3 matrix of rank 2, which allows us to perform 3D reconstruction using the triangulation method by see Hartley and Sturm [20]. We compute F from a set of N corresponding image points $\{\vec{x}_i \leftrightarrow \vec{x'}_i \mid i = 1, \ldots, N\}$, whereby we determine these image points as defining points of the squares of our calibration pattern (a 8 × 7 checkerboard, which can be seen for example in Figure 6.3 on page

59), detected in both the left and right image used for stereo rig calibration.

6.5.1 Hand detection

With the stereo rig calibrated, we can now proceed to detecting hands in the stereo input video stream. Here detection serves for the purpose of initiating the process of tracking, described in Section 6.5.3 below.

For this end, we define two “detection areas” (left and right), one for each of both hands in the application client area. By definition, if a hand is not being tracked, its “detection area” gets shown on the application screen as a red rectangle, at the predetermined location and with a predetermined size. If the user now moves her hand into the corresponding detection area, and puts her hand into the predefined posture (`HAND_POSTURE_OPEN` has been chosen as the tracking initialization posture), the system will detect the hand, output the corresponding bounding rectangle and start tracking the hand within this bounding rectangle. At this moment, the red rectangle disappears and is being replaced by a green rectangle, signifying that that particular hand is currently being tracked by the system.

For detection, the Viola-Jones method (see Appendix B on page 106) has been chosen as the detection method, due to the following properties:

– invariance with regard to background
– insensitivity to changes in illumination/lighting
– invariance with regard to person
– invariance with regard to camera
– invariance with regard to scale
– fast execution.

It is a method which requires training & validation using four sets of samples:

– Two *positive* sample sets:

  – **Positive training set** A — for this set we moved our right hand in posture `HAND_POSTURE_OPEN` randomly in the workspace, approximately under the natural inclination (see Figure 6.2 left), under our lab’s standard lighting conditions, and took a number (in the range of hundreds) photos containing the hand. Note that “approximately natural inclination” indicates that we included a number of shots of hands rotated to a degree ($\pm 15°$) relative to all three axes, in order to increase the robustness of the classifier.

  - **Positive validation set** B — under the same conditions as above, we took additional photos (few hundreds) to be used as validation images after the training is complete.

- Two *negative* sample sets:

  - **Negative training set** C — for the negatives, we took a number (> 1000) of images that do not contain hands in posture `HAND_POSTURE_OPEN`, from two public domain image collections (burningwell.org, easystockphotos.com).
  - **Negative validation set** D — we took additional images as a negative validation set (a couple of hundreds), from the same public domain image collections.

We then used OpenCV facilities to train boosted cascades of weak classifiers in the following way:

1. we manually marked bounding rectangles for hands in the positive training samples, and saved the list of bounding rectangles in a file F.

2. before training, we set the required false positives threshold to be 10^{-6}.

3. we ran the training tool on file F and on the two training sets, the positive A and negative C. After the training has completed, we obtained a 15-stage classifier for posture `HAND_POSTURE_OPEN` with detection rate of approximately 98%.

4. we successfully tested the classifier using sets B and D.

5. we built the trained classifier into our prototype application. An OpenCV function loads the classifier; other function detects hands in posture `HAND_POSTURE_OPEN` in the current video frame. Note that we trained the classifier with our right hand; for the left hand, before the recognition stage we mirror the left side of the application area in order to be able to use the same classifier to recognize the left hand.

6.5.2 Hand segmentation based on human skin color

The hand detection method described above not only gives an answer whether there is or isn't a hand in an image, but also the bounding rectangle of the image region containing the hand. Considering this region of interest (ROI) only, we now make use of the characteristic hue of human skin to determine the pixels belonging to a hand. The reason we perform this segmentation is to increase the hand tracking robustness — see Section 6.5.3.

To this end, we used color histograms — both in the detection stage (using HSV color space), and for the learning (using normalized RGB histogram) of the color of the hand that has just been detected, i.e. we perform color learning immediately after the hand has been detected (pre-tracking stage).

6.5.3 Hand tracking

After a hand has been detected in an image, and hand pixels color-segmented using the properties of the human skin, we start tracking it. For tracking the method proposed by Kölsch in [35] has been chosen, which in turn is based on Kanade-Lucas-Tomasi (KLT) features (see Appendix C on page 113), also called "good features to track". KLT features are based on the early work done in [23], and then developed further in [24] and [25]. To increase robustness, the "Flocks of Features" approach to tracking by Kölsch adds two additional properties to simple KLT tracking:

- tracked KLT features never exceed a predetermined maximum distance from the median of all tracked KLT features, and
- tracked KLT features can never be closer to each other than a predetermined minimum distance.

Differently from the application showcased by Kölsch in [35], which is able to track only one hand using just one (monocular) camera, our application 1) implements four fully independent object trackers (four due to each camera tracking up to two hands), and 2) uses stereo disparity for 3D reconstruction of the hand's position in 3D workspace.

We now clarify what is meant by "tracking a hand". After a hand has been detected as explained in Section 6.5.1 in both cameras' views, and hand pixels color-segmented, up to N (for example, 100) KLT features are being collocated on the hand (i.e. on the blob defined by the detected hand's pixels). By averaging in each frame the 2D positions of all of these N features, we obtain a mean (average) position P of the hand being tracked. Therefore the

2D position P is the output of the tracking routine. Since we can have up to two active (tracked) hands, and each hand gives rise to one triangulated 3D position, we can have up to $2 \times 3 = 6$ d.o.f. at our disposal to implement spatial manipulation operations.

3D reconstruction (triangulation)

Finally, with the two corresponding 2D points u, u' tracked (point u in the left camera view, point u' it the right camera view, we can compute, in real time, the global 3D position $\vec{x} = (x, y, z)$ of a hand (either the left or the right hand) in the workspace. For this we use the triangulation method by Hartley and Sturm [20], a fast, non-iterative method that always finds the global optimum. Formulated as a least-squares minimization problem, the method computes image points $\hat{u}, \hat{u}'$ such that

$$d(u,\hat{u})^2 + d(u',\hat{u}')^2 \rightarrow \min, \qquad \hat{u}'^{\tau} F \hat{u} = 0$$

where $d(*,*)$ is the Euclidean distance function and F the fundamental matrix of the stereo rig (see Appendix D for more on the Hartley-Sturm triangulation method). Assuming Gaussian error distribution (tracking of u, u' is noisy because of digitization errors), the points $\hat{u}, \hat{u}'$ are the most likely values for true image correspondences. Since the corresponding rays through $\hat{u}, \hat{u}'$ meet *exactly* in 3D space, we can now find easily $\vec{x}$ (the global position of the hand in the workspace) using any other triangulation method, for example Mid-point triangulation method described in Section 5.4.1 on page 43.

6.5.4 Hand posture recognition

The last step in the CV pipeline is the hand posture recognition, which enables us to implement a simple static gesture recognition. For hand posture recognition, we again use the Viola-Jones method. For this we repeated the training process explained in Section 6.5.1, only with positive samples containing other postures besides `HAND_POSTURE_OPEN`.

6.5.5 Activity diagram

A detailed integrated view (an activity/control flow diagram) of all the CV-related processed described in this section can be seen in Figure 6.6.

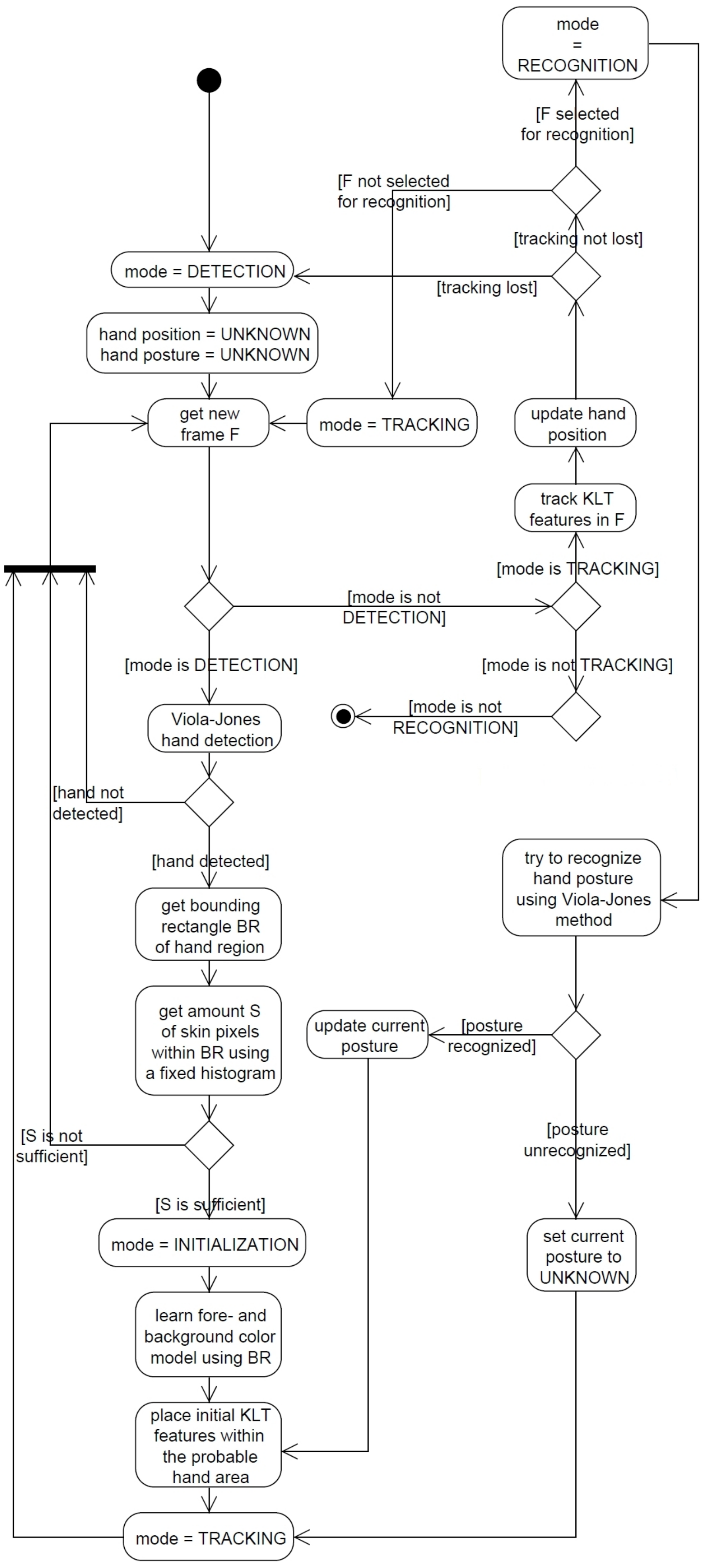

Figure 6.6: Detailed activity diagram for detection, tracking and posture recognition

6.6 Hardware and software configuration

All experiments were done on a personal computer equipped with an 2.66 GHz dual core processor, 2 GB RAM, and two web cameras connected to two dedicated USB 2.0 ports grabbing 30 color frames per second at the resolution of 320 × 240 pixels.

The software application was developed utilizing the C++ language, together with the following libraries:

– OpenGL for low-level 3D graphics rendering
– GLUT for windows handling
– OpenCV computer vision library for low-level image processing and extended Viola-Jones detection method (see Appendix B on page 106)
– A number of libraries were consulted and used to implement hand tracking based on KLT features (see Appendix C on page 113). These include:
    – KLT implementation by Jean-Yves Bouguet, found in the OpenCV library
    – KLT implementation by Stan Birchfield at the Clemson University (www.ces.clemson.edu/˜stb/klt/)
    – KLT implementation by Mathias Kölsch found in the HandVu library (www.movesinstitute.org/˜kolsch/HandVu/HandVu.html)
    – GPU-based KLT implementation by Sudipta Sinha at the University of North Carolina at Chapel Hill (cs.unc.edu/˜ssinha/Research/GPU_KLT/)

6.7 Tests and results

We'll now assess qualitatively estimation accuracy for a hand's position. Since the difference between hand's estimated position and ground truth is difficult to measure for an uninstrumented hand, we give here the figures demonstrating the hand's trajectory in space, from which we can deduce visually the amount of noise present in estimated positions. We trace three simple figures in space with the right hand: a line, a circle and a figure "eight" (Figure 6.7).

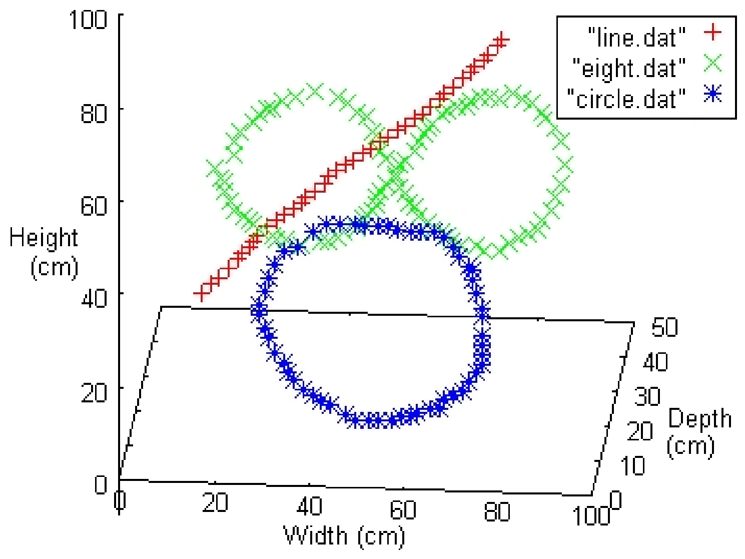

Figure 6.7: 3D plot of estimated hand positions, obtained by tracing a line, a circle and an "eight" in the workspace

6.7.1 Frames per second rates

Using the system described, we achieved tracking-related latencies from 7 to 30ms with just one hand tracked (i.e. with two trackers active), and up to 60ms with both hands tracked (i.e. with all four trackers active). Taking the application as a whole, i.e. taking all the other system processes into consideration, we achieved frame rates from 8 to 15 fps.

6.7.2 Detection performance

In this section the results on detection performance, depending on various training configurations, are presented. Viola-Jones detectors (see Appendix B on page 106) for all three hand postures were trained with various training parameters, and here their performance, relative to these parameters, will be compared.

The parameters which define detector performance are:

- **Number M of positives** (i.e. images that contain at least one instance of the hand in the targeted hand posture)
- **Number N of negatives** (i.e. images that don't contain any instance of the hand in the targeted hand posture)
- **Number K of stages** in the finalized cascaded detector
- **Minimum hit rate** α (for each stage). Note that the overall hit rate for the finalized detector is then α^K.
- **Maximum false alarm rate** β (for each stage). Note that the overall maximum false rate for the finalized detector is then β^K.

The performance parameters being measured (relative to some set containing test images — each of the three postures has its own such set) are:

- **Total hits** Θ — how many positive instances in the images from the test set were correctly detected.
- **Total misses** Γ — how many positive instances in the images from the test set weren't detected.
- **False hits** Ω — how many hits were reported, although there wasn't any positive instance (in the target posture) in the image from the test set, for all images in the set.

Note that by "detector performance" the manner of functioning in terms of hit rates only is presupposed. In other words, the speed of detection is not measured in this work, since in all cases it's good enough (in the range from 10 to 30 ms) to process all frames at the grabbing frame rates.

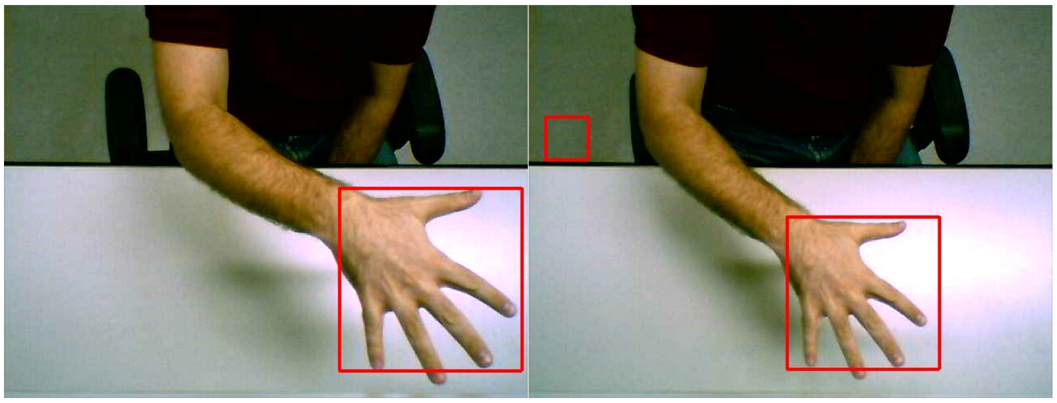

Figure 6.8: A hit (left) and a hit and false hit (right). Posture `HAND_POSTURE_OPEN`

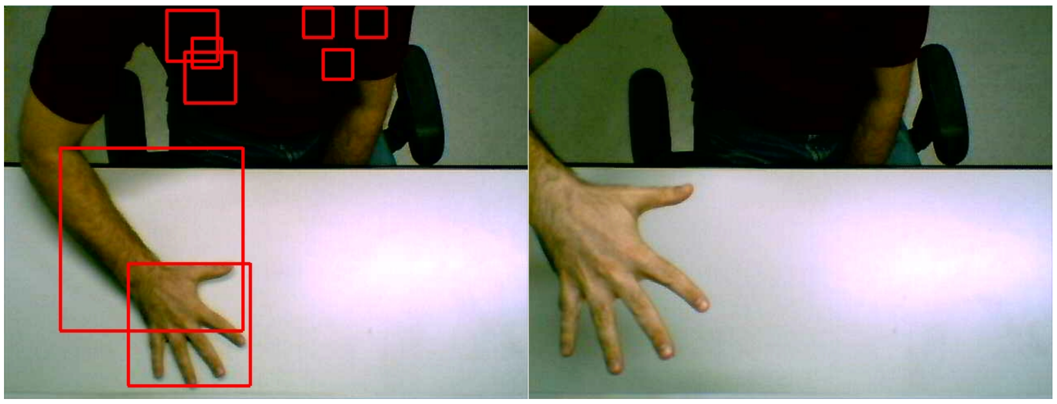

Figure 6.9: A hit and multiple false hits (left), and a miss (right). Posture `HAND_POSTURE_OPEN`

Training times for detectors trained ranged from a couple of hours to several days, using a current personal computer (2.4 GHz dual-core processor) and with 1 GB RAM allocated to the training process.

Detector performance for `HAND_POSTURE_OPEN`

Here we compare the performance of various detectors for the hand posture `HAND_POSTURE_OPEN`. To measure the performance, a set M' of 177 images has been created, some of which contained snapshots of the right hand in the hand posture `HAND_POSTURE_OPEN`, taken at various heights (y-axis) from the desk surface, and at various horizontal (x-axis) and depth (z-axis) locations.

	M	N	K	α	β
index	positives	negatives	number of stages	min. hit rate	max. false alarm rate
1	516	1000	8	0.995	0.5
2	545	1551	3	0.995	0.5
3	545	1551	3	0.995	**0.4**

Table 6.1: Training sets for `HAND_POSTURE_OPEN`

Please note training set #3 which has the same parameters as the set #2 however with maximum false alarm set at 0.4.

	M'	Θ	Γ	Ω
index	test set size	total hits	total misses	false hits
1	177	97 (54.80%)	80 (45.20%)	4 (2.26%)
2	177	154 (87.01%)	23 (12.99%)	813 (459.32%)
3	177	146 (82.46%)	31 (17.54%)	741 (418.64%)

Table 6.2: Detector performance for `HAND_POSTURE_OPEN`

We can see that the training set #1 achieved the highest number of stages and consequently the lowest false hit rate. Sets #2 and #3 achieved required leaf false alarm rate very early in the training phase, which however leads to unacceptably high false hit rates.

Detector performance for `HAND_POSTURE_POINTING`

Here we compare performance of various detectors for the hand posture `HAND_POSTURE_POINTING`. To measure the performance, a set M' of 162 (positive) images has been created some of which contained snapshots of the right hand in the hand posture `HAND_POSTURE_POINTING`, taken at various heights (y-axis) from the desk surface, and at various horizontal (x-axis) and depth (z-axis) locations.

	M	N	K	α	β
index	positives	negatives	number of stages	min. hit rate	max. false alarm rate
1	233	1000	7	0.995	0.5
2	233	1000	10	0.995	**0.3**

Table 6.3: Training sets for `HAND_POSTURE_POINTING`

Training set #2 has the same parameters as the set #2 however with maximum false alarm set at 0.3.

	M'	Θ	Γ	Ω
index	test set size	total hits	total misses	false hits
1	162	115 (70.99%)	47 (29.01%)	1 (0.62%)
2	162	103 (63.58%)	59 (36.42%)	1 (0.62%)

Table 6.4: Detector performance for `HAND_POSTURE_POINTING`

As we can see, training sets #1 and #2 have the same false hit rates, however, for larger test sets, we can expect the set #2 to outperform #1 due to greater number of stages.

Detector performance for `HAND_POSTURE_FIST`

Here we compare performance of various detectors for the hand posture `HAND_POSTURE_FIST`. To measure the performance, a set M' of 191 (positive) images has been created some of which contained snapshots of the right hand in the hand posture `HAND_POSTURE_FIST`, taken at various heights (y-axis) from the desk surface, and at various horizontal (x-axis) and depth (z-axis) locations.

As we can see, training set #2 has the same parameters as the set #1 however with maximum false alarm set at 0.3. Training set #4 has the same parameters as the set #1 however with 1053 negatives instead of 1000, and only four stages. Also, training sets #3 and #4 have an increased number of positives (453). Sets #3 and #4 achieved required leaf false alarm rate very early in the training phase, which led to low number of stages for these detectors.

index	M positives	N negatives	K number of stages	α min. hit rate	β max. false alarm rate
1	222	1000	10	0.995	0.5
2	222	1000	10	0.995	**0.3**
3	453	1000	5	0.995	0.5
4	453	**1053**	**4**	0.995	0.5

Table 6.5: Training sets for `HAND_POSTURE_FIST`

index	M' test set size	Θ total hits	Γ total misses	Ω false hits
1	191	91 (47.64%)	100 (52.36%)	2 (1.05%)
2	191	78 (40.84%)	113 (59.16%)	3 (1.57%)
3	191	55 (28.80%)	136 (71.20%)	312 (163.35%)
4	191	63 (32.98%)	128 (67.02%)	406 (212.57%)

Table 6.6: Detector performance for `HAND_POSTURE_FIST`

We can see that the training set #2 has higher false hit rates than the set #1, however this is probably due to statistical fluctuation. For larger test sets, we can expect the set #2 to outperform #1 due to lower maximum false alarm rate.

An example session while working with the application

In this section, several snapshots of the video taken when working with the prototype application, are shown.

The scene consists of just two simple objects (wire-frame spheres) to be manipulated; by default, the right sphere is already selected, which is indicated by its red color. The left camera's image (of the stereo camera pair) is being rendered as a textured polygon at the bottom of the working volume, depicted here as a simple box delineated by a couple of gray lines.

Finally, the top left corner contains a picture-in-picture movie of hands, taken in real time with a third camera, in order to give a better overview of gestures and operations being performed.

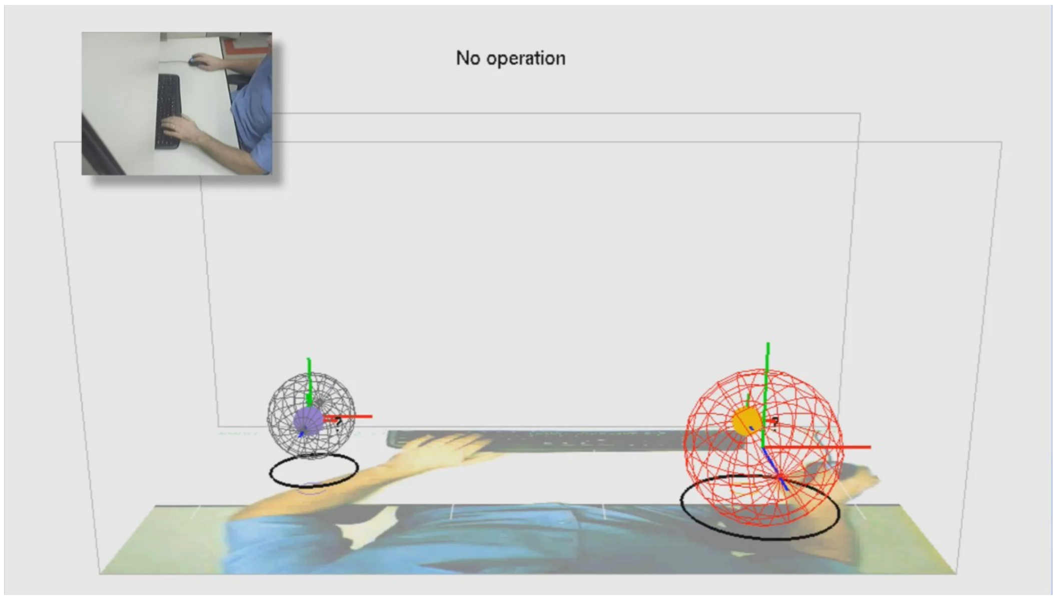

Figure 6.10: The application upon startup. No hand has been detected yet, therefore hands are not being tracked, thus no static gesture is being recognized, thus no manipulation operation is being performed

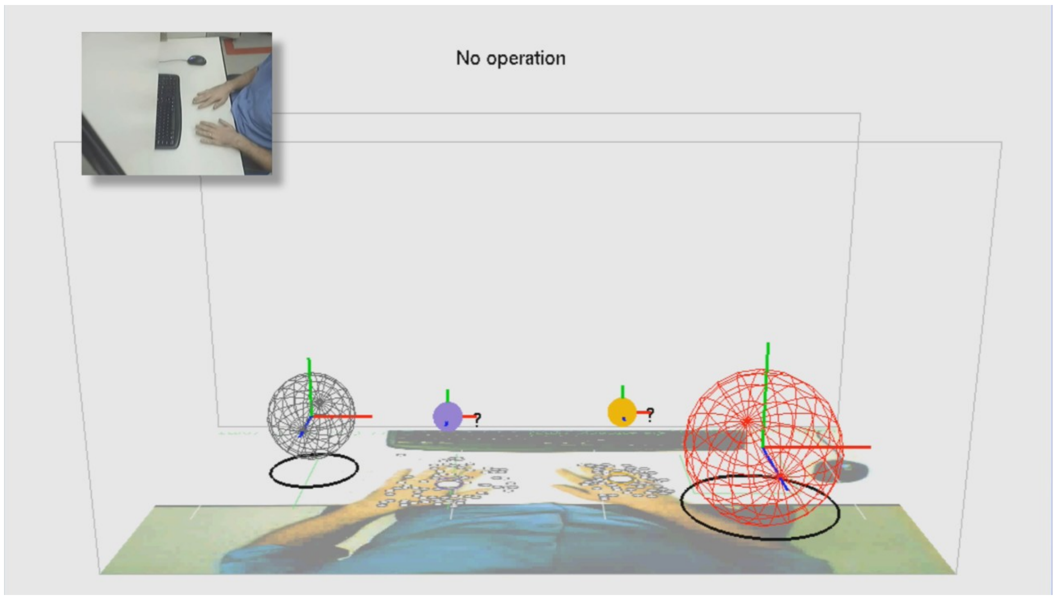

Figure 6.11: Application started to track hands, after both of them assumed posture `HAND_POSTURE_OPEN`. We can see that the application placed two flocks of KLT features on both hands

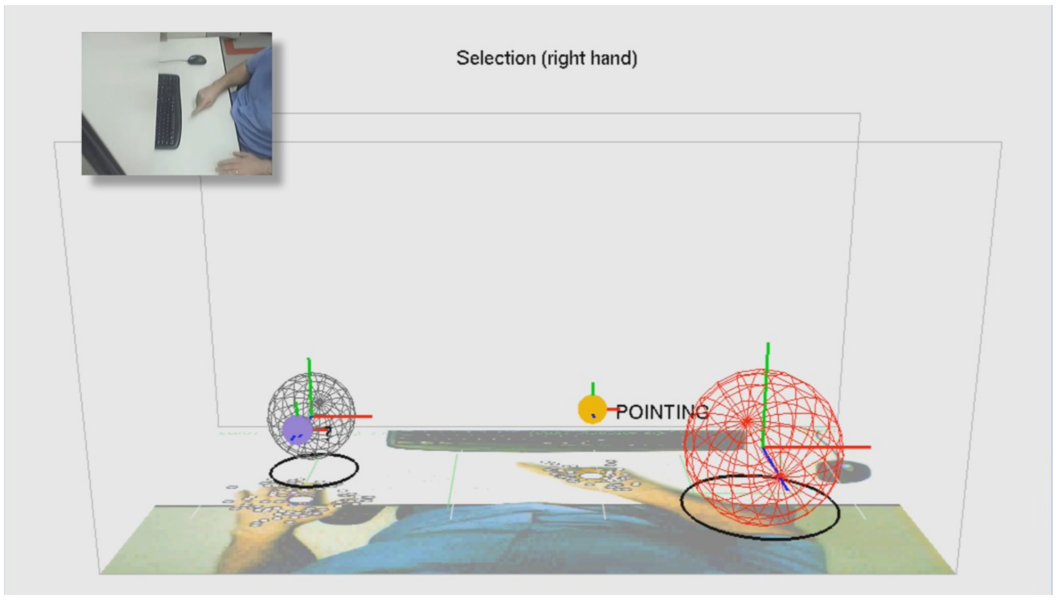

Figure 6.12: The right hand assumed posture `HAND_POSTURE_POINTING`, therefore the application started performing the operation `OP_SELECT` using the right hand

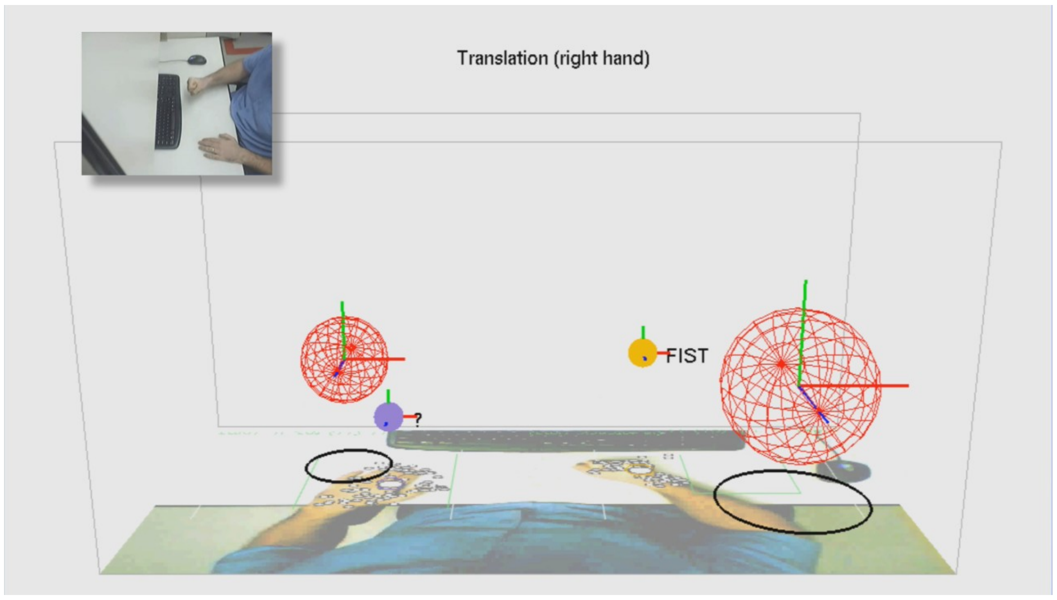

Figure 6.13: The right hand assumed posture `HAND_POSTURE_FIST`, therefore the application started performing the operation `OP_TRANSLATE` using the right hand

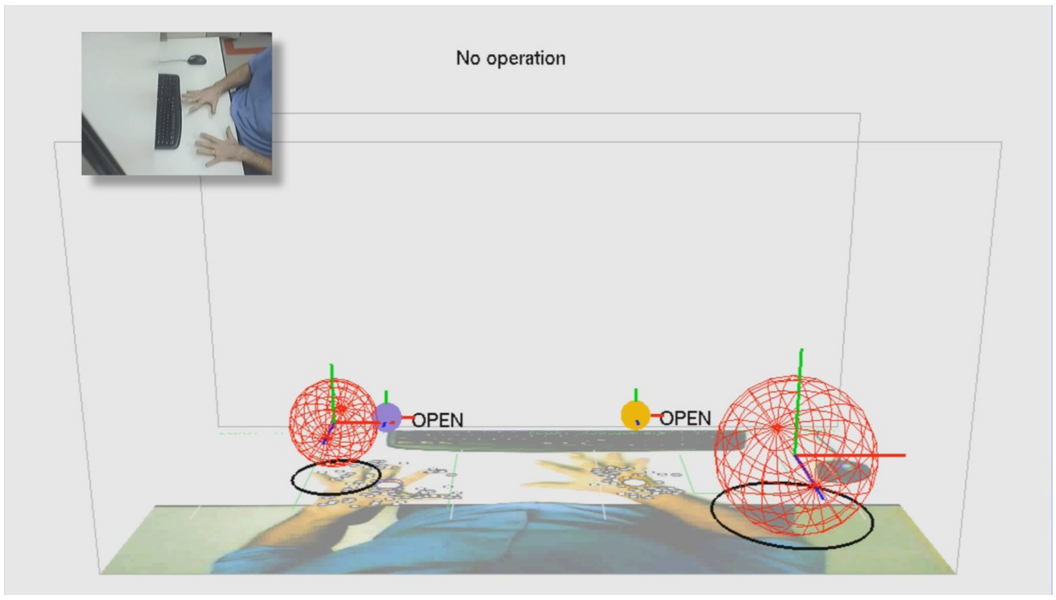

Figure 6.14: Both hands assumed posture `HAND_POSTURE_OPEN`, upon which the previous manipulation operation has been cancelled

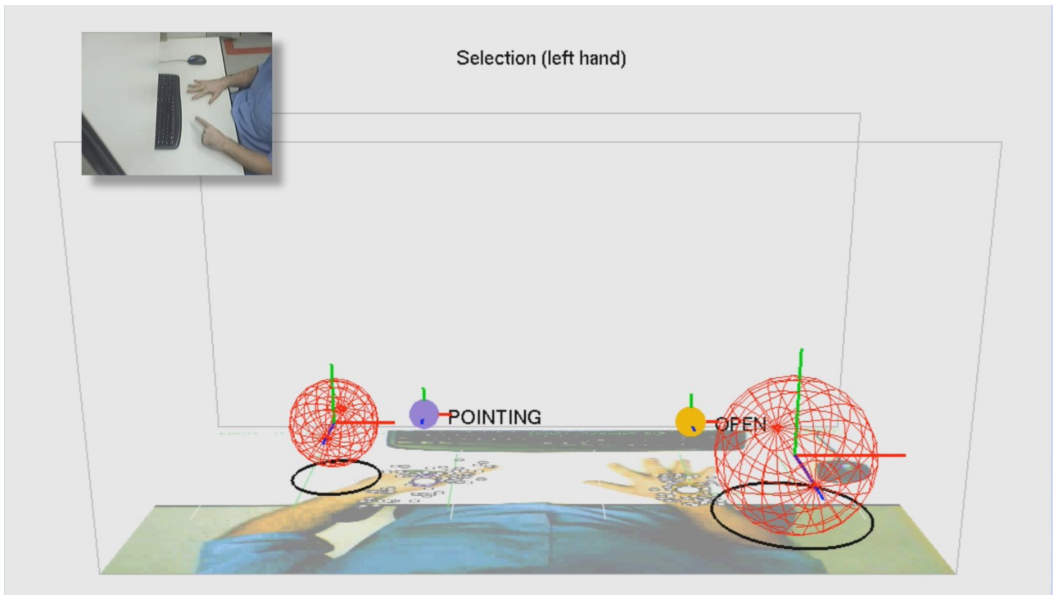

Figure 6.15: The left hand assumed posture `HAND_POSTURE_POINTING`, therefore the application started performing the operation `OP_SELECT` using the left hand

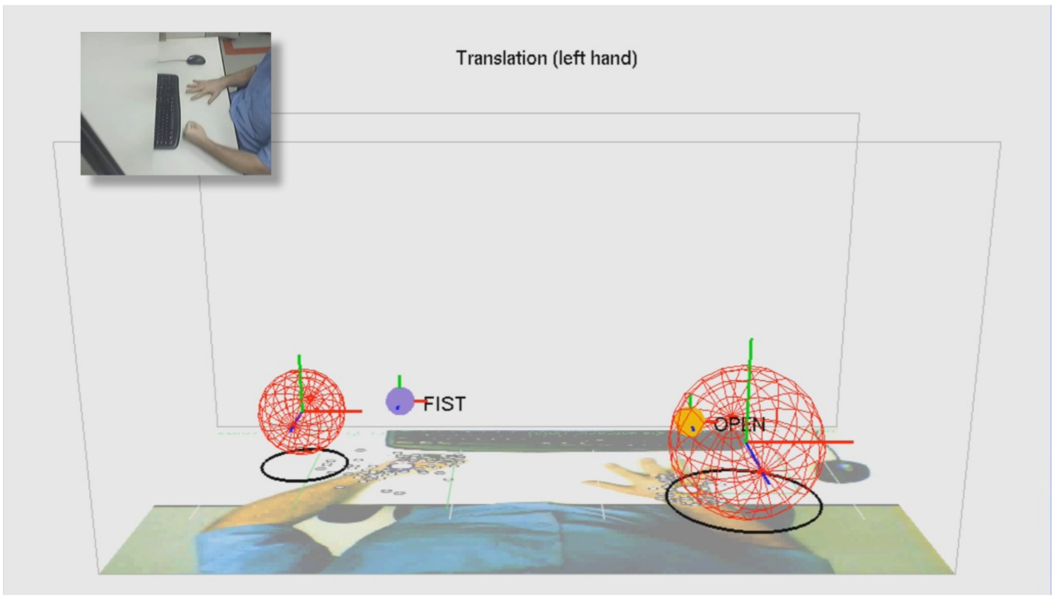

Figure 6.16: The left hand assumed posture HAND_POSTURE_FIST, therefore the application started performing the operation OP_TRANSLATE using the left hand

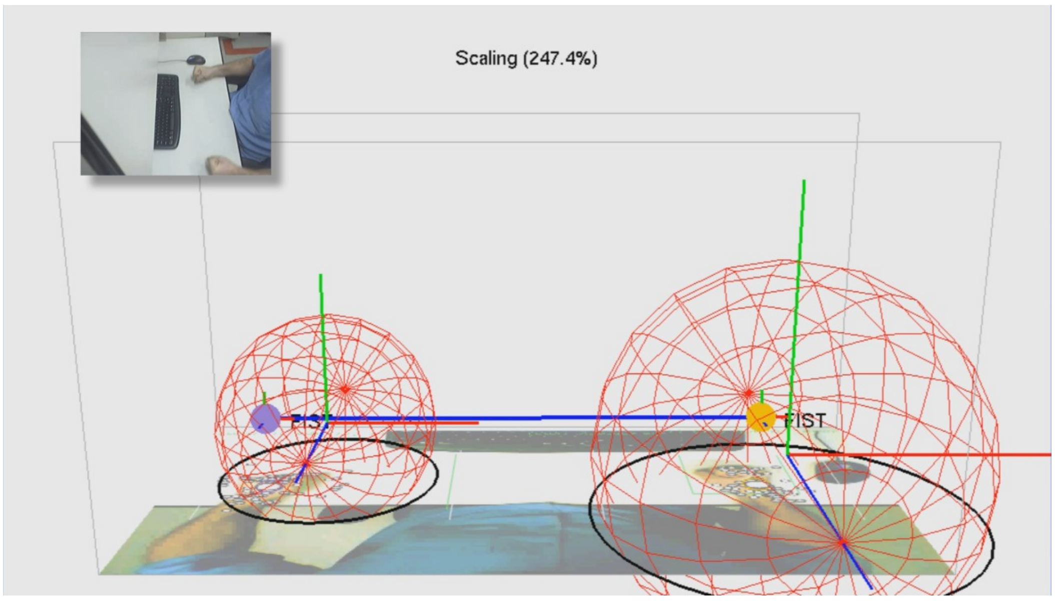

Figure 6.17: Both hands assumed posture HAND_POSTURE_FIST, therefore the application started performing the operation OP_SCALE using both hands

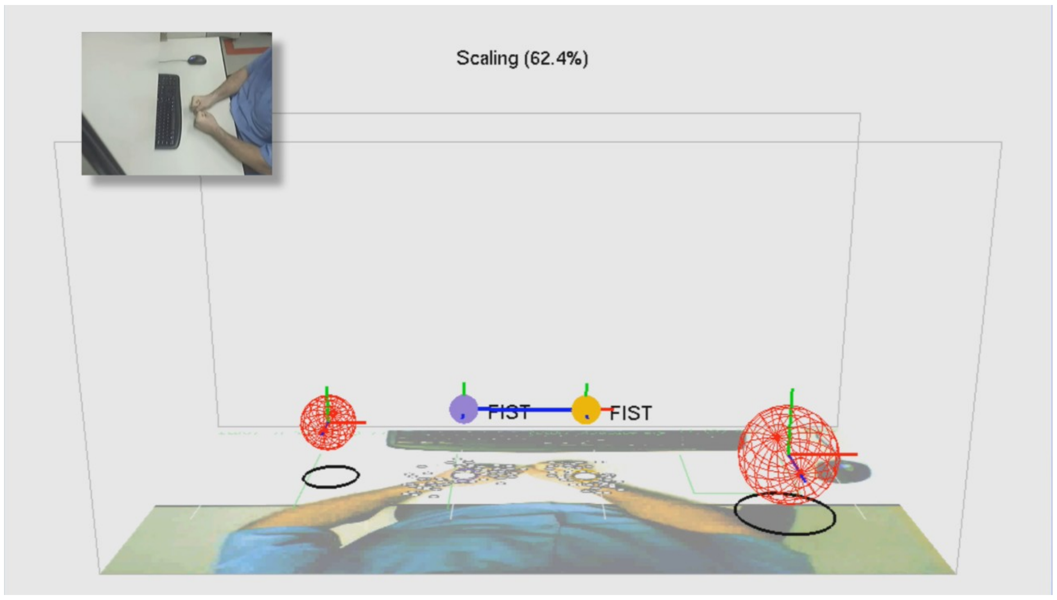

Figure 6.18: Another example of `OP_SCALE`

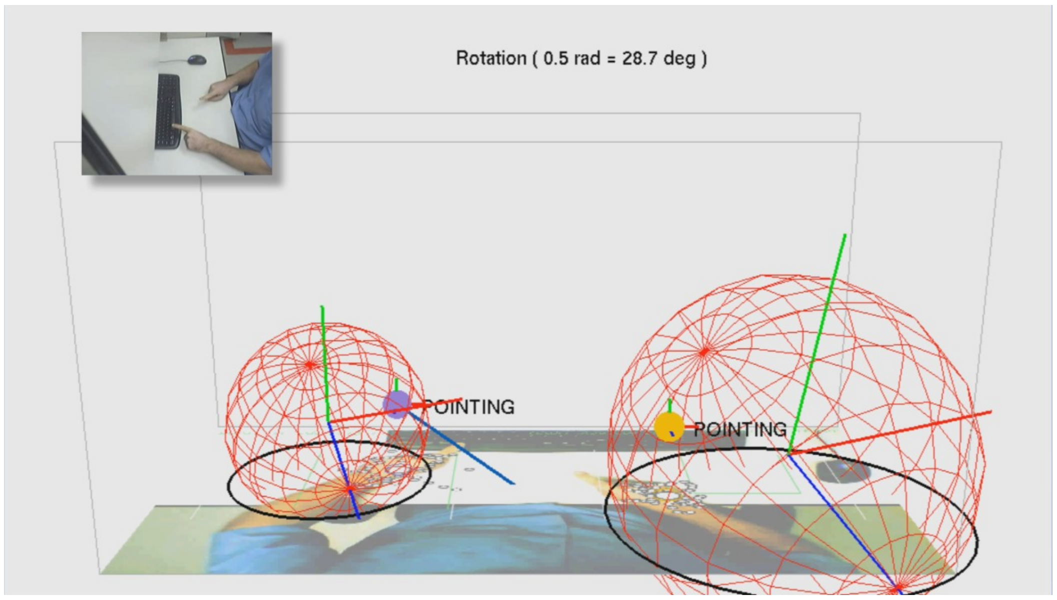

Figure 6.19: Both hands assumed posture `HAND_POSTURE_POINTING`, therefore the application started performing the operation `OP_ROTATE` using both hands

7
Conclusions and future work

In this work, a prototype system that is capable of performing 3D manipulation operations of virtual 3D objects using user's own hands directly, is demonstrated. The system tracks bare (that is, unmarked, uninstrumented) hands of the user and recognizes one-handed and two-handed static hand gestures in a non-contact way using passive computer vision techniques, in order to implement operations for manipulation of virtual 3D objects.

7.1
Contributions

Compared to existing approaches, this work brings about the following contributions:

1. 3D UNMARKED HAND TRACKING — A very inexpensive way to recover continuous 3D position of up to two unmarked hands in real time.
2. SPATIAL INPUT — implementation of a novel $2 \times 3 = 6$ d.o.f. spatial input device, using two stereo trackers based on the "Flocks-of-features" [35] hand tracking and Viola-Jones method [29] applied to hand detection and hand recognition, in order to implement state switching.
3. FREE-HAND SPATIAL MANIPULATION — implementing spatial operations on top of this novel input spatial device, thus enabling the user to manipulate virtual 3D objects using free-hand motions, without any need to put any extra hardware onto his hands.

7.2
Future work

Future work includes the following:

1. Increasing the working volume (workspace) where the user can move his hands by using wide-angle or fish-eye cameras.

2. Expanding the system to work in more diverse environments, for example by pointing the cameras *towards* the user, instead of having the cameras point just downwards as shown in Figure 6.1 on page 56. This would be convenient, for example, for notebook or desktop computer users, who could clip a stereo pair of cameras onto their notebooks' screens.

3. Increasing the expressiveness of direct manipulation by including fingers into the manipulation operations. This probably entails implementing some sort of model-based tracking, that is, using a parametric 3D hand model.

4. Increasing the expressiveness of direct manipulation by including *dynamic gestures* into manipulation operations. This entails implementing some sort of real-time dynamic gesture analysis, for example by using Hidden Markov Models (HMMs).

5. Increasing the robustness of one-hand and two-hand tracking, perhaps by implementing predictive filters like for example the Kalman filter.

6. Enriching the set of manipulation operations, especially by adding more complex, physically based manipulation and deformation operations, like "attract", "repel", "cut", "shear", "distort", "emboss", "drill", "abrade" and similar. Those operations, again, would be performed using just unmarked hands.

7. Developing appropriate spatial and topological data structures for the aforementioned manipulation and deformation operations.

8. Conduct appropriate usability studies.

Bibliography

1 DIETZ, P.; LEIGH, D.. **Diamondtouch: a multi-user touch technology**. In: UIST '01: PROCEEDINGS OF THE 14TH ANNUAL ACM SYMPOSIUM ON USER INTERFACE SOFTWARE AND TECHNOLOGY, p. 219–226, New York, NY, USA, 2001. ACM.

2 KRUEGER, W.; FROEHLICH, B.. **The responsive workbench [virtual work environment]**. Computer Graphics and Applications, IEEE, 14(3):12–15, May 1994.

3 SHNEIDERMAN, B.. **Direct manipulation: A step beyond programming languages**. IEEE Computer, 16(8):57–69, 1983.

4 STURMAN, D. J.. **Whole-hand input**. PhD thesis, MIT, 1992. Supervisor: David Zeltzer.

5 REHG, J. M.; KANADE, T.. **Visual tracking of high DOF articulated structures: an application to human hand tracking**. In: ECCV (2), p. 35–46, 1994.

6 RIJPKEMA, H.; GIRARD, M.. **Computer animation of knowledge-based human grasping**. In: SIGGRAPH '91: PROCEEDINGS OF THE 18TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, p. 339–348, New York, NY, USA, 1991. ACM.

7 YING WU; HUANG, T.. **Hand modeling, analysis and recognition**. Signal Processing Magazine, IEEE, 18(3):51–60, May 2001.

8 NIREI, K.; SAITO, H.; MOCHIMARU, M. ; OZAWA, S.. **Human hand tracking from binocular image sequences**, 1996.

9 PAVLOVIC, V.; SHARMA, R. ; HUANG, T.. **Visual interpretation of hand gestures for human-computer interaction: A review**, 1997.

10 QUEK, F. K. H.. **Toward a vision-based hand gesture interface**. In: VRST '94: PROCEEDINGS OF THE CONFERENCE ON VIRTUAL REALITY SOFTWARE AND TECHNOLOGY, p. 17–31, River Edge, NJ, USA, 1994. World Scientific Publishing Co., Inc.

11 QUEK, F.. **Eyes in the interface**. IVC, 13(6):511–525, August 1995.

12 GUIARD, Y.. **Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a model**, 1987.

13 KENDON, A.. **Current issues in the study of gesture**. The Biological Foundations of Gesture, p. 23–47, 1986.

14 BOWMAN, D. A.; KRUIJFF, E.; LAVIOLA JR., J. J. ; POUPYREV, I.. **3D User Interfaces: Theory and Practice**. Addison-Wesley/Pearson, 2005.

15 POUPYREV, I.; BILLINGHURST, M.; WEGHORST, S. ; ICHIKAWA, T.. **The go-go interaction technique: non-linear mapping for direct manipulation in vr**. In: UIST '96: PROCEEDINGS OF THE 9TH ANNUAL ACM SYMPOSIUM ON USER INTERFACE SOFTWARE AND TECHNOLOGY, p. 79–80, New York, NY, USA, 1996. ACM.

16 STOAKLEY, R.; CONWAY, M. J. ; PAUSCH, R.. **Virtual reality on a WIM: Interactive worlds in miniature**. In: PROCEEDINGS CHI'95, 1995.

17 BOWMAN, D. A.; HODGES, L. F.. **An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments**. In: SYMPOSIUM ON INTERACTIVE 3D GRAPHICS, p. 35–38, 182, 1997.

18 MINE, M.. **Virtual environment interaction techniques**. Technical Report TR93-005, UNC Chapel Hill, Dept of Computer Science, North Carolina, USA, 1995.

19 ZHANG, Z.. **A flexible new technique for camera calibration**. IEEE Trans. Pattern Anal. Mach. Intell., 22(11):1330–1334, 2000.

20 HARTLEY, R.; STURM, P.. **Triangulation**. 68(2):146–157, November 1997.

21 KAKUMANU, P.; MAKROGIANNIS, S. ; BOURBAKIS, N.. **A survey of skin-color modeling and detection methods**. Pattern Recogn., 40(3):1106–1122, 2007.

22 HARRIS, C.; STEPHENS, M.. **A combined corner and edge detector**. Alvey Vision Conference Proceedings, p. 147–152, 1988.

23 LUCAS, B.; KANADE, T.. **An iterative image registration technique with an application to stereo vision**, 1981.

24 TOMASI, C.; KANADE, T.. **Detection and tracking of point features.** Technical Report CMU-CS-90-166, Carnegie Mellon University, USA, Apr. 1991.

25 SHI, J.; TOMASI, C.. **Good features to track.** In: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR'94), Seattle, June 1994.

26 LOWE, D.. **Distinctive image features from scale-invariant keypoints.** In: INTERNATIONAL JOURNAL OF COMPUTER VISION, volumen 20, p. 91–110, 2003.

27 BAY, H.; TUYTELAARS, T. ; GOOL, L.. **Surf: Speed-up robust features.** In: EUROPEAN CONFERENCE ON COMPUTER VISION, p. 404–417, 2006.

28 MIKOLAJCZYK, K.; SCHMID, C.. **A performance evaluation of local descriptors.** Transactions on Pattern Analysis and Machine Intelligence, 27(10):1615–1630, Oct. 2005.

29 VIOLA, P.; JONES, M.. **Robust real-time object detection.** International Journal of Computer Vision, 2002.

30 FREUND, Y.; SCHAPIRE, R. E.. **A decision-theoretic generalization of on-line learning and an application to boosting.** In: EUROPEAN CONFERENCE ON COMPUTATIONAL LEARNING THEORY, p. 23–37, 1995.

31 KÖLSCH, M.; TURK, M.. **Robust hand detection.** In: FGR, p. 614–619, 2004.

32 BRADSKI, G. R.. **Computer vision face tracking for use in a perceptual user interface.** Intel Technology Journal, (Q2):15, 1998.

33 ISARD, M.; BLAKE, A.. **Condensation – conditional density propagation for visual tracking.** International Journal of Computer Vision, 29(1):5–28, 1998.

34 ISARD, M.; BLAKE, A.. **ICONDENSATION: Unifying low-level and high-level tracking in a stochastic framework.** Lecture Notes in Computer Science, 1406:893–908, 1998.

35 KOLSCH, M.; TURK, M.. **Fast 2d hand tracking with flocks of features and multi-cue integration.** In: CVPRW '04: PROCEEDINGS OF THE 2004 CONFERENCE ON COMPUTER VISION AND PATTERN

RECOGNITION WORKSHOP (CVPRW'04) VOLUME 10, p. 158, Washington, DC, USA, 2004. IEEE Computer Society.

36 SHIMADA, N.; KIMURA, K. ; SHIRAI, Y.. **Real-time 3-d hand posture estimation based on 2-d appearance retrieval using monocular camera.** ratfg-rts, 00:0023, 2001.

37 ATHITSOS, V.; SCLAROFF, S.. **3d hand pose estimation by finding appearance-based matches in a large database of training views.** Technical Report BU-CS-TR-2001-021, Computer Science Department, Boston University, Boston, USA, 2001.

38 SUTHERLAND, I.. **Sketchpad: A man-machine graphical communication system.** PhD thesis, Massachusetts Institute of Technology, January 1963.

39 NEWMAN, W. M.. **An experimental program for architectural design.** The Computer Journal, 9(1):21–26, May 1966.

40 NEWMAN, W. M.. **A system for interactive graphical programming.** In: AFIPS SPRING JOINT COMPUTER CONFERENCE, p. 47–43, 1968.

41 CLARK, J. H.. **Designing surfaces in 3-d.** Commun. ACM, 19(8):454–460, 1976.

42 KAY, A.; GOLDBERG, A.. **Personal dynamic media.** Computer, 10(3):31–41, 1977.

43 BOLT, R. A.. **"put-that-there": Voice and gesture at the graphics interface.** In: SIGGRAPH '80: PROCEEDINGS OF THE 7TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, p. 262–270, New York, NY, USA, 1980. ACM.

44 NEGROPONTE, N.. **The media room.** Technical Report Report for ONR and DARPA, Cambridge, MA, USA, 12, 1978.

45 SACHS, E.; STOOPS, D. ; ROBERTS, A.. **3-draw: a three dimensional computer aided design tool.** Systems, Man and Cybernetics, 1989. Conference Proceedings., IEEE International Conference on, p. 1194–1196 vol.3, 14-17 Nov 1989.

46 SACHS, E.; ROBERTS, A. ; STOOPS, D.. **3-draw: A tool for designing 3d shapes.** IEEE Computer Graphics and Applications, 11(6):18–26, 1991.

47 KRUEGER, M.. **Artificial Reality II**. Addison-Wesley: Reading, MA, 1991., second edition, 1991.

48 KRUEGER, M. W.; GIONFRIDDO, T. ; HINRICHSEN, K.. **Videoplace — an artificial reality**. SIGCHI Bull., 16(4):35–40, 1985.

49 ROBINETT, W.; HOLLOWAY, R.. **Implementation of flying, scaling and grabbing in virtual worlds**. In: SI3D '92: PROCEEDINGS OF THE 1992 SYMPOSIUM ON INTERACTIVE 3D GRAPHICS, p. 189–192, New York, NY, USA, 1992. ACM.

50 SU, S. A.; FURUTA, R.. **A specification of 3d manipulation in virtual environments**, 1994.

51 STRAUSS, P. S.; CAREY, R.. **An object-oriented 3d graphics toolkit**. SIGGRAPH Comput. Graph., 26(2):341–349, 1992.

52 CONNER, D. B.; SNIBBE, S. S.; HERNDON, K. P.; ROBBINS, D. C.; ZELEZNIK, R. C. ; VAN DAM, A.. **Three-dimensional widgets**. In: PROCEEDINGS OF THE 1992 SYMPOSIUM ON INTERACTIVE 3D GRAPHICS, SPECIAL ISSUE OF COMPUTER GRAPHICS, VOL. 26, p. 183–188, 1992.

53 BUTTERWORTH, J.; DAVIDSON, A.; HENCH, S. ; OLANO, M. T.. **3dm: a three dimensional modeler using a head-mounted display**. In: SI3D '92: PROCEEDINGS OF THE 1992 SYMPOSIUM ON INTERACTIVE 3D GRAPHICS, p. 135–138, New York, NY, USA, 1992. ACM.

54 HINCKLEY, K.; PAUSCH, R.; GOBLE, J. C. ; KASSELL, N. F.. **A survey of design issues in spatial input**. In: UIST '94: PROCEEDINGS OF THE 7TH ANNUAL ACM SYMPOSIUM ON USER INTERFACE SOFTWARE AND TECHNOLOGY, p. 213–222, New York, NY, USA, 1994. ACM.

55 SHAW, C.; GREEN, M.. **THRED: A two-handed design system**. Multimedia Systems, 5(2):126–139, 1997.

56 MURAKAMI, T.; NAKAJIMA, N.. **Direct and intuitive input device for 3-d shape deformation**. In: CHI '94: PROCEEDINGS OF THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, p. 465–470, New York, NY, USA, 1994. ACM.

57 LIANG, J.; GREEN, M.. **Jdcad: a highly interactive 3d modeling system**. Computers & Graphics, 18(4):499–506, 1994.

58 DEERING, M. F.. **Holosketch: a virtual reality sketching/animation tool**. ACM Trans. Comput.-Hum. Interact., 2(3):220–238, 1995.

59 GRIMM, C.; PUGMIRE, D.. **Visual interfaces for solids modeling**. In: ACM SYMPOSIUM ON USER INTERFACE SOFTWARE AND TECHNOLOGY, p. 51–60, 1995.

60 MAPES, D. P.; MOSHELL, J. M.. **A two-handed interface for object manipulation in virtual environments**. Presence, p. 403–416, 1995.

61 MINE, M. R.. **Working in a virtual world: Interaction techniques used in the chapel hill immersive modeling program**. Technical Report TR96-029, 12, 1996.

62 ZELEZNIK, R. C.; HERNDON, K. P. ; HUGHES, J. F.. **SKETCH: An interface for sketching 3D scenes**. In: Rushmeier, H., editor, SIGGRAPH 96 CONFERENCE PROCEEDINGS, p. 163–170. Addison Wesley, 1996.

63 MINE, M. R.; BROOKS, JR., F. P. ; SEQUIN, C. H.. **Moving objects in space: Exploiting proprioception in virtual-environment interaction**. Computer Graphics, 31(Annual Conference Series):19–26, 1997.

64 CUTLER, L. D.; FROEHLICH, B. ; HANRAHAN, P.. **Two-handed direct manipulation on the responsive workbench**. In: SYMPOSIUM ON INTERACTIVE 3D GRAPHICS, p. 107–114, 191, 1997.

65 ZHAI, S.. **User performance in relation to 3d input device design**. SIGGRAPH Comput. Graph., 32(4):50–54, 1998.

66 FJELD, M.; BICHSEL, M. ; RAUTERBERG, M.. **Build-it: An intuitive design tool based on direct object manipulation**. In: PROCEEDINGS OF THE INTERNATIONAL GESTURE WORKSHOP ON GESTURE AND SIGN LANGUAGE IN HUMAN-COMPUTER INTERACTION, p. 297–308, London, UK, 1998. Springer-Verlag.

67 FORSBERG, A.; LAVIOLA, J. ; ZELEZNIK, R.. **Ergodesk: A framework for two and three dimensional interaction at the activedesk**, 1998.

68 NISHINO, H.; UTSUMIYA, K. ; KORIDA, K.. **3d object modeling using spatial and pictographic gestures**. In: VRST, p. 51–58, 1998.

69 SCHKOLNE, S.; SCHRODER, P.. **Surface drawing**. Technical Report CS-TR-99-03, Pasadena, CA, USA, 1999.

70 SCHKOLNE, S.; PRUETT, M. ; SCHRÖDER, P.. **Surface drawing: creating organic 3d shapes with the hand and tangible tools**. In: CHI '01: PROCEEDINGS OF THE SIGCHI CONFERENCE ON HUMAN

FACTORS IN COMPUTING SYSTEMS, p. 261–268, New York, NY, USA, 2001. ACM.

71 LEIBE, B.; STARNER, T.; RIBARSKY, W.; WARTELL, Z.; KRUM, D.; SINGLETARY, B. ; HODGES, L. F.. **The perceptive workbench: Toward spontaneous and natural interaction in semi-immersive virtual environments**. In: VR, p. 13–20, 2000.

72 WU, M.; BALAKRISHNAN, R.. **Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays**. In: UIST '03: PROCEEDINGS OF THE 16TH ANNUAL ACM SYMPOSIUM ON USER INTERFACE SOFTWARE AND TECHNOLOGY, p. 193–202, New York, NY, USA, 2003. ACM.

73 BUCHMANN, V.; VIOLICH, S.; BILLINGHURST, M. ; COCKBURN, A.. **Fingartips: gesture based direct manipulation in augmented reality**. In: GRAPHITE, p. 212–221, 2004.

74 PRATINI, E.. **Modeling with gestures: Sketching 3d virtual surfaces and objects using hands formation and movements**. In: ASCAAD INTERNATIONAL CONFERENCE, p. 35–41, 2004.

75 KIM, H.; ALBUQUERQUE, G.; HAVEMANN, S. ; W. FELLNER, D.. **Tangible 3d: Immersive 3d modeling through hand gesture interaction**. Technical Report TUBS-CG-2004-07, Institute of Computer Graphics, University of Technology, Braunschweig, Germany, 2004.

76 LINGRAND, D.; RENEVIER, P.; PINNA-DERY, A.-M.; CREMASCHI, X.; LION, S.; ROUEL, J.-G.; JEANNE, D.; CUISINAUD, P. ; SOULA, J.. **Gestaction3d: a platform for studying displacements and deformation of 3d objects using hands**. In: INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN OF USER INTERFACES (CADUI), p. 105–114, 2006.

77 DACHSELT, R.; HINZ, M.. **Three-dimensional widgets revisited - towards future standardization**. In: PROCEEDINGS OF THE WORKSHOP 'NEW DIRECTIONS IN 3D USER INTERFACES', 2005.

78 DACHSELT, R.; HUEBNER, A.. **Virtual environments: Three dimensional menus: A survey and taxonomy**. Comput. Graph., 31(1):53–65, 2007.

79 TRUYENQUE, M. A. Q.. **A computer vision application that uses hand gestures for human-computer interaction**. Master's thesis, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil, March 2005.

80 BETTIO, F.; GIACHETTI, A.; GOBBETTI, E.; MARTON, F. ; PINTORE, G.. **A practical vision based approach to unencumbered direct spatial manipulation in virtual worlds**. In: EUROGRAPHICS ITALIAN CHAPTER CONFERENCE, Conference held in Trento, Italy, February 2007. Eurographics Association.

81 AMIT, Y.; GEMAN, D. ; WILDER, K.. **Joint induction of shape features and tree classifiers**, 1997.

82 Solla, S. A.; Leen, T. K. ; Müller, K.-R., editors. **Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]**. The MIT Press, 2000.

83 ROWLEY, H. A.; BALUJA, S. ; KANADE, T.. **Neural network-based face detection**. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(1):23–38, 1998.

84 SCHNEIDERMAN, H.; KANADE, T.. **A statistical method for 3d object detection applied to faces and cars**. Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on, 1:746–751 vol.1, 2000.

85 SUNG, K.-K.; POGGIO, T.. **Example-based learning for view-based human face detection**. IEEE Trans. Pattern Anal. Mach. Intell., 20(1):39–51, 1998.

86 LIENHART, R.; MAYDT, J.. **An extended set of haar-like features for rapid object detection**. In: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, p. 900–903, Rochester, USA, September 2002. IEEE.

87 HARTLEY, R. I.; ZISSERMAN, A.. **Multiple View Geometry in Computer Vision**. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

A
Timeline of manipulation-related research

The following chronologically sorted and annotated list contains publications that are related to direct spatial manipulation of virtual geometric objects. The criteria to include a publication in this list are:

1. The publications must deal with manipulation of virtual geometrical three-dimensional objects, and/or their 2D sections or projections.
2. The spatial 3D manipulation must be initiated and executed through an interface involving human hands, fingers and/or arms, using any input device.

The second criterion above practically means that both input done vision-based tracking of human hands, as well as input done using devices like mice, pens, tables, datagloves and all sorts of many-d.o.f. devices will be included. The list is not meant to be exhaustive, but *representative* of prior work related to manipulation of 3D geometry, using hands.

A.1 Pre-1980s

1963

– PhD thesis [38] by Ivan Sutherland that influenced almost everything we know and think about computing, see Figures A.1 and A.2. The application (Sketchpad, also known as "Robot Draftsman") that Sutherland developed as a part of his thesis used a light-pen to draw and manipulate (grab, copy and move) 2D shapes on the screen, changing their sizes and using constraints. It can be considered to be the precursor to modern Computer-Aided Design (CAD) applications. This thesis influenced the development of Xerox Star workstation which later on influenced the development of Mac OS, Windows and X-Windows operating systems.

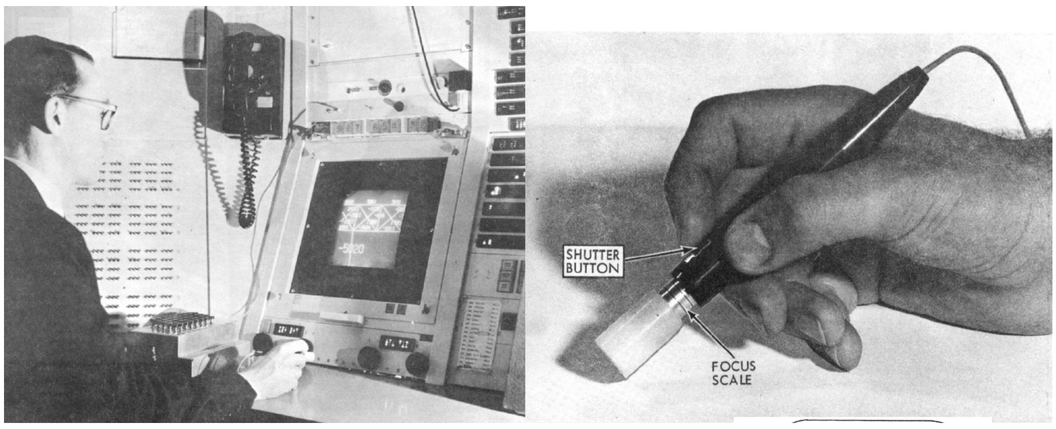

Figure A.1: Sutherland's Sketchpad in use (Lincoln TX-2 console, lightpen)

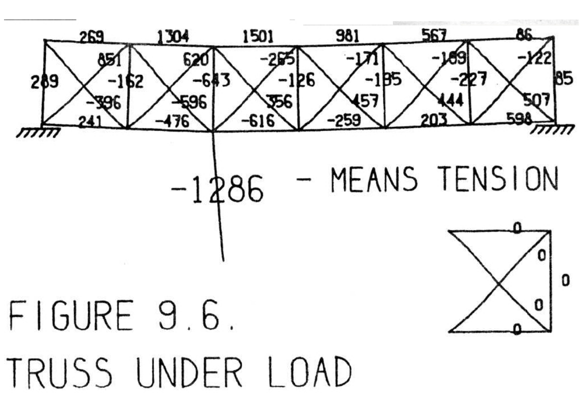

Figure A.2: Example of a drawing and calculation made in Sutherland's Sketchpad: truss load

1966

– [39] William Newman's early system for architectural design. Based on a PDP-7 computer, Type 340 Display and a type of light pen. The system uses modular building blocks. The paper describes the approaches used to organizing the display list for efficient manipulation and an algorithm for computing areas of enclosed spaces.

1968

– [40] William Newman's "reaction handler", based on tablet and stylus. Provided direct manipulation of graphical shapes. Introduced "light handles", a type of graphical potentiometer, which could be considered the first "manipulation widget".

1976

– [41] "Designing surfaces in 3-D" system [41] by another pioneer in the field of computer graphics (and co-founder of Silicon Graphics, Netscape Communications and some other companies) James H. Clark. One of the earliest interactive design systems for drawing free form 3D (parametric) surfaces, utilizing HMD. The mechanically-tracked HMD utilized in the Clark's system was designed by Ivan Sutherland. Surfaces can be controlled by manipulating control points on the associated wireframe grid (see Figure A.4). The system used a 3-D wand that computed its position my measuring its distance to the ceiling.

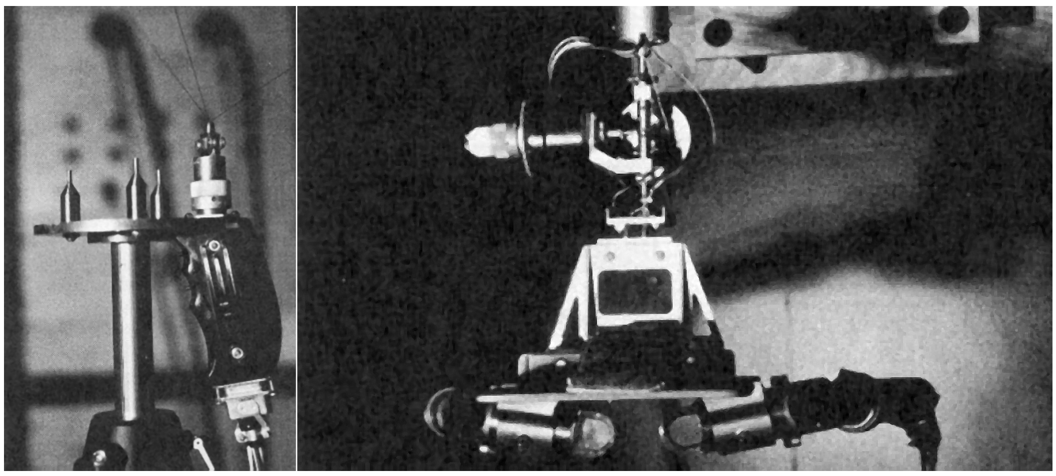

Figure A.3: James H. Clark's system: 3D-wand (left) and HMD armature (right)

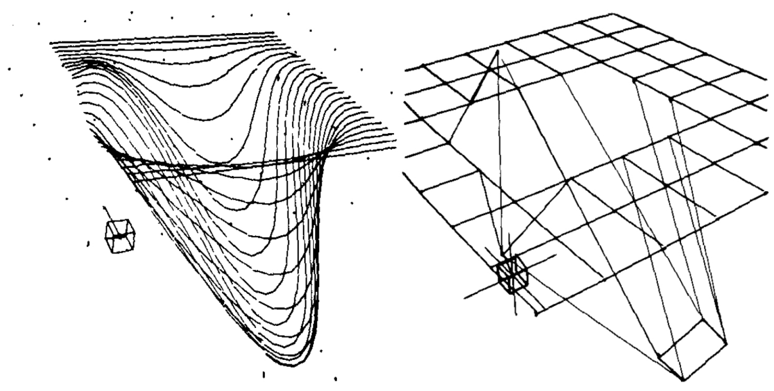

Figure A.4: James H. Clark's system: 3D surface being edited (left) and its grid of control points (right)

1977

- [42] Alan Kay's (Xerox PARC) gives a vision of direct manipulation interfaces for everyone, using the Smalltalk language and Dynabook.

A.2 1980s

1980

- Bolt's "Put-That-There" system [43]. Uses hand gestures together with speech input. Manipulates simple shapes on a large, wall-sized screen. The user can, for example, point to a shape, and then utter a command to modify the shape. Uses a "Media Room" by Negroponte [44], an enclosure which supplants the CRT display and turns the whole room into a sort of input-output space. The user sits in a modified chair: each arm of the chair has a small joystick sensitive to direction and pressure. Besides them there are two small touch-sensitive pads. On either sides of the chair are located TV monitors, whose cathode tube's surface has been coated with a touch-sensitive surface. Commands: CREATE ("create a blue square there"), MOVE ("move the blue triangle to the right of the green square"), MAKE THAT ("make that blue triangle smaller"), DELETE ("delete the large blue circle"), CALL THAT ("call that blue square the calendar"). Six-d.o.f. Polheus tracker epoxied in a cube, attached to user's wrist via a watchband.

1983

- [3] Ben Shneiderman coins the expression "direct manipulation" and defines its constituent components as well as psychological foundations. Describes graphically-based interaction, visibility of objects, incremental action and rapid feedback.

1987

- [12] Guiard gives a theoretical framework for the study of asymmetry in the context of human bimanual action. Most skilled manual activities involve two hands playing different roles. The two hands represent two motors, which cooperate with one another as if they were assembled in series, thereby forming a *kinematic chain*. In right-handed people, motion produced by the right hand tends to be articulated with motion produced by the left.

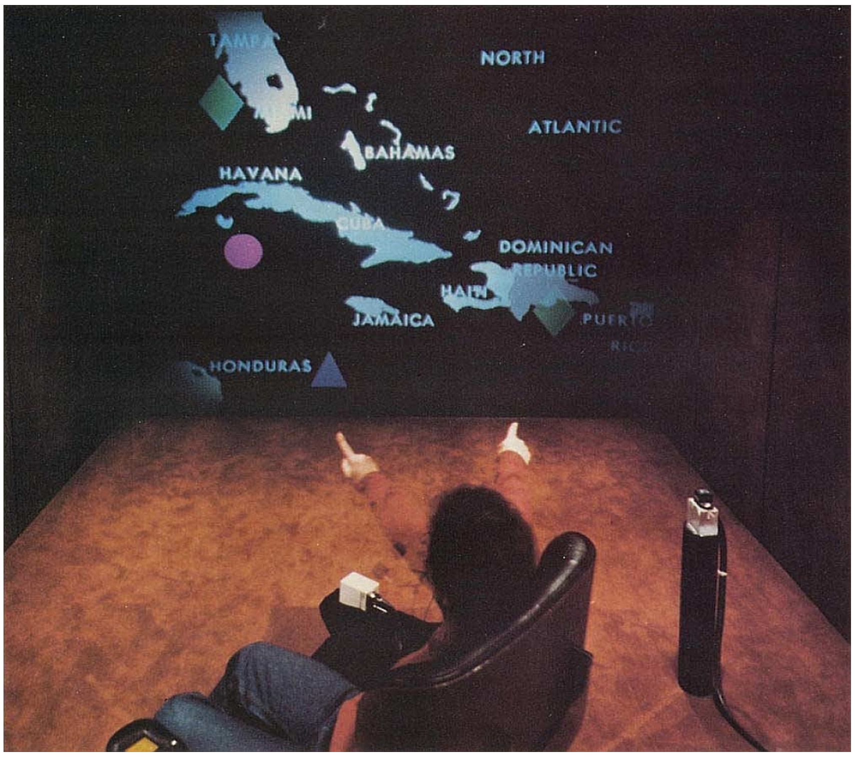

Figure A.5: "Put-That-There" system by Bolt: manipulating shapes on the wall-sized screen. The user currently points at the circular shape

1989

– 3-Draw, a 3D computer-aided design tool [45], [46], see Figure A.6. Output image is shown on a conventional non-stereo display. Capable of drawing complex free-form shapes. Uses two 6-d.o.f. sensors - one sensor is a configurable 3D drawing and editing tool, and the other sensor controls an object's position and orientation. One hand holds a tracked palette that acts as a movable reference frame in modeling space. The other hand holds a stylus and draws 2D curves on the palette. This combination thus resulted in curves in 3D space. Users found the interface natural and quick. Simultaneous use of two hands provided kinesthetic feedback that enabled users to feel as though they were holding the objects displayed on the screen.

A.3 1990s

1991

– VIDEODESK [47] — it consists of a large surface over which the user moves his fingers, hands and arms, see Figure A.7. The system uses over-

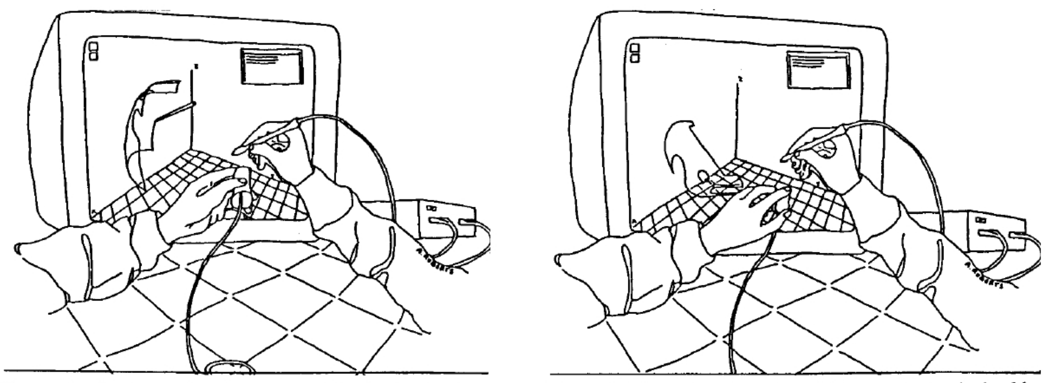

Figure 1. Designer sketches automobile fender in three dimensions using 3-Draw tools.

Figure 2. New orientation of model interactively obtained by moving left hand.

Figure A.6: 3-Draw by Sachs *et al* — based on two 6-d.o.f. sensors and a conventional non-stereo display.

head video cameras to track the appearance and 2D hand position and to detect image features such as the hand, fingers, and their orientation. The system uses a large horizontal table with a bright background (for easier detection and segmentation of hands). The geometry being manipulated is being modeled using splines. Control points on these splines can then be controlled using index fingers and thumbs of both hands. Similarly, using both index fingers it is possible to draw a circle on the screen.

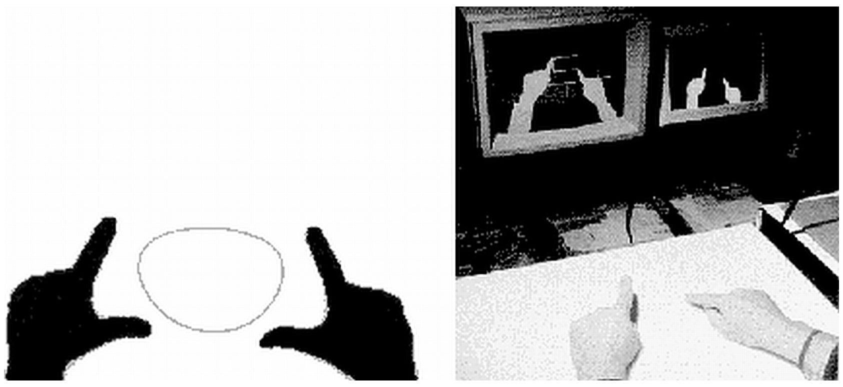

Figure A.7: Krueger's VIDEODESK. Splines are controlled by fingertip positions.

A related, older system by the same author is VIDEOPLACE [48] which tracks the whole body, not just hands. That is, the user uses his entire body as input to the system. It was installed as a part of a large installation on the Computers and Art exhibit at IBM building in New York, US. The author has linked this environment with VIDEODESK.

1992

- [49] Hand gestures initiate and terminate fundamental actions (translate, scale, rotate) that change the state of virtual world. Manual input device may be an instrumented glove or or a hand-held device with buttons. Specifies transforms for grabbing an object, flying, scaling the world.
- [50] A specification of 3D manipulation operations, based on hand gestures. Three basic gestures (touching, pointing and gripping) are defined. *Touching* is a simple gesture with no extra information; by tracking the logical hand in virtual 3D space, and using collision detection, it is said that the hand "touched" an object if the hand collided with the object. *Pointing* requires an extended index finger; its fingertip position then defines the starting point of the pointing, and the orientation of the index finger is the pointing direction. Using this information it is possible to determine the 3D object pointed at. Finally, *gripping* is defined as an analog of the click-and-drag operation in classical 2D WIMP interfaces.
- [51] Describes what is to become Open Inventor. Describes *manipulators* (trackball, one-axis scale, jack, handle box, spot light, directional light, one-axis translate).
- Conner et al [52] describe three-dimensional *widgets* (Figure [52]). Gives precise state diagrams. Virtual sphere, handles, snapping, color picker, rack, cone tree.

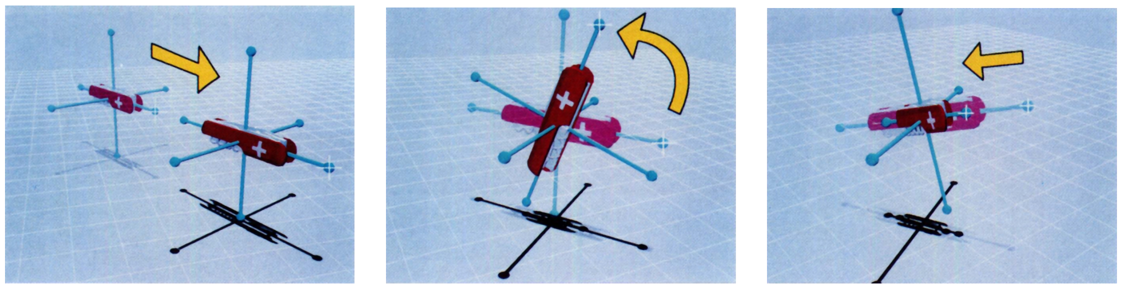
(a) (b) (c)

Figure A.8: Widgets by Conner et al. Translating a knife along its x axis (a), rotating a knife along an axis (b), and scaling a knife along an axis (c)

- [53] 3DM (Three-Dimensional Modeler). An interactive surface-modeling system. It uses a stereo HDM, and one single bat with 6 d.o.f. User can create 3D objects. Walking, flying, grabbing the world, scaling the user.

1994

– [54] A survey of design issues for developing effective free-space 3D user interfaces. People do not innately understand 3D reality, but rather they experience it. Concepts that facilitate 3D space perception: spatial references, relative vs. absolute gestures, two-handed interaction, multisensory feedback, physical constraints, head tracking techniques. Coarse vs. precise positioning tasks: gridding and snapping. Dynamics and size of the working volume of user's hands. Use of mice and keyboards in combination with free-space input devices; voice input, touch screen; hybrid interfaces. Clutching mechanisms. Importance of ergonomic details in spatial interfaces.

– [55] THRED (Two-Handed Refining EDitor). Output images displayed on a conventional, non-stereo monitor. The user manipulates two 3D position and orientation trackers with three buttons (i.e. button-enhanced Bats, that is, standard Polhemus sensors with three buttons attached), for each hand. Four postures for holding the Bat. Dominant hand for picking and manipulation, less-dominant hand for context setting. System intended for free-form sketching of polygonal surfaces (terrains, natural objects). Surfaces are hierarchically-refined polygonal surfaces based on quadrilaterals.

– A real elastic object, made from electrically conductive polyurethane, is being used as an input device for 3D shape deformation [56], see Figure A.9. The operation of twisting while bending is possible. Any combination of pressing, bending, twisting possible. A tactile input device, thus giving haptic feedback.

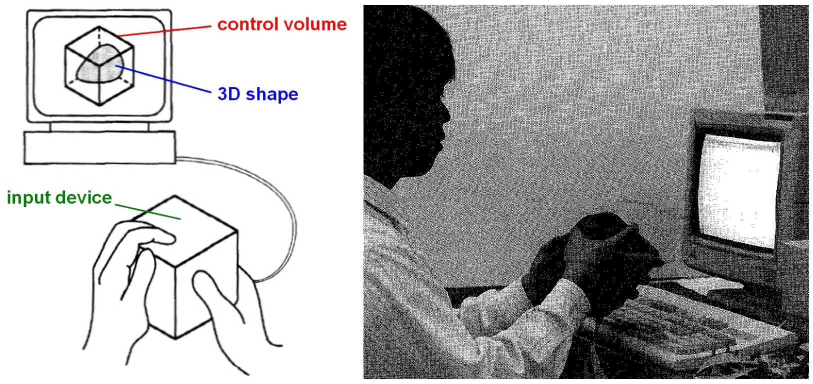

Figure A.9: Murakami's elastic cube for 3D deformation: schematic (left) and usage (right)

– JDCAD [57] — input is a 6-d.o.f. bat, and output is a kinetic head-tracked non-stereo display, see Figure A.10. Object selection using the spotlight metaphor; innovative menus (daisy, ring); object creation, manipulation, viewing.

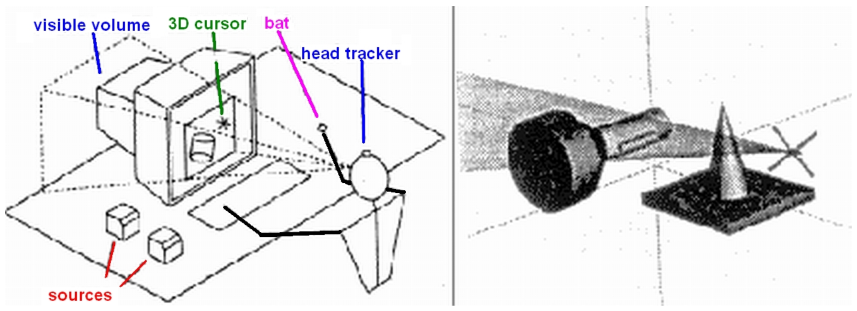

Figure A.10: JDCAD: schematic (left) and cone selection technique (right)

1995

– Mine discusses [18] virtual environment interaction techniques, and gives an introduction to fundamental forms of interaction (Figure A.11): movement (specifiying direction and speed), selection (local and at-a-distance), manipulation (change in position, orientation and center of rotation) and scaling (center of scaling and scaling center; uniform and non-uniform scaling). Lists hand tracking, gesture recognition, pointing, gaze direction. Lists physical and virtual controls. Gives coordinate system transforms in an appendix.

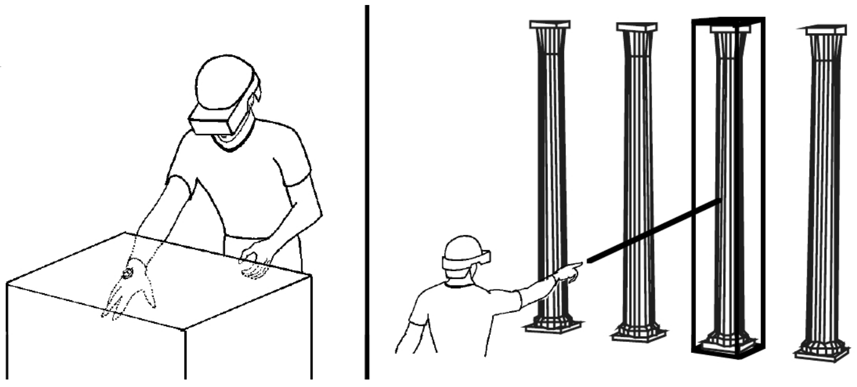

Figure A.11: Mine's local selection (left) and at-a-distance selection (right)

– HoloSketch [58], a VR system for 3D geometry creation and manipulation (Figure A.12). Fishtank stereo CRT, head-tracked stereo glasses, 3D mouse/wand ("one-fingered data glove") augmented by an offset digitizer rod, effectively making from it a six-axis wand. The wand tip feels like an extension of index finger. This 3D mouse has three top buttons and one side button. Multi-level 3D fade-up menu system, invoked by holding down right wand button. Modal editor (a single current drawing or editing mode in force at any given moment). Draws rectangular solids, spheres, ellipsoids, cylinders, cones, rings, free-form tubes, 3D text, isolated line segments, free-form and polyline wires. Editing operations, Significant gains in productivity over 2D interface technology reported.

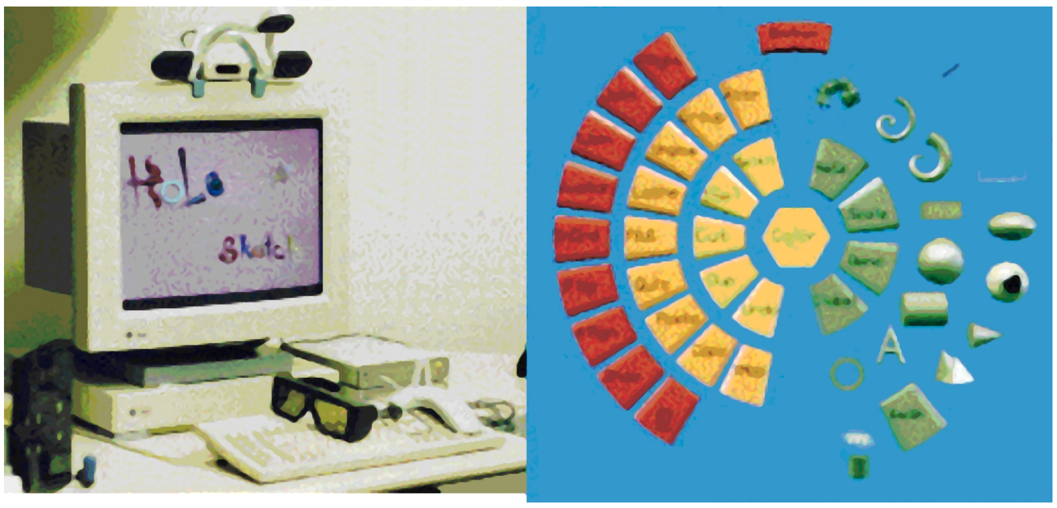

Figure A.12: Deering's Holosketch: head-tracked stereo glasses and 3D mouse/wand (left) and 3D fade-up menu (right)

- [59] Visual interfaces (manipulators) for solids modeling. Similar to [51]. Free-form operators such as blends, sweeps, and deformations. Sweep tool. Warp tool. Rail-tie tool. Rail-curve manipulation tool. Operator space. Visual tool should provide visual clues on its function and use. The design of the visual tool should be based on the user's intuition on how the operator should behave, not on the parameters to the operator.
- [60] PolyShop system. Two ChordGloves (datagloves which have electric contacts on fingertips and on the palm) used for bimanual input. Two hands used for translating, rotating and scaling of virtual objects. Hand gestures, read from different combinations of contacts between fingers and the palm.
- [16] Stoakley, Conway and Pausch present a two-handed architectural design system named WIM (Worlds In Miniature). The user is fully immersed into the virtual environment, and has a concurrent view to a miniature hand-held copy of the entire world attached to a tracker manipulated by the left hand. A clipboard is attached to the left tracker, and the surface of the clipboard represents the floor of the WIM. The right hand holds a ball containing another tracker with two buttons (the first button for selecting objects, and the other for moving them). The

WIM provides both an aerial perspective of the entire scene, and allows the user to manipulate objects in the miniature version of the scene.

1996

– CHIMP (Chapel Hill Immersive Modeling Program) [61], see Figure A.13. The system uses two separate bats, one for each hand. User can perform a unimanual operation for translations and rotations, and a bimanual symmetric movement for scaling. Uses action-at-a-distance for remote selection and interaction with objects, look-at menus, constrained object manipulation, flying, worlds-in-miniature, interactive numbers.

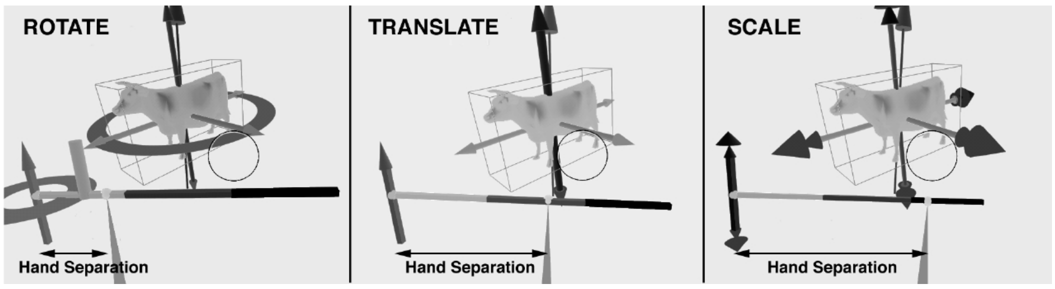

Figure A.13: CHIMP by Mine: Two-handed mode selection

– Go-go technique [15] for non-linear mapping for direct manipulation in VEs. The technique interactively "grows" the user's arm in order to reach remote objects which are to be manipulated.

– [62] Zeleznik's SKETCH system. Tries to bridge the gap between hand sketches and computer-based modeling programs. Uses stroke gestures to generate, move and rotate 3D solids.

1997

– [17] Evaluates techniques for grabbing and manipulating remote objects in virtual environments. Techniques: arm-extension, ray-casting, world-in-miniature (WIM), scaling the user, scaling the entire environment, go-go technique, hybrid techniques. Conclusions: grabbing and manipulation should be considered to be separate issues.

– Mine et al address [63] the lack of haptic feedback in VEs, see Figure A.14. To compensate, authors propose exploiting the only real object user has while in a VE — his own body. Thus the sense of proprioception. Three forms of body-relative interaction: direct manipulation, physical mnemonics, gestural actions. Select, grab, manipulate, release. Pull-down menus, hand-held widgets, head-butt zoom, look-at menus, two-handed flying, throw-over-the-shoulder deletion, virtual object docking. Hand-held tablet as a real surface to work on.

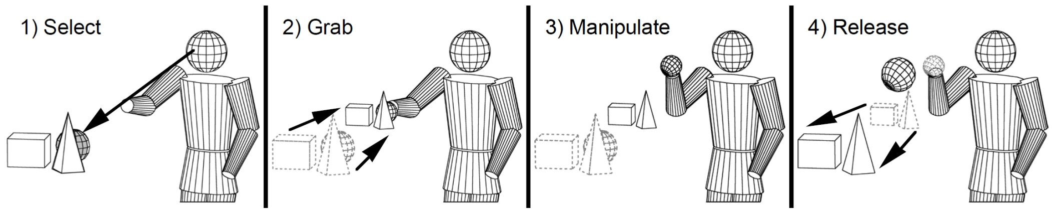

Figure A.14: Mine suggests proprioception as a way to address lack of haptic feedback

– [64] Two-handed direct manipulation on a "Responsive Workbench" [2] (Figures A.15 and A.16), a tabletop stereo display. Fakespace's PINCH gloves, stylus providing single dinstinguished point of action. Polhemus 6-d.o.f. tracker attached to stereo shutter glasses for head tracking. Four basic navigational tasks identified: 1) user slides, lifts up and pushes down the model; 2) user rotates the model around one of principal axes defined by tabletop or model; 3) user zooms in or out; 4) user changes his position relative to table thus relative to model, by walking around the table or by moving the head closer to/away from model. Devices, manipulators, tools. Tools can be *unimanual* (one-handed grab, panning, cutting plane, opacity, temperature, particle, streamline), *bimanual symmetric* (symmetric scale, slide-and-turn, turntable, grab-and-twirl, grab-and-carry), and *bimanual asymmetric* (grab-and-scale, trackball, zoom, free rotation, axis rotation, heuristic rotation, pinch rotation, constrained translation).

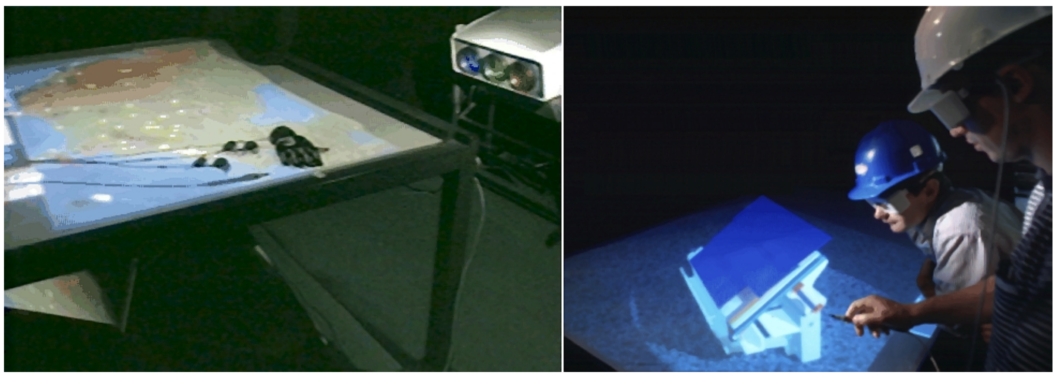

Figure A.15: Responsive Workbench: stereo video projected on mirrors below the desk (left), and persons observing a 3D house model displayed in stereo (right)

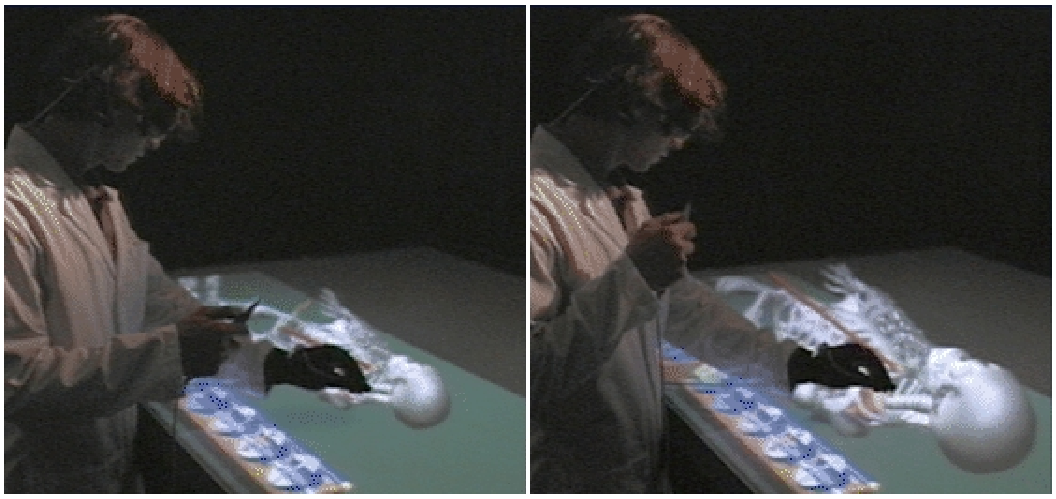

Figure A.16: Responsive Workbench: two-handed operation of zooming in

1998

- [65] Reviews the usability of various 6-d.o.f. input devices. Performance measures: speed, accuracy, ease of learning, fatigue, coordination, device persistence and acquisition. Mices modified for 6 d.o.f., the Bat, the Cricket, the MITS glove, Fingerball, Spaceball, SpaceMaster, Space Mouse, Elastic General-purpose Grip (EGG), Multi-d.o.f. armatures. Conclusion: none of the the existing devices fulfills all aspects of usability requirement for 3D manipulation; however, many insights into the characteristics and pros and cons of various designs; selection of various types of devices for different tasks (speed and short learning — free moving devices; fatigue, control trajectory quality and coordination — isometric or elastic rate control devices;)
- [66] "BUILD-IT" system. AR system, "Natural User Interface". Tangible, graspable control objects. To select an object, user puts a small "brick" (that is, interaction handler) at the object's position on the table. The object can then be rotated, translated and fixed by manipulating its associated brick. Multi-brick and multi-user interaction.
- "ErgoDesk" [67], see A.17. Interaction at ActiveDesk, a rear-projected drafting table-size display, similar to Responsive Workbench [2]. User creates 3D geometry using 2D lightpen-based input. Two-handed operation (user performs camera operations using a 3D tracker in his non-dominant hand). Speech input. Users had difficulty in creating 3D models. Weak modeling functionality. Many deficiencies in the hardware (display blurriness, tethered lightpen, noisy input from lightpen, difficult drawing in 3D).

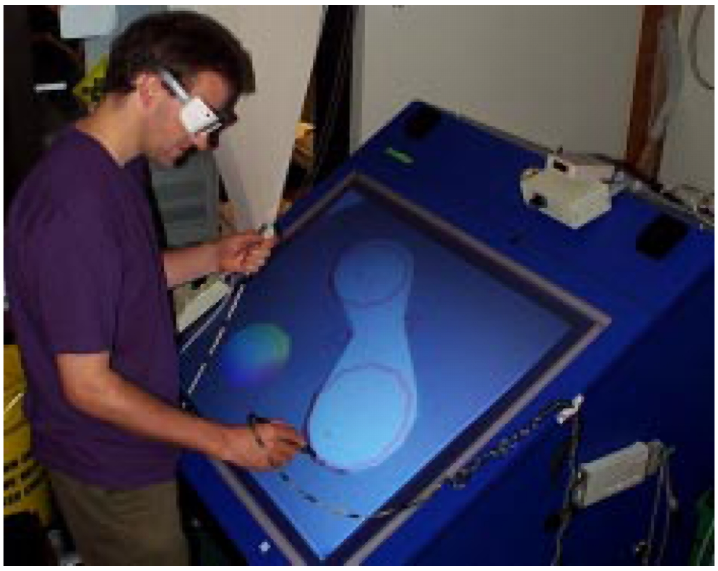

Figure A.17: ErgoDesk by Forsberg et al

– In [68], one- and two-handed gestures (deform, grasp, point, scale, rotation) are used to model and manipulate 3D objects, using data gloves. See Figure A.18.

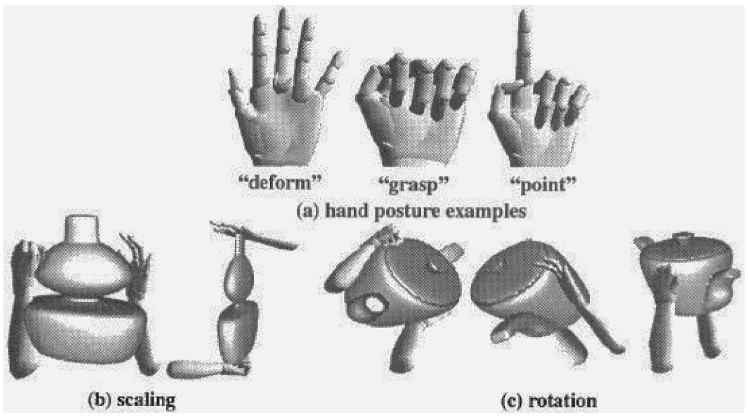

Figure A.18: Some manipulation gestures by Nishino et al

1999

– [69], [70] "Surface Drawing"[1] (Figures A.19 and A.20), a system for drawing organic 3D shapes, intended for artists. Wired glasses. Wired dataglove. Hand gestures. Users construct 3D shapes through "repeated marking". Hand marks 3D space in a semi-immersive environment (Responsive Workbench). Shapes created thus "float" in space above Re-

[1]schkolne.com/sdraw

sponsive Workbench. Tangible tools for edition and manipulation. Tongs[2] to move and scale 3D models. Magnet tool for free spatial hand motion. Magnet tool for drawing and editing (for example, bending) of 3D objects.

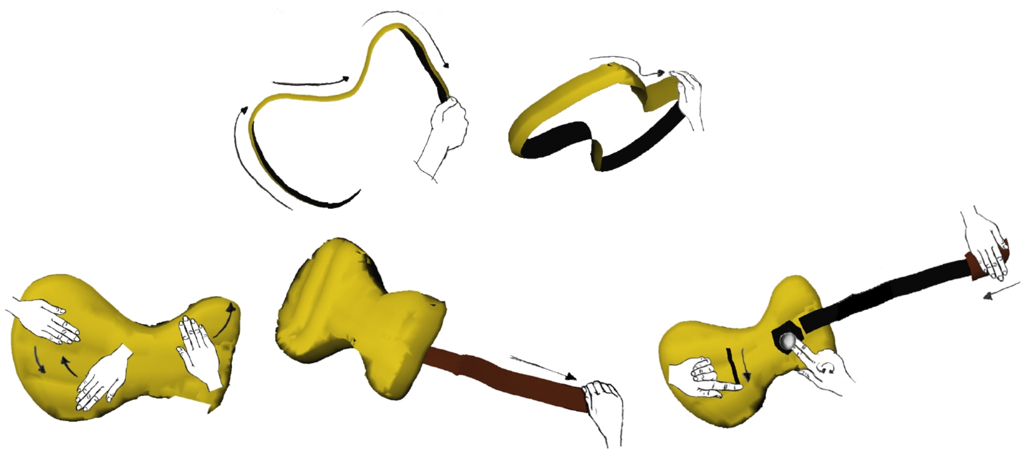

Figure A.19: "Surface drawing" by Schkolne et al: modeling a guitar in five steps

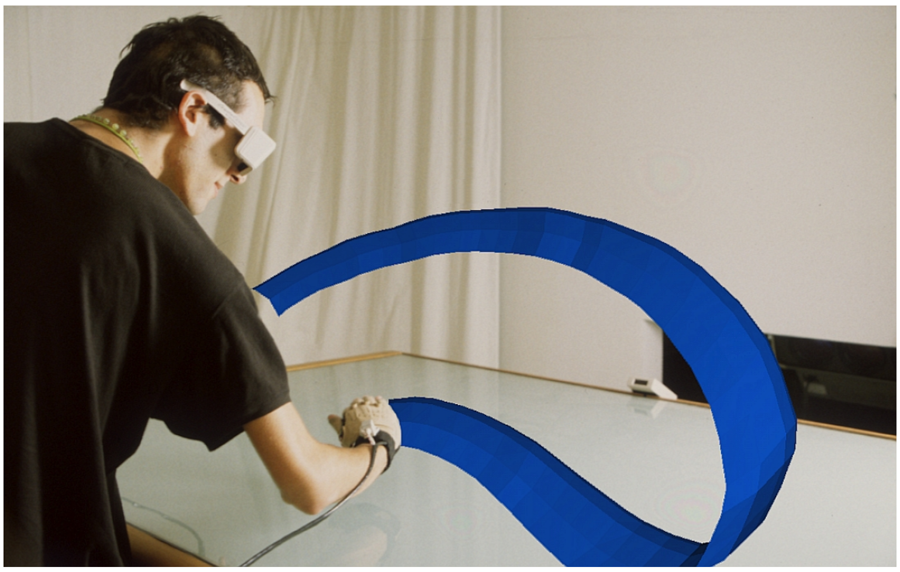

Figure A.20: "Surface drawing" by Schkolne et al: hand motions create 3D shapes which "float" over the Responsive Workbench

[2]A tong is a device for taking hold of objects; usually has two hinged legs with handles above and pointed hooks below.

A.4 2000-2008

2000

– [71] Perceptive Workbench by Leibe et al. Objects are recognized and tracked when placed on the display surface. Uses vision-based methods for interaction. Can identify 3D hand position, pointing direction, and arm gestures, which enhance selection, manipulation, and navigation tasks. Similar to Responsive Workbench however it uses infrared light instead.

2003

– [72] "RoomPlanner". Works on tabletop displays (MERL DiamondTouch used). Eight hand gestures defined. Tapping, dragging, flicking, catching, freeform rotation and scaling, tool palette manipulation and selection, parameter adjustment widget, flat hand, vertical hand, horizontal hand, tilted horizontal hand, two vertical hands, two corner-shaped hands.

2004

– FingARtips [73] by Buchmann et al. The technique tracks hand gestures by using image processing software and finger- and hand-based fiducial markers. The approach allows users to interact with virtual content using natural hand gestures.

– In [74], a sketching system prototype, utilizing gesture recognition and data gloves, was developed. Gestures grab, scale, and drop have been implemented (Figures A.21 and A.22).

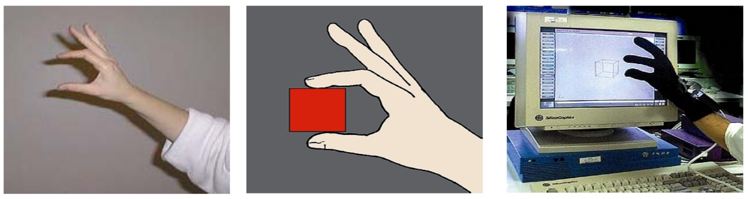

Figure A.21: Operation GRAB in Pratini's system

– In [75], vision-based gesture recognition utilizing white fingertip markers and so-called "black light" is used in order to manipulate 3D virtual objects in front of a large back-projection screen with two projectors for passive stereo (the user wears polarized glasses).

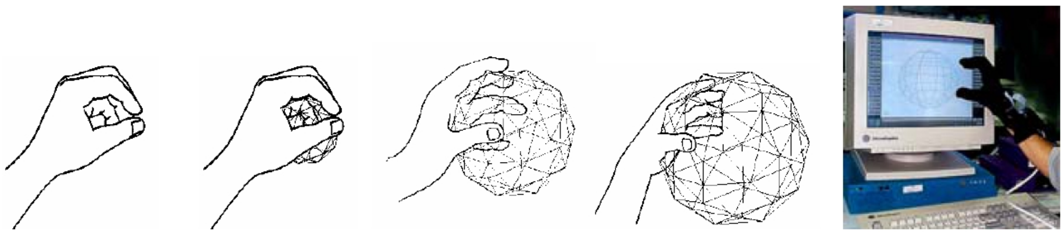

Figure A.22: Operation SCALE implemented as opening/closing the fist in Pratini's system

– In [76], a system for 3D translation and deformation using black gloves with five colors for each finger, stereo vision and passive stereo rendering is reported.

2005

– [77], [78] Gives a classification of 3D widgets, including 3D menus[3]. Especially suited for desktop 3D systems; classification is made according to interaction purpose. Four main classes of widgets:

1. widgets for direct 3D object interaction:
   – *object selection* (direct selection, occlusion selection, distance selection) and
   – *geometric manipulation* (linear transformation, non-linear transformation, high-level object manipulation).
2. widgets for 3D scene manipulation,
3. widgets for exploration and visualization, and
4. widgets for system/application control.

– In [79] unmarked hand gestures are being used for human-computer interaction. The prototype applications learns the background's characteristics in order to segment the hand, and detects and tracks fingertips for state switching.

[3]Online 3D-widget classification site: www.3d-components.org

2007

– [80] An approach for direct manipulation of 3D scenes (Figure A.23), based on visual, non-contact hand tracking and gesture recognition was presented. The system supports translation, rotation and scaling operations. The tracking cameras are located below the interaction volume. Six d.o.f. input is provided using both hands; the system does not require the user to wear any marker or any other kind of device.

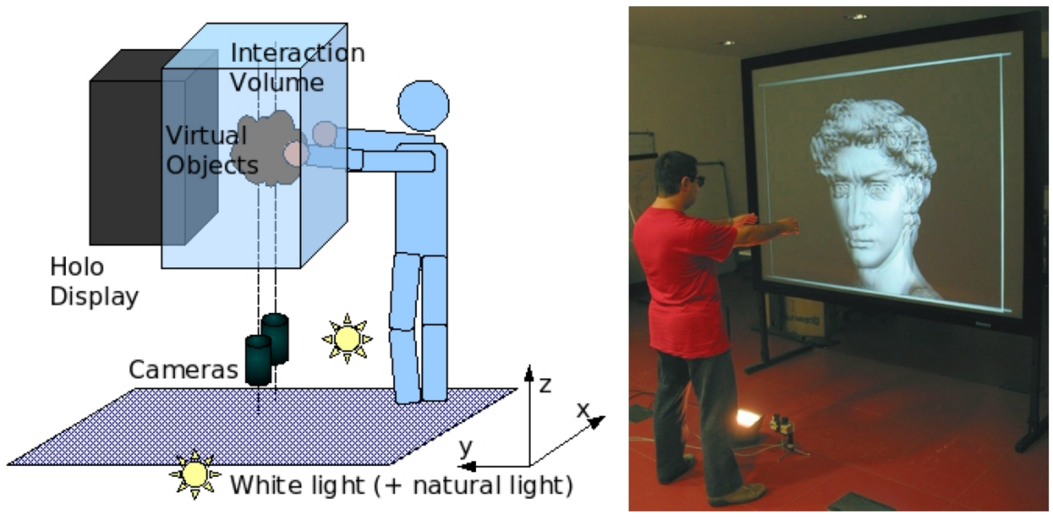

Figure A.23: The setup by Bettio et al. The user stands in front of a large stereo display, and manipulates the model using optically tracked hands.

B
Viola-Jones detection method

The Viola-Jones detection method [29] is a multi-stage detection method that has quickly found wide adoption in the computer vision community, due to its high speed of detection, and high detection rates. Compared to the best previously known detection methods [81] [82] [83] [84] [85], the Viola-Jones method is significantly faster (around fifteen times [29]) while achieving a comparable accuracy. There are four crucial features which distinguish this method:

– **Haar-like features** — Viola-Jones method classifies images based on the values of the so-called Haar-like features, which are simple features based on rectangles. (They are called "Haar-like" due to their similarity with the coefficients in the Haar wavelet transform.)

– **Integral image** — this is a novel data structure used in the pre-processing step of this algorithm, which allows the subsequent phases to run very quickly.

– **AdaBoost-based learning** — the learning part of the Viola-Jones method is based on AdaBoost [30], which combines a relatively small number of weak classifiers into a strong classifier.

– **Cascading strong classifiers** — this part of the Viola-Jones method combines strong classifiers into a "cascade" which discard regions of no interest quickly, thus leaving more processing times for regions that likely contain objects of interest.

B.1
Haar-like features

Haar-like features are prominent local aspects of an image which can be calculated very efficiently.

Let's take a look at Figure B.1, which depicts the extended Viola-Jones method [86]. Suppose we are dealing with a gray-level image I of $W \times H$ pixels. As we will see in Section B.2, there is a very fast way to compute the sum of all the pixels contained in either the upright rectangle, or rectangle inclined at $45°$. A rectangle r, either the upright or inclined one, can be defined as:

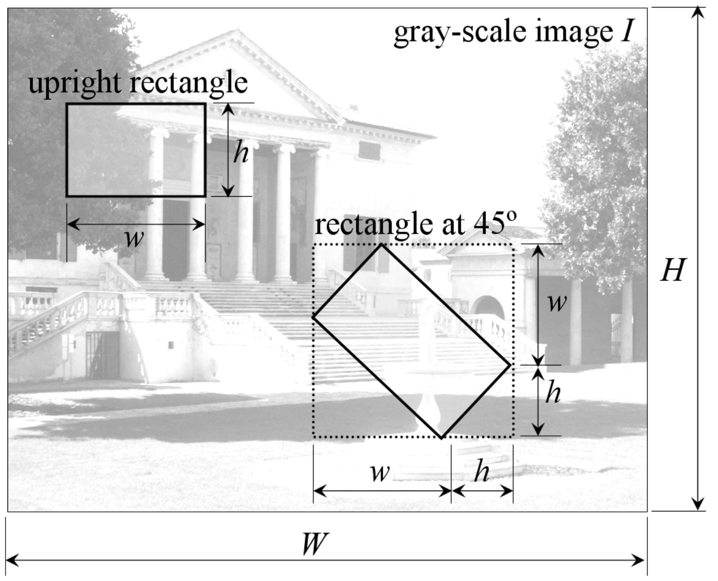

Figure B.1: Two types of rectangles used in the extended Viola-Jones method: 1) upright rectangle, and 2) rectangle inclined at 45°. We compute the sum of all gray-level intensities in rectangle r using function $\texttt{sum}(r)$.

$$r = (x, y, w, h, \alpha) \tag{B-1}$$

where

$$0 \leq x < x + w \leq W, \qquad 0 \leq y < y + h \leq H$$

$$x, y \geq 0, \qquad w, h > 0$$

$$\alpha = 0^\circ \text{ or } 45^\circ$$

The set Φ of all possible Haar-like features ϕ can then be defined as:

$$\Phi = \left\{ \phi \mid \phi = \sum_{i \in \{1, \ldots, N\}} \omega_i \cdot \texttt{sum}(r_i) \right\} \tag{B-2}$$

where N is an arbitrary number of rectangles chosen, r_i are parametrizations of those rectangles (see Equation (B-1)), $\omega_i \in \mathbb{R}$ are weights, and $\texttt{sum}(r_i)$ is the function that sums all the intensity values of all the pixels contained in rectangle r_i.

The problem with set (B-2) is that is infinitely large, therefore we reduce it to the following set:

$$\Phi = \left\{ \phi \mid \phi = \omega_1 \cdot \texttt{sum}(r_1) + \omega_2 \cdot \texttt{sum}(r_2), \quad \omega_1 = -1, \quad \omega_2 = \frac{\texttt{area}(r_1)}{\texttt{area}(r_2)} \right\} \tag{B-3}$$

Thus in this newly defined set (B-3) of features we restrict N to 2, and constrain weights ω_1, ω_2 so that they have opposite signs and are used to compensate for the difference in area size between rectangles r_1, r_2.

We can now define the following set of 14 “template” or “prototype” features (Figure B.2), which will allow us to obtain real features (those that belong to set Φ in Equation (B-3)) by scaling and translating:

- Four edge features — two upright, two inclined
- Eight line features — four upright, four inclined
- Two center-surround features — one upright, one inclined

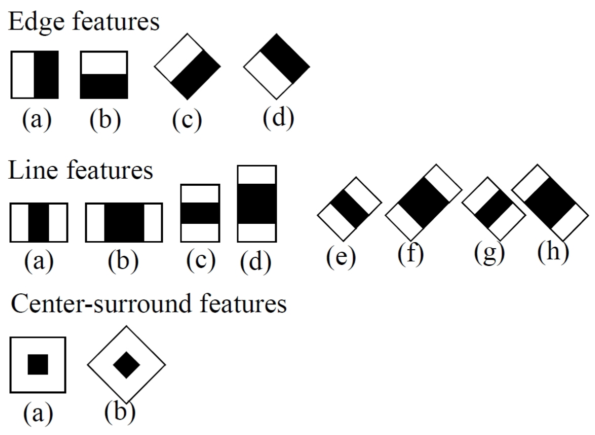

Figure B.2: Fourteen feature prototypes (templates) used in the extended Viola-Jones method

Let now $k = \lfloor W/w \rfloor$ and $l = \lfloor H/h \rfloor$. For seven upright features shown in Figure B.2, by scaling and translating we can generate a total of

$$kl\left(W + 1 - w\frac{k+1}{2}\right)\left(H + 1 - h\frac{l+1}{2}\right)$$

features, while for the remaining seven features inclined at 45° the total is

$$kl\left(W + 1 - z\frac{k+1}{2}\right)\left(H + 1 - z\frac{l+1}{2}\right), \qquad z = w + h$$

Note that line features can be calculated using two rectangles only, first rectangle r_1 encompassing the black *and* white rectangle, and second rectangle r_2 encompassing the black rectangle. For example (Figure B.3), line feature (a) with top left corner located at $(5, 3)$ and dimensions 6×2 pixels can be written as:

$$\phi = -\texttt{sum}(5, 3, 6, 2, 0°) + \frac{12}{4}\texttt{sum}(7, 3, 2, 2, 0°)$$

which represents the combination of one big, encompassing 6×2 white rectangle r_1, and one smaller 2×2 black rectangle r_2 located in the middle of r_1.

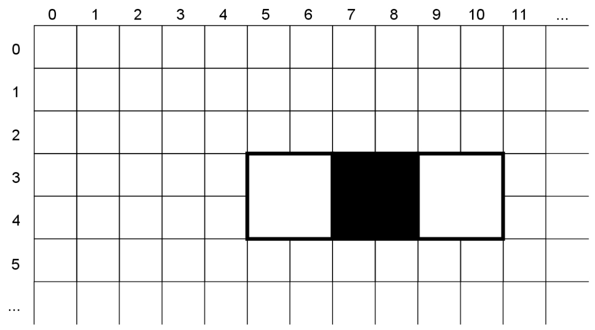

Figure B.3: Example: computing a 6×2-pixel "line feature" (see Figure B.2, feature (a) in the second row) whose top left corner is located at pixel $(5, 3)$

B.2 Integral images

Integral images are useful because, once computed, they enable the Viola-Jones method to subsequently compute features in constant time, i.e. in $O(1)$.

Let I be an $W \times H$ gray-level image. We define *integral image* $I_\int$ to be an image of same dimensions, whose value at pixel (x, y) is defined by:

$$I_\int = \sum_{u \leq x,\ v \leq y} I(u, v) \tag{B-4}$$

Intuitively, pixel $I_\int(x, y)$ contains the sum of all gray-level intensities for pixels that are to the left and up (relative to pixel (x, y)) in the original image I.

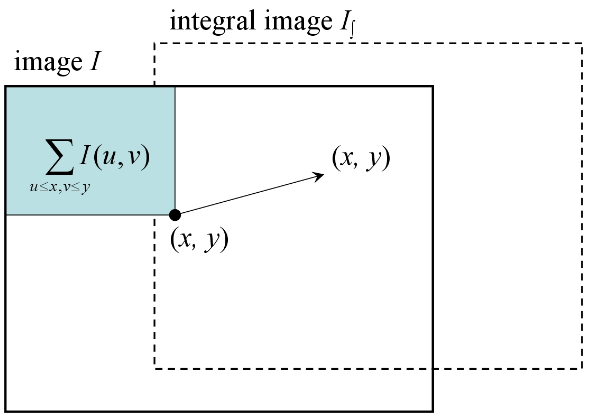

Figure B.4: The value of pixel (x, y) of the integral image $I_\int$ is equal to the sum of all pixels left and up from (x, y) in image I

We can use the following two recurrent relations to compute integral image $I_\int$ in just one pass over the original image I:

$$s(x, y) = x(x, y - 1) + I(x, y), \qquad s(x, -1) = 0$$

$$I_\int(x,y) = I_\int(x-1,y) + s(x,y), \qquad I_\int(-1,y) = 0$$

Here $s(x,y)$ is the function that sums up row values in a cumulative fashion.

Integral images have the following beneficial properties:

- To compute the sum of any rectangle (sub-area) within the image I, just four array look-ups are needed.
- Therefore, the difference between two rectangles can be computed in just eight array lookups.
- Since two-rectangle features shown in Figure B.2 involve two adjacent rectangles, obviously just six array lookups are needed.
- Similarly, any three-rectangle features demands just eight array lookups.

B.3 AdaBoost-based learning

AdaBoost can be defined as "a general method for improving the accuracy of any given learning algorithm" [30]. As a special case, "any" learning algorithm could mean a learning algorithm that guesses the right answer just a little bit above 50%, i.e. just a little bit better than pure chance.

The AdaBoost algorithm:

- **GIVEN**: training set $S = \{(x_1,y_1),\ldots,(x_m,y_m)\}$ where $x_i \in X$ ("instance space") and $y_i \in \{-1,+1\}$ ("set of labels"). In our context, instances $\{x_1,x_1,\ldots,x_m\}$ are $k \times k$-pixel images (for example, $k = 25$) containing ($y_i = +1$) or not containing ($y_i = -1$) human hand.
- **GOAL**: to output a final hypothesis $H(x)$ about the correct label for all $x \in X$
- **ALGORITHM**:
  - Initialize $D_1(i) = 1/m, \quad i \in \{1,\ldots,m\}$
  - For $t = 1,\ldots,T$:
    1. Train weak learner using current distribution D_t
    2. Get weak hypothesis $h_t\colon X \to \{-1,+1\}$ from the weak learner, so that error
       $$\epsilon_t = \sum_{i:\ h_t(x_i)\neq y_i} D_t(i)$$
       is low with respect to D_t
    3. Choose factor
       $$\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon}$$
       Factor α_t measures importance given to h_t.

4. Update D_{t+1}:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$$

or equivalently

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

where Z_t is the normalization factor (chosen so that D_{t+1} is a distribution). This step increases (decreases) the weight of correctly (incorrectly) classified training example $i \equiv (x_i, y_i)$.

– Output the final hypothesis $H(x)$:

$$H(x)\colon X \to \{-1, +1\}$$

$$H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right), \qquad x \in X \tag{B-5}$$

This final hypothesis ("strong classifier") $H(x)$ can be considered a weighted majority vote of T weak hypotheses, and factor α_t can be considered the weight attributed to the weak hypothesis h_t.

B.4 Cascading strong classifiers

In practice, a strong classifier (see Eq. B-5) can achieve any desired accuracy, however the speed is dissatisfying. Because of this strong classifiers are chained into the so-called "attentional cascade", or just "cascade", in order to achieve high frame rates. In such a chain, all strong classifiers are trained to detect approximately all objects *and* to reject a certain percentage of subwindows that do not contain the object.

For example:

– the first strong classifier in the cascade could be made of just two features, reject 50% of non-hand subwindows and detect hands correctly in 99.999% of all subwindows.

– the second strong classifier could consist of five features, reject 80% and detect correctly in 99.0% cases.

– the next six strong classifiers could consist of 25 features, reject 90% and detect correctly in 98.0% cases.

– . . . and so on.

Taking now these classifiers, and chaining them into a series, we would obtain a cascade consisting of eight strong classifiers. The point in building a cascade is that a cascade significantly reduces processing times: the first strong classifiers reject many of subwindows that do not contain the object, while at the same time detecting correctly almost 100% of all the subwindows containing the object. All the subwindows that “passed through” the first strong classifier now have to be processed by the second classifier, which rejects even more subwindows, and so on. A subwindow must pass through all the classifiers in order to be classified as “positive”.

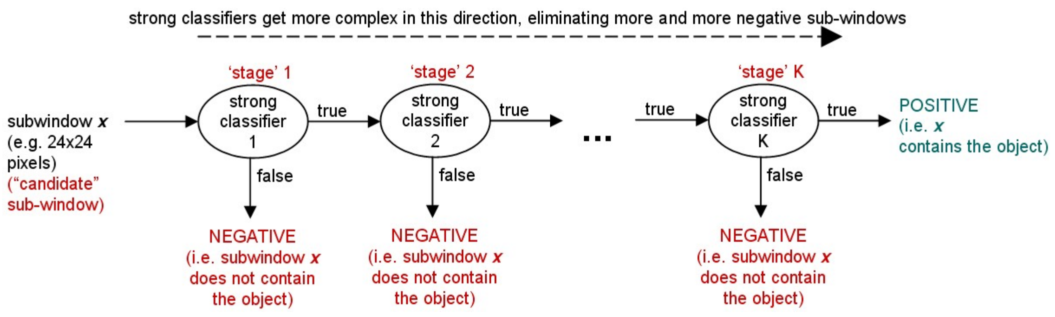
strong classifiers get more complex in this direction, eliminating more and more negative sub-windows
subwindow x (e.g. 24x24 pixels)
(“candidate” sub-window)
‘stage’ 1
strong classifier 1
‘stage’ 2
strong classifier 2
‘stage’ K
strong classifier K
true
false
NEGATIVE (i.e. subwindow x does not contain the object)
POSITIVE (i.e. x contains the object)

Figure B.5: Cascade of strong classifiers using Haar-like features

C
KLT features

As was already mentioned in Section 5.7 on page 47, *features* are properties of textured surfaces that allow us to “latch” onto them, see Figure 5.9 on page 47. By “latching” onto these features, we can thus effectively “latch” onto the object being tracked, therefore tracking the object.

The mathematical details behind KLT features [23] [24] [25] will now be given. Let $I(x, y, t)$ be “gray-level image sequence” functions defined on a sequence of $M \times N$ arrays at time moments $0, 1, 2, \ldots, L$, i.e.:

$$I : \{0, 1, 2, \ldots, N-1\} \times \{0, 1, 2, \ldots, M-1\} \times \{0, 1, 2, \ldots, L\} \longrightarrow [0, 1]$$

where N is the width of the image, M height, and L the time instant for the last image in the sequence. Let

$$I(x, y, t) = c, \;\; c \in [0, 1]$$

where c is a gray level between 0 (black) and 1 (white).

Let now W be a window in an image I, with dimensions $M' \times N'$, with the upper left corner located at (x', y'). Then we can restrict function I to the window W, thus obtaining function I_W:

$$I_W : W \longrightarrow [0, 1]$$

We are interested in tracking objects visible in the input image stream. Put differently, there exist certain patterns in the input image sequence which can be expressed formally like this:

$$I(x, y, t + \tau) = I(x - \xi, y - \eta, t) \tag{C-1}$$

Intuitively, Equation C-1 says that, having the current image $I(x - \xi, y - \eta, t)$, we can compute the next image (at time $t + \tau$) by moving all the pixels from the image $I(x - \xi, y - \eta, t)$ by a displacement vector $\vec{d} = (\xi, \eta)$.

Let now define $J(\vec{x}) = I(x, y, t + \tau)$ and $I(\vec{x} - \vec{d}) = I(x - \xi, y - \eta, t)$. Note that we omitted time parameter t for brevity (by definition, image J follows

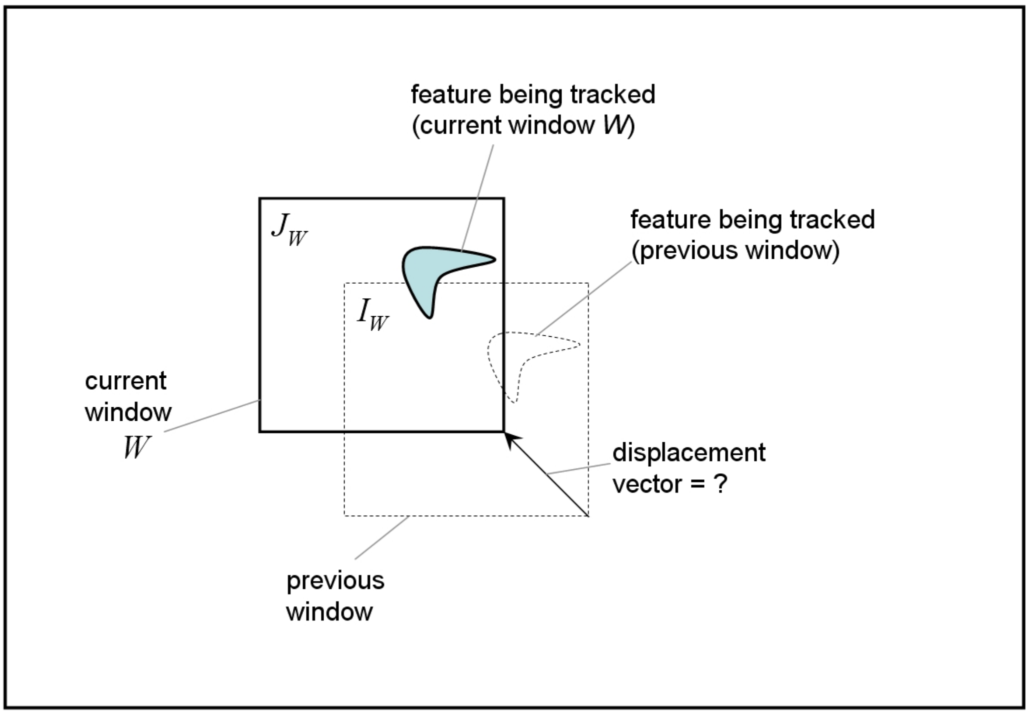

Figure C.1: Illustration of tracking based on KLT features. Window W is the current window, for example a rectangle of 10×10 pixels. J_W is the restriction of I on the current window W. I_W is the restriction of I on the previous window. What is being searched for, is the displacement vector $\vec{d}$, which enables us to position window W correctly in the current image.

I). The we can rewrite Equation C-1 as

$$J(\vec{x}) = I(\vec{x} - \vec{d}) + n(\vec{x}) \tag{C-2}$$

where $n(\vec{x})$ represents noise present in $J(\vec{x})$. The desired displacement vector $\vec{d}$ is then computed minimizing the following area integral over W:

$$\epsilon = \int_W \Big(I(\vec{x} - \vec{d}) - J(\vec{x})\Big)^2 w(\vec{x})\, d\vec{x} \tag{C-3}$$

Function $w(\vec{x})$ is the weighting function, which can be set to a desired function, for example to a constant function ($w(\vec{x}) = 1$) or to the Gaussian — depends on the application.

The question now is how to solve Equation C-3 for $\vec{d}$ so that:

$$\epsilon \longrightarrow \min$$

Note that when $\vec{d}$ is small, we can develop I into its Taylor series:

$$I(\vec{x} - \vec{d}) = I(\vec{x}) - \vec{g} \cdot \vec{d} + \ldots$$

...where $\vec{g}$ is a constant vector. We keep just the first two terms, so $I(\vec{x}-\vec{d}) = I(\vec{x}) - \vec{g}\cdot\vec{d}$, therefore equation C-3 becomes

$$\begin{aligned} \epsilon &= \int_W \Big(I(\vec{x}-\vec{d}) - J(\vec{x})\Big)^2 w(\vec{x})\, d\vec{x} = \\ &= \int_W \Big(I(\vec{x}) - \vec{g}\cdot\vec{d} - J(\vec{x})\Big)^2 w(\vec{x})\, d\vec{x} = \\ &= \int_W \Big(h(\vec{x}) - \vec{g}\cdot\vec{d}\Big)^2 w(\vec{x})\, d\vec{x} \qquad \text{(C-4)} \end{aligned}$$

where $h(\vec{x}) = I(\vec{x}) - J(\vec{x})$. Equation C-4 can now be solved in the closed form, because ϵ is now a quadratic function. To find the minimum for ϵ, we now differentiate Equation C-4 relative to $\vec{d}$ and set the resulting expression to zero:

$$\int_W \Big(h(\vec{x}) - \vec{g}\cdot\vec{d}\Big)\,\vec{g}\; w(\vec{x})\, dA = 0$$

We can now replace $(\vec{g}\cdot\vec{d})\vec{g}$ by $(\vec{g}\cdot\vec{g}^{\tau})\,\vec{d}$. Since $\vec{d}$ can be considered constant for all pixels in W, we now obtain

$$\int_W h(\vec{x})\;\vec{g}\; w(\vec{x})\, dA = \left(\int_W (\vec{g}\cdot\vec{g}^{\tau})\;\; w(\vec{x})\, dA\right)\cdot\vec{d}$$

or simply switching the sides

$$\left(\int_W (\vec{g}\cdot\vec{g}^{\tau})\;\; w(\vec{x})\, dA\right)\cdot\vec{d} = \int_W h(\vec{x})\;\vec{g}\; w(\vec{x})\, dA$$

The previous equation can now be rewritten as

$$G\vec{d} = \vec{e} \qquad \text{(C-5)}$$

where

$$G = \int_W (\vec{g}\cdot\vec{g}^{\tau})\;\; w(\vec{x})\, dA$$

and

$$\vec{e} = \int_W h(\vec{x})\;\vec{g}\; w(\vec{x})\, dA$$

Thus to find $\vec{d}$, we must, for each pair of consecutive frames, first compute G, then $\vec{e}$, and then using the linear system C-5 we can compute $\vec{d}$.

D
Hartley-Sturm triangulation method

The Hartley-Sturm triangulation method [20] is an algorithm that, under the assumption of Gaussian noise present in image point measurements, gives a *provably optimal* global solution to the triangulation problem.

In further text we assume that we know fundamental matrix F exactly, and that any error is due either to 1) the digitalization process on the CMOS/CCD chip of the camera, or 2) to the feature extraction process. It is assumed that these errors follow Gaussian distribution.

Let:

$\vec{u} \leftrightarrow \vec{u}'$ — an noisy, incorrect measured pair of correspondent features for the left and right camera respectively. This pair does not satisfy $\vec{u}'^{\tau} F \vec{u}$.

$\vec{\hat{u}} \leftrightarrow \vec{\hat{u}}'$ — a correct pair of correspondent features for the left and right camera respectively. Point $\vec{\hat{u}}$ should in general lie close to point $\vec{u}$, and $\vec{\hat{u}}'$ to $\vec{u}'$. Points $\vec{\hat{u}}, \vec{\hat{u}}'$ satisfy $\vec{\hat{u}}'^{\tau} F \vec{\hat{u}}$.

The goal therefore is to find points $\vec{\hat{u}}, \vec{\hat{u}}'$ that minimize the function

$$\left(d(\vec{u}, \vec{\hat{u}})\right)^2 + \left(d(\vec{u}', \vec{\hat{u}}')\right)^2 \tag{D-1}$$

where $d(\vec{u}, \vec{v})$ represents Euclidean distance between 2D points $\vec{u}, \vec{v}$. This minimization task is equivalent to finding real number t for which the following cost function attains minimum:

$$s(t) = \frac{t^2}{1 + f^2 t^2} + \frac{(ct + d)^2}{(at + b)^2 + f'^2 (ct + d)^2} \tag{D-2}$$

The algorithm (see [87], page 318):

- **GOAL** — compute 2D points $\vec{\hat{u}}, \vec{\hat{u}}'$ that minimize Eq. D-1. Given are measured 2D correspondent points $\vec{u}, \vec{u}'$, and fundamental matrix F.
- **ALGORITHM**:

  1. define transformation matrices

$$T = \begin{pmatrix} 1 & & -u \\ & 1 & -v \\ & & 1 \end{pmatrix} \text{ and } T' = \begin{pmatrix} 1 & & -u' \\ & 1 & -v' \\ & & 1 \end{pmatrix}$$

2. replace F by $T'^{-\tau}FT^{-1}$
3. compute epipoles $\vec{e} = (e_1, e_2, e_3)^\tau$ and $\vec{e}' = (e'_1, e'_2, e'_3)^\tau$ so that $\vec{e}'^\tau F = 0$ and $F\vec{e} = 0$. Normalize $\vec{e}, \vec{e}'$.
4. form matrices

$$R = \begin{pmatrix} e_1 & e_2 & \\ -e_2 & e_1 & \\ & & 1 \end{pmatrix} \text{ and } R' = \begin{pmatrix} e'_1 & e'_2 & \\ -e'_2 & e'_1 & \\ & & 1 \end{pmatrix}$$

5. replace F by $R'FR^\tau$
6. set $f = e_3, f' = e'_3, a = F_{22}, b = F_{23}, c = F_{32}, d = F_{33}$
7. form 6-degree polynomial

$$g(t) = t\left((at+b)^2 + f'^2(ct+d)\right)^2 - (ad-bc)(1+f^2t^2)^2(at+b)(ct+d)$$

8. solve $g(t)$ in order to obtain 6 roots
9. evaluate cost function $s(t)$ (see Eq. D-2) at the real part of each of the six roots. Also, find $\lim_{t\to\infty} s(t)$. Select t_{min} that gives the smallest value for $s(t)$.
10. evaluate two lines $\vec{l} = (tf, 1, -t)$ and $\vec{l}' = F(0, t, 1)^\tau = (-f'(ct + d), at + b, ct + d)^\tau$ at t_{min}, and find $\vec{\hat{u}}$, $\vec{\hat{u}}'$ as the closest points on these lines to the origin.
11. replace $\vec{\hat{u}}$ by $T^{-1}R^\tau\vec{\hat{u}}$ and $\vec{\hat{u}}'$ by $T'^{-1}R'^\tau\vec{\hat{u}}'$
12. compute the requested 3D point $\vec{X}$ by any other method, for example by mid-point method.