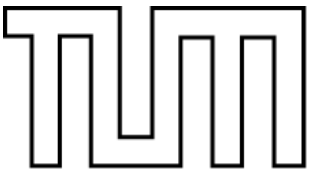
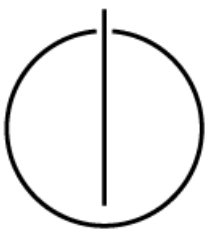


Fakultät für Informatik
der Technischen Universität München

Diplomarbeit in Informatik

Building a gesture based information terminal

Nikolas Dörfler



Fakultät für Informatik
der Technischen Universität München

Diplomarbeit in Informatik

Building a gesture based information terminal

Bau eines gestengesteuerten Informationsterminals

Bearbeiter: Nikolas Dörfler
Aufgabensteller: Prof. Ph.D. Gudrun Klinker
Betreuer: Dipl.-Inf. Florian Echtler

Abgabedatum: 15.9.2008

Ich versichere, dass ich diese Diplomarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I assure the single handed composition of this diploma thesis only supported by declared resources.

(*Nikolas Dörfler*)

Contents

List of Figures

Abstract

In this thesis a Multi-Pointer Virtual Touchscreen system is developed, which allows the application of pointing gestures in a standard GUI interface. Unlike in most related projects, an acoustic tap detection method supports the optical tracking. The system is designed as an information terminal for public places, though other applications are possible. Its uses a projected beamer screen as output. Users can control several pointers, similar to mouse pointers in a standard desktop interface. Selection of objects and drag-and-drop actions can be done by tapping the table surface with the finger tip. The optical tracking system derives one pointer position per hand. A novel Multi-Pointer X-Server (MPX) is utilized and configured to handle these coordinates similar to normal mouse input. The new architecture provides the ability to display several mouse cursors and supports Multi-Pointer aware applications. These applications handle the input independently and allow simultaneous actions. Aside from that, standard applications can be operated in conventional manner. For detection of surface touches, a tangible acoustic interface is applied. Tap locations are distinguished implementing Time Difference of Arrival (TDOA) estimation. The mathematical basis for this approach is the Generalized Cross Correlation with Phase Transform (GCC-PHAT). Employing only a stereo audio input allows differentiation of tap locations, though archived accuracy is still limited.

Chapter 1

Introduction

"Imagine eating Chinese food with only one chopstick..."
Bill Buxton [1]

Computer development has always concentrated on improving values, such as calculation performance or higher memory densities. These attributes allow more complex problems to be solved and more calculation intensive software is possible. Also great effort has been spent on the design and appearance of graphical user interfaces. While the look of these user interfaces was optimized the preferably used interaction methods had not changed for many years.

But recently a new interest in improving human computer interaction can be observed. An interesting aspect is the development of touch sensitive devices. Touchpads and touchscreens have been in use for some time. The recent progress in Multi-Touch technology may lead to new commonly used interaction methods.

In the beginning of computer development, computers were built for experts, who where trained in the use of slot carts, keyboards, and complicated operating system software. Nowadays computer are used by a broad mass of people which not necessarily possess expert knowledge. Some input methods e.g. mouse and keyboard had been established. Although these devices seem very intuitive and exact for many computer users, they can be difficult to handle for new and technically unexperienced persons. An interaction method, which resembles the way humans communicate, may ease computer use for these people.

Different ways for doing this have been researched. Speech recognition, handwriting recognition and gesture input are the most important ones. Technical effort for most of these approaches is high. Speech input has made some advances, but is still not commonly accepted, mostly due to high error rates and computational load for the hardware. Handwriting is only practical for text input and is already usable to some extent.

Unlike this, the development of graphical user interfaces pushed the use of pointing and other gestures into spotlight. A kind of pointing gestures are already implemented in nearly all graphical user interfaces. Pointing devices, such as mouse or trackball, had been established and are commonly accepted by users. But such a device is only a metaphor for a more natural input method of directly pointing with the finger or hand to the objects one wishes to manipulate.

This is an important reason for the development of touch and gesture based input systems. The new generation of these devices is Multi-Touch or Multi-Pointer capable. A recently discussed product which support gestures and Multi-Touch is e.g. the IPhone. These devices can detect

multiple touch points at once. This allows a completely new input technique.

There is a great number of techniques to produce a multi-touchable device. Systems to detect touches directly such as capacitive or resistive sensing or frustrated internal reflection are restricted to the specially prepared surface and require specialized hardware. In this work we present a different approach, using computer vision to create a virtual touchscreen with a projected display. A camera and image processing software tracks the positions of hands and other objects. In this way pointer coordinates are estimated.

We wanted our system to support as much standard software applications as possible. Therefore we made use of an graphical X Window system, which is specially designed to support multiple mouse cursors. These cursors can now be controlled by the computer vision input system. When the user want to move the cursors he simply moves his hand to the desired location on the screen. Our approach will not support true Multi-Touch, but Multi-Pointer. But the great advantage of this new system is, that legacy applications, which are designed for single pointer use, can be run together with Multi-Pointer aware software.

Our system is designed as an information terminal system, to be placed in a public space e.g. an airport or station. Most of the users here would appreciate the natural input technique, because they may be unexperienced to computers. Requirements for such a terminal system are high, since they could be exposed to environmental influences such as dirt, humidity and vandalism. The use of an image processing system can prove advantageous here. Cameras can be placed in a protected case, e.g. on the ceiling, which makes the system vandalism safe.

As a starting point we used the Siemens Virtual Touchscreen (SiViT) [2]. The SiViT was the first commercial virtual touchscreen terminal developed for public use. The original SiViT, in the version we had at hand for this project was dated from 1998. We have been donated with a SiViT model. It was equipped with new hardware and software components and upgraded to Multi-Pointer operation.

The horizontal position of the interaction surface in such a table top system could resemble a table workspace. These systems are called table top interfaces. There have been a lot of research for table top systems and their benefit for collaborative work. Present Groupware Systems are not designed for the simultaneous work in a common virtual workspace. This has some advantages but also constrains people to solitary work behavior. For group discussions it may be interesting to share a common interface. A large shared input space, such as a table top interface could be used by

multiple people. Terminal systems similar to the SiViT may also profit from these developments.

1.1 Making the table surface touchable

The SiViT uses computer vision as an input method. The great disadvantage of this method is, that it is hard to detect, when the user actually wants to select and manipulate an object. Normal mouse input allows us to select objects with the mouse button click. Other touch technology can directly spot the point of touch by finger or stylus. This is more difficult in a computer vision system. The click event has to be simulated. We considered different approaches for use in this project:

- Trigger a click event whenever the user has pointed to an object for some time. This has the advantage of being very easy to implement but harder for the user to handle.

- Make use of the sound the fingertip makes when tapping or knocking on the surface of the table. This seems easy to implement. But false clicks can be triggered by noises or when the user accidentally hits the table border. Sound source localization methods use multiple microphones. This may avoid these problems and allow Multi-Pointer systems. Interfaces that work with acoustic tracking methods are called tangible acoustic interfaces (TAI).

- Use gestures e.g. wink, stroking or circle movement for activation. This has been considered for this project, but not yet implemented. The applicability of this method depends on the design of the graphical user interface and the application.

The first method requires some user training and could lead to reduced performance. While the second approach performs better, it has the above described problems. Anyhow we decided to use these two methods. The third approach would need a specially designed graphical user interface or application. Implementing this method would go beyond the scope of this thesis but may be subject to future research.

1.2 Application scenarios for virtual touchscreen systems

Virtual touchscreen systems similar to the SiViT, have several advantages over normal touchscreens. The hardware complexity is relatively low, which makes them affordable. The use of a beamer instead of a monitor allows a bright and very large display area. This could be advantageous for visually handicapped persons. Generally touchscreens suffer from the problem that the user occludes the screen with the arm or hand. The beamer projection reduces this problem when it is done from the top. It allows more arbitrary interaction surfaces, as a projection can be made to nearly every bright table. Another advantage is that the surface does not need to be directly hit by the users finger. This makes the systems interesting for the application in a sterile environment, e.g. in operating rooms in hospitals. On the other hand, it may be beneficial in spaces that are very dirty, e.g. factories.

Pointing with fingers is a very natural gesture for selecting, drawing and placing objects. Additionally other gestures may be implemented. One example would be winking, to skim through pages. Two handed gestures offer even more possibilities.

Using Virtual Touchscreens in an office environment have been proposed by some researchers. The application scenarios for such an office system, reach from virtual calculators up to augmenting paper documents.

Collaborative Workspace, where several persons work together and share a big screen area have been discussed. Again the big display size is advantageous here. As we will see the underlying graphical environment have to be carefully designed for Multi-Person use.

Finally virtual touchscreens may allow new concepts for the development of computer games and entertainment programs.

1.3 Outline of this thesis

The main scope of this project is the development of a new interaction system. This system is based on computer vision (Virtual Touchscreen) combined with a tangible acoustic interface. The system will allow multiple hand detection and even multiple users.

In this section a short overview of the following chapters will be given. First we will present a short introduction in human computer interaction in graphical user environments. The different actions, humans can use to manipulate objects will be described. Some input devices will be in-

troduced to relate them to the input system of this work. We explain the three-state-model as a good theoretical basis for pointer interaction. Furthermore we discuss application of gesture input methods.

In chapter 2 a short overview on other touchscreen technologies will be given. Touchscreens with Multi-Touch ability as a special input device will be introduced. Additionally different tabletop systems will be discussed, also as a common workspace for group activities.

In chapter 3 Virtual Touchscreen systems will be explained in more detail. The SiViT, as a project basis will be of special interest. Further on, the mathematical basis for acoustic touch detection using time difference of arrival will be explained. The Generalized Cross Correlation with Phase Transform (GCC-PHAT) is an adequate method and will be explained here.

A short requirements analysis is done in chapter 4. Chapter 5 explains the basic system design used in this project. The layered design structure of the system is explained here. The transport and exchange of data between the layers is described. Double tap as a method for activating Drag-and-Drop is presented.

A more detailed view on the single components will be given in chapter 6. The internal functionality of the program units is explained. Special emphasis is placed on the acoustic tap detection and the operation system interface. For test purpose a demonstration application, the game Multi-Touch Puzzle will be developed. The OpenGL Utility Library (GLUT) had to be enhanced to make this application Multi-Pointer aware.

In the last chapter, the main findings of this project will be summarized and proposals for improvement and further research will be given.

Chapter 2

Background Information

> *The systems which they designed were like violins, rather than record players: if you learned them, you could do amazing things, and while this took an investment, the design was such that your time was not wasted learning to work around poorly conceived design decisions.*
>
> *Bill Buxton* [3]

In this chapter an introduction on available human computer interfaces and their possibilities and disadvantages is given. Input devices can be classified by multiple attributes. These attributes are explained and a taxonomy of devices can be given. The three state model for interaction is introduced to describe pointer actions in graphical environments. Additionally we examine some of the tabletop interfaces and touch technologies already developed in previous works. A short discussion on the abilities of singletouch- multitouch and multi pointer systems is presented.

2.1 Human Computer Interaction

Human Computer Interaction (HCI) is an important field of research in the last years. The main goal of HCI is to allow the human user to make input, a computer system can interpret, understand and give back results in some kind of human understandable output. The practicability and ergonomics of a computer system, in a large part, depends on the used input method. Human forms of input are often hard to accomplish in computer systems, e.g. voice input is natural for humans though it is harder to archive for a computer.

To make input to the computer system input devices are used. An input device is defined, as a computer hardware object, used to input data in an information processing system. It is therefore the interface between humans and machines. Input devices can be anything from buttons to speech recognition systems. We will discuss the different available input device classes later in this document.

Essential for the acceptance and usability of a human computer interface is the choice of good metaphors. Generally a metaphor describes something unknown by associating it with a familiar thing. Metaphors are commonly used in poetics and literature, but they also allow easy understanding of complicated scientific matters.

In HCI, a good metaphor also helps the user to learn a new concept by relating it to something known and previously learned. Examples for metaphors in a graphical user interface are the desktop metaphor or the

typewriter metaphor in a word processing application. A mouse pointer is also a metaphor. It helps the user to understand mouse operation. The pointer should be actually replaced by a real pointing gesture. (also see [4] pages 123-124).

2.1.1 Interaction in graphical user interfaces

The development of graphical user interfaces provided a great deal of new possibilities for computer interaction. In a text environment, e.g. MS-DOS, input and output was limited. The user had to learn complex commands, type them and understand the often complicated output. In a graphical environment objects can be better represented by pictures, graphics and text. Input devices used in such an environment must be suitable and possibly need different attributes than in a text system. One example for such a device is the mouse. It allows precise control on a pointer, to select and manipulate objects. Though the mouse has been invented in 1968 it needed a long time until graphical user interfaces had been introduced. Mouse interaction is linked to an interface concept, called WIMP. ([4] p.107-112) WIMP stands for Windows, Icons, Menus and Pointers. These four items represent different concepts. Windows are areas where information can be displayed and manipulated. Icons represent saved information and program object and menus allow the execution of commands. Pointers represents the user input. Therefore the pointer can be seen as a metaphor for the finger or the hand.

Computer interactions in a graphical user interface consider mostly modifications of (virtual) objects, such as texts, pictures or windows.

Possible Interactions handling these objects could be [3]:

- Select an object
- Position an object in one or more dimensions
- Orient and rotate an object in one or more dimensions
- Ink: e.g. draw a line.
- Text input
- Enter discreet scalar values

A specific input device can be better suited for some of these tasks but less for others, e.g. a keyboard is good for text input but less performant in line drawing.

2.1.2 Taxonomy of input devices

Different input devices can be classified by various criteria. To give a short overview the taxonomy of input devices following [3] is described. Figure 2.1 provides an overview on these input systems and their classifications. Interesting distinctions on the devices should be made:

- **continuous or discreet**: Some devices allow continuous input while others allow only discreet input. Discreet input could e.g. be keystrokes or button clicks. Continuous input can be the movement of a pointer. This is a important distinction and needs to be considered when designing a user interface.

- **relative or absolute**: Also interesting is the question whether a device provides relative (motion) or absolute (position) data. E.g. joysticks normally provide relative movement data, therefore they are not applicable for tracing a map. A touchscreen determines absolute coordinates.

- **Degrees of Freedom(DOF)**: The application of a device depends also on the DOF a device provides. Continuous devices like mice and touch pads normally provide 2 DOF (X and Y Coordinate). Special 3D devices may allow 3 or more DOF.

- **direct or indirect**: Some devices e.g. a mouse use an intermediary mechanic system to translate movements into screen coordinates. These coordinates can also be scaled, e.g. to allow the movement over the complete screen by a small touchpad movement. Touch screens on the contrary side, detect the positions directly and return the actual position of the touch.

A touchscreen is a special input device, because it provide a direct way of interaction. As the user interacts directly with the picture, pointer coordinates are similar to input coordinates. This is highly intuitive for the user. Touchscreens, which additionally sense pressure, multiple touch-spots at once (Multi-Touch) and arbitrary shapes (e.g the hand area) can provide further possibilities. We will discuss them in the next chapters.

2.1.3 State model for making input

The state model is a model to describe interactions with a pointing device on a (graphical) user interface. This model has been formulated by Buxton,

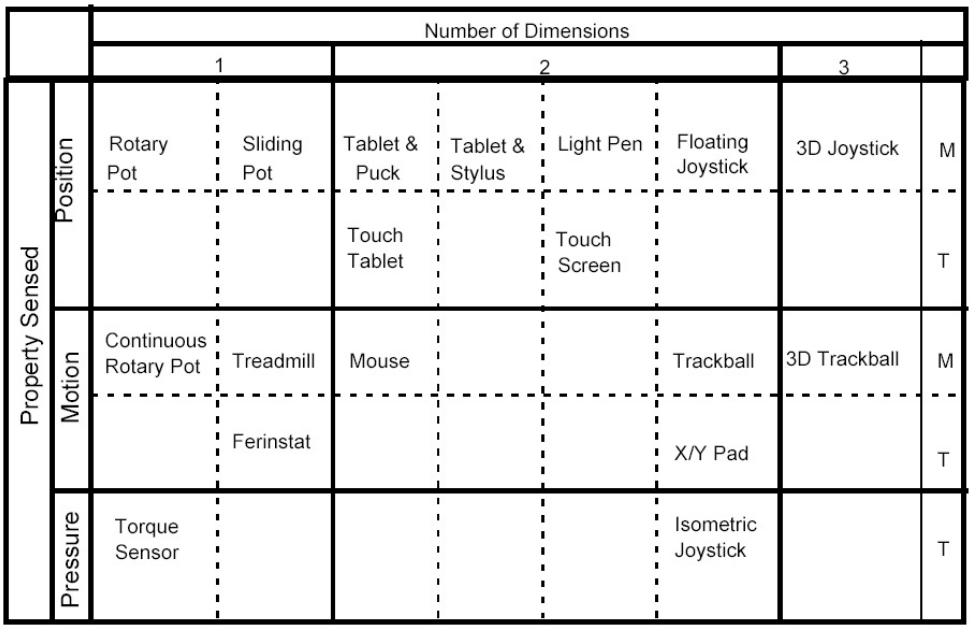


Figure 2.1: Taxonomy of input devices taken from [3]. *Devices can be categorized by DOF (large columns), sensed input property(large rows), whether they use an intermediary device (such as a stylus) or direct touch input(subrows) and comparable motor control (subcolumns).*

Hill and Rowley (1985) [5]. It has later been reformulated for direct input methods, such as touchscreens.

When objects are manipulated in general three different states can be distinguished.

- state 0: The device does not provide any input or the provided input is out of range or inappropriate. This is e.g. when a touchscreen is not touched at all.
- state 1: The device is in tracking state. That means that pointing data is returned. An example would be the mouse moving without any button pressed.
- state 2: The device which needs an extra activation method, like a button is moved while being activated. An example would be a mouse being moved with the button pressed.

In figure 2.2 we see this model applied to an graphic tablet with a stylus. The stylus has a tip switch similar to a mouse button. All dragging operations are activated with the tip switch pressed. A touchscreen can

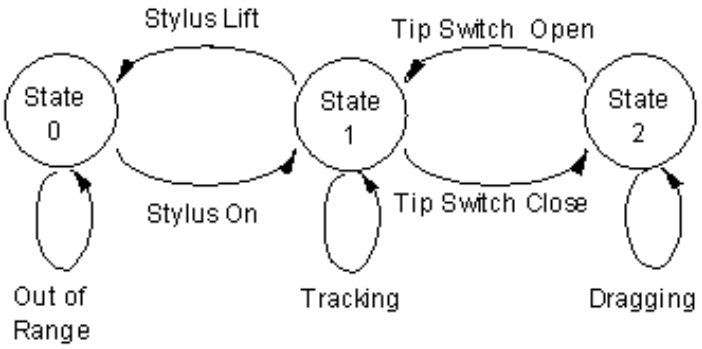
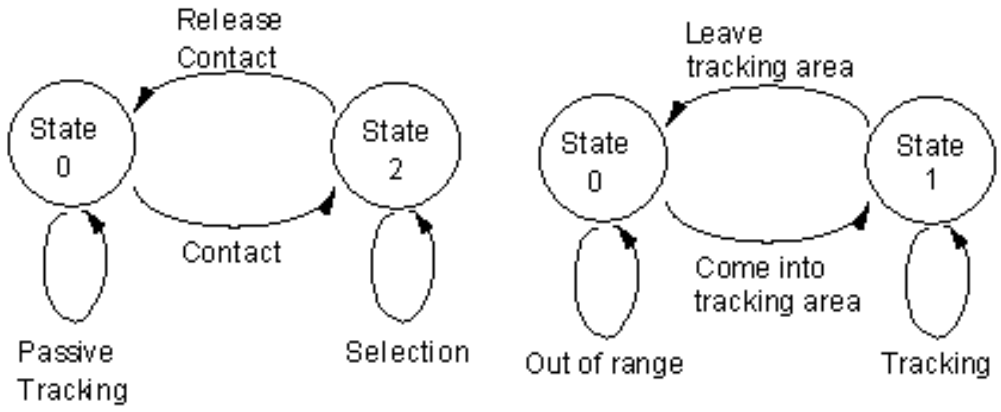


Figure 2.2: *The three state model for a tablet with stylus input with a tip switch* (image taken from [6])

naturally only distinguish two states. (see figure 2.3) State 0 is when the finger is not on the surface (OOR). We speak of passive tracking, because the system does not get any tracking information until contact [6]. With a virtual touchscreen and a computer vision tracking system we have the contrary case. The system gets tracking information all the time the finger is inside the tracking area. But there is no trivial way of detecting a surface touch which could be used to switch to state 2.



a) b)

Figure 2.3: *Three state model for a touchscreen a) and a virtual touchscreen b).* (images taken from [6])

2.1.4 Using Gestures as Input

Gestures have been established in our everyday communication.They are also suited to provide a good computer interaction method. We can distinguish different kinds of gestures([3]): Symbolic gestures, e.g. the OK sign, Iconic gestures, e.g. to point out the size of something, and dietic gestures. The last one is most often used in HCI. They are also known as pointing gestures and are especially interesting, because they are easy to detect.

Attempts to implement input methods, which make use of iconic or symbolic gestures, have been made, but these systems have not been established yet. Sign language recognition would be one possible application, but much research is necessary.

Pointing gestures on the other hand provide a direct way to select the object the user wants to work with. On the normal graphical user interface, there is already a method for using pointing gestures, known as the system pointer. When the system accepts direct pointing to virtual objects, these gestures are no longer metaphors but real. In a touchscreen situation this can be implemented in a direct manner.

2.2 Table Top and Touchscreen systems

An interesting way of integrating computers into our environment in the use of table top interfaces. Enhancing tables with this technology can be seen as a new subdomain in the field of Augmented Reality (AR) and HCI. As we will see, contrary to conventional computing, these systems provide many new possibilities.

2.2.1 Collaborative Work

When multiple users work together the process is different from that of a single worker. Gathering information together can be one problem for such a group. The discussion of such problems normally took place on a table. This has several reasons: It allows a direct eye contact and makes all objects visible for everyone. The table arrangement additionally focuses concentration of the participants on the subject.

One example, which is related to such an collaborative workspace is a table top game. This may be a board, card or miniature game played by two or more players on top of a table. Most of these games have items, coins or miniature figures which represent something on the surface. The player uses these items to act in this game world. When multiple Players

take part in such a game a lot of social interaction is needed. The table situation the players to communicate in a natural way. People can talk, look at each other, and interfere in each others game play. Therefore Table Top interaction systems are interesting for games and entertainment. In [7] an AR tabletop game is presented. It uses head mounted displays to play a tank war game for multiple players on a table surface. Participants can communicate easily. This shows that a tabletop environment can be used effective, when multiple persons take part in work or game task.

2.2.2 Table Top Interfaces

As we have seen the table is a convenient interaction space. The development of displays, touch screens, and new input devices allowed for the table itself to become an interactive device.

We talk of a Table Top computer interface if the surface of a table provides methods to display, control and modify computer objects and data. In general all Table Top systems consist of

- a table surface (interaction space)
- a display to visualize virtual objects. (Projected or screen)
- a tracking or touch detection device.

So far, many systems have been developed, which differ in technology and desired application scenario. We can distinguish Table Top systems by:

- Display technology: Where is the information displayed? Is there a screen or touchscreen embedded in the table, or is the output projected onto the surface. Does the interaction take place directly in the virtual image or is the image displayed on an external monitor? (see section Taxonomy of input 2.1.2)
- Tracking and touch detection technology: We can distinguish between real touchscreen systems, where only actual surface hits are detected and virtual touch screens using Computer Vision tracking.
- Possible interaction methods: The choice of the first two issues can very much depend on the kind of interaction, the application needs. E.g. when tangible objects, such as sheets of paper or game figures are involved, a camera based system can easily detect them.

The horizontal display mounting still induces the problem of users, supporting their arms on the table. The system may get false input and produce errors. Preventing the user from resting its arm on the interaction workspace can be done by inserting a table border. Alternatively the system has to detect and neglect this false input. This problem does not occur in a vertical mounting.

A Table Top interface can be designed for single or multi-user operation. As we have seen tabletop interaction is naturally applicable for multiple user. But even most single user systems allow the intervention of other persons, at least with limitations. Important issues especially for multi-user environments are orientation and reachability of objects.

Orientation is important in a multi-user scenario. The design of game cards e.g. shows the pictures from two sides, so opposite players both can read them. Virtual objects on a Table Top interface have to either include this feature, or be rotatable. Gestural methods for rotation or translation described in [8] could be applied here. In some cases a table top environment may even orient documents automatically in the right direction for each user.

Virtual objects have to be reachable by everyone. Thus it is not practicable when the table is to large. It has to be small enough to allow everyone to reach all necessary objects easily. Real objects the user places on the table can interfere with the tracking system. It has to detect these objects, and eventually adapt on the new situation.

An example for a Table Top interface is the Microsoft^{TM}Surface Computer [9]. Primarily developed for public spaces such as bars and shopping centers, it offers a multitude of functionalities. E.g. it allows the input of photos by Wireless transfer. A WiFi-camera which is placed on the table transfers photos to it. These pictures are displayed on a unsorted stack and can now be sorted by the clients. The pictures can be rotated and sorted by finger gestures. Zooming gestures are also supported by using two fingers. When each one is applied on an opposite corner and drawn outward a photo can be scaled. Alternative applications include an interactive water demo where the user can touch the virtual water surface and a music application.

Other table top projects are the Digital Desk [10] or the Multi-Touch tables by Jeff Han [11]. These contributions will be described in the next chapter.

2.3 Touchscreen Technologies

All tabletop interfaces have the common need to detect the users fingers, stylus or other objects on the table surface. There are a great number of projects and technologies which have its advantages and disadvantages. These systems include:

- **Computer Vision based systems**: These systems include a camera which can be top mounted or standing in front of the screen. This camera detects the users movements using image recognition. Projects such as [2] and [12] use this technique. Since this is one of the key technologies in this project we will discussed and explained it later in chapter 3.1.

- **Capacitive detection**: The detection surface here was made of a thin grid of wires, which resemble a small capacity. A hand or any other device that is brought near the surface changes the capacity. This difference can be measured and allows a detection precision which only depends on the sensing wire distance. Capacitive detection can detect multiple touch points at once. SmartSkin [13] makes use of this technique.

- **Sensing by Frustrated Total Internal Reflection (FTIR)** (): This technology was known for several years and recently became famous in the work of Jeff Han [11]. Much attention has been paid to this method because of the possibility for building this Multi-Touch device with cheap, off-the-shelf components. It uses the fact that IR-light which is normally totally reflected inside an acrylic plate, can be scattered out of the plate when a finger touches the surface. This causes a break in the total reflection and the evading light is captured by an IR-Camera. Image coordinates are calculated from these IR-blobs. This technique is also fully Multi-Touch capable.

- **Acoustic Tracking**: Almost any surface can be changed to a tactile input device by acoustic tracking. There are multiple approaches which will be covered in chapter 3.3. These methods involve measuring sound waves in the object, which are generated by the touch or changed by the touch. Methods like Time reversal, TDOA (Time Difference of Arrival) or Acoustic Holography allow more or less precise localization of the touch position.

- **Combined systems**: The above techniques can be combined in different ways to circumvent some of the disadvantages a single tech-

nology might have. In [10] a virtual touch screen system is combined with a simple acoustic tap detection. Another work, which is also highly important for our system, is the TISCH project [14]. It combines the Multi-Touch feature of the FTIR approach with a Computer Vision shadow tracker. This permits to "hover" over an item, similar to the mouse hover method. We reused the shadow tracking software of the TISCH project (TOUCHD, see chapter 5.2.1) for our virtual touchscreen system.

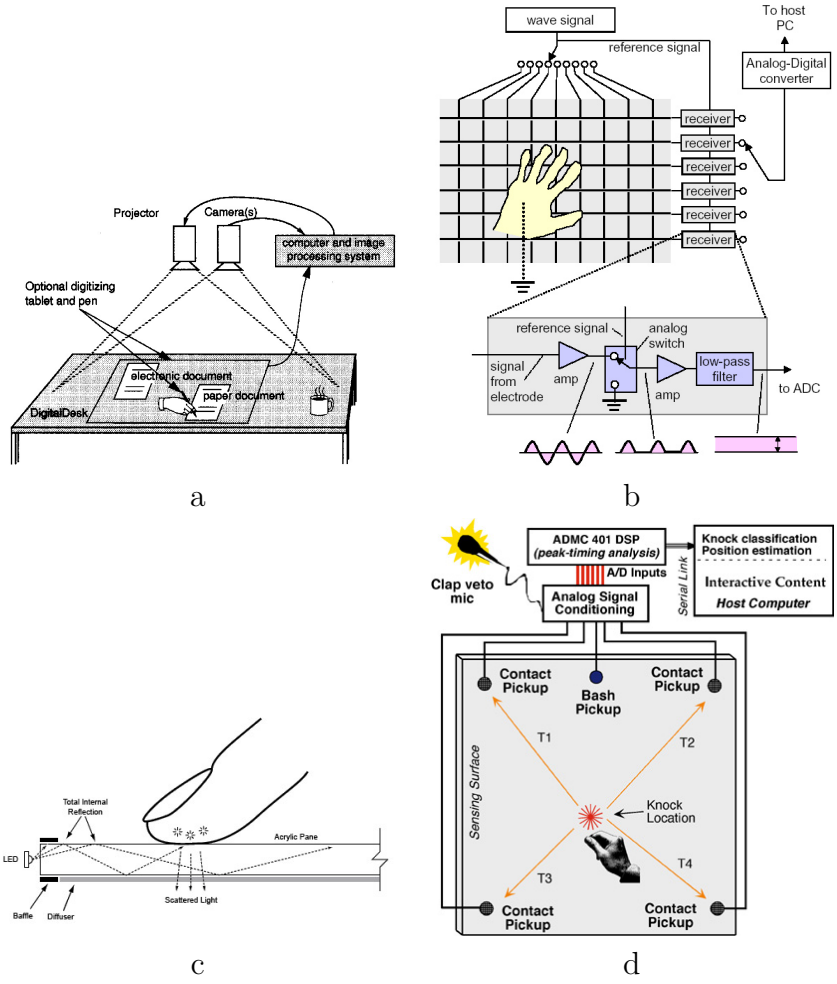


Figure 2.4: *a) Computer vision, the digital desk [10], b) Capacitive SmartSkin [13], c) Frustrated internal reflection [11], d) Acoustic tap tracking* [15]

In chapter 2.1.2 we have categorized input devices by issues like Degrees of Freedom (DOF), relative or absolute input and other input qualities. Touch screens generally sense absolute, two-dimensional positions. The special ability of a touch screen system is the direct input method. Important issues with touch screens are ([1]):

- **Pressure sensitivity**: Does the screen provide information about how hard the finger is pressed on the screen?

- **Size of touch**: Does it sense how large the contact area is? This can also be a measure for pressure, e.g when using the FTIR technique because the finger tip get broader when pressed on the surface. The size of the touch point can be a problem when objects should be selected which are smaller than the finger itself.

- **Single-touch or Multi-Touch**: First touch screen systems supported only single touches. This input method is similar to mouse input. As with a mouse, one can manipulate only one point at once. This may be enough for most applications. In ticket sales terminal, where one discreet action (choose ticket class, choose ticket type, enter money) single buttons are pressed in an sequential order. In this case no Multi-Touch ability is necessary. But input systems that can recognize multiple actions at once allow different input methods such as two handed interaction and Multi-Person systems. When gestural input is used, Multi-Touch is strongly required. [1] describes single touch interfaces as a restriction "to the gestural vocabulary of a fruit fly."

  Multi-Touch systems are introduced in several products and projects. The currently most famous is the I-Phone. It allows more simultaneous actions and more complex gestures.

- **Multi-Hand, Multi-Touch or Multi-Pointer** : Some systems such as virtual touch screens can detect only detect disconnected blobs, e.g. the whole hand as a pointing object. When the pictures of two hands overlap problems may emerge. Real touch screen systems might distinguish multiple finger tips. They are truly Multi-Touch capable. Another question is, whether the device provide data about the touched area or only a simple coordinate pair. The last case is called Multi-Pointer. In case the whole touch area is available, advanced gesture detections might be possible, e.g. dependent on the touch area outline.

- **Multi-Person**: A system that not only distinguishes different touch points but even different persons is called "Multi-Person". This is much more difficult to archive and until now not commonly used.

- **Pointing or more complex gestures**: Finger pointing gestures can be used for state 1 (tracking) or state 2 (dragging) operations in a normal graphical user interface. Other gestures could be possible. E.g. showing the desktop with a wiping gesture. Two handed or Multi-Touch gestures offer even more possibilities, e.g. zooming an object by dragging the edges apart.

- **Stylus or finger detection**: The problem with touch screens is still that it is not practicable for small targets. Targets smaller than the finger tip will not be accessible. In this case the use of a stylus as a kind of smaller finger might solve this problem. Another way would be to display a cursor which marks the detected touch point at the position of the finger touch. This allows much more precise target acquisition.

Summary

In this chapter a short introduction on HCI has been given. The usability of a computer system depends on the applied input method. Metaphors can help the user and ease the working process. The system pointer is one of them. As described, in a graphical user interface it is mostly used to select, manipulate and position objects on the screen. Input methods had been categorized by different attributes. Of special interest for pointing are devices which provide absolute and continuous input. In addition, touchscreens are special due to the direct way the input can be made.

In another section, the state model for pointer input is presented. This model can show, for different devices, how device states, like tracking, selection and manipulation of objects relate to each other. Touchscreens and Virtual Touchscreens are again special devices. Generally they do not support all three states at once. However they provide a good way to implement pointing gestures for a graphical user interface. Furthermore Table Top systems had been presented as an interesting way to integrate computer support in an inter-human discussion and groupwork process. Issues such as reachability and orientation of objects had to be considered in such an system.

Additionally different technologies for implementing these systems had been shortly introduced. It is of great importance whether these systems are Multi-Touch or Multi-Pointer capable.

Chapter 3

State of the Art

In this chapter a short overview on computer vision techniques and virtual touchscreens are given. The SiViT Terminal and its functional principle is described. Moreover, the different methods for building tangible acoustic interfaces will be presented. The mathematical background of the generalized cross correlation and time difference of arrival analysis will be given.

3.1 Computer Vision

Computer Vision (CV) connotes to tasks which are solved by the computer using vision oriented abilities. Usually input images are provided by camera. One computer vision application is e.g. the detection and tracking of features in either video or still frames. Multiple cameras can be used for a 3D reconstruction of a scene.

Systems which use a camera for simulating a touchscreen on an real or projected screen are called Virtual Touchscreens (VIT).

In a virtual touchscreen system, computer vision is applied to find the position, the user points to. The tracked feature could be either his hand or a specially marked pointing device.

Using a camera as an input device for a virtual touchscreen system has several advantages. A camera driven system can cover a large tracking area very easily. Compared to other methods which need touch sensing hardware it is easy to install. Another advantage is that it can be mounted vandalism safe on the ceiling. It does not require the user to touch the interaction surface, a fact that makes it interesting for the operation in sterile environments in medicine. Additionally it allows the implementation of gestural input.

But the CV approach has some drawbacks. Image processing is in general computational very intensive. The detection of features is strongly related to the quality of used algorithms. Another great problem which still is not solved well is, to detect, when a user actually decides to make an action or selects an object. To refer to the three-state-model a virtual touchscreen is normally in state 0 or 1, i.e. "Out of range" or "tracking". Switching to state 2 needs a special action. With a mouse this is normally done by pressing a button. For our VIT several solutions are possible:

- Design the system to accept gestures as 'click' events. For example buttons could be activated by stroking them in a certain direction. An example for such an interface is the *dontclick.it* project. [16]

- Use stereo vision to determine 3D positions of the hands. This would allow to detect a surface touch but is computational intensive. Also multiple cameras and complex algorithms are needed. Installation height and angles are restricted.

- Use hand shadows for tap detection. This is described in [12]. When a finger is approaching the surface its shadow is first visible and when the finger touches the surface it becomes sharp and small. One can measure the shadow width or the derivation of the shadow outline at the tip. When a certain threshold is crossed the surface has been touched.

- Use additionally acoustic tap detection. This method is used in this thesis and is described in detail later in this chapter. (see chapter 3.3)

Other problems, computer vision systems have to face, are changing light conditions. Near windows the sunlight can disturb the detection and fast changing ambient light intensity have to be avoided. Aside from that, shadows of the users hands and arms can interfere and confuse the tracking algorithms.

Marker based tracking can be used as a precise and simple detection method. But this always restricts the user to these special and constrictive devices. Unencumbered tracking presented in this work is much harder to archive but easier for users to accommodate. Such a system for vision based interaction consists of the following parts:

- Image acquisition: A camera which is located so that it can see the image of the interaction area. For a tabletop interface this means that the users hands are imaged e.g. from a top mounted camera. Either visual light or infrared can be used. The first has the advantage of color information. The second is invisible for human eyes and does not interfere with projected image output.

- Image processing: The camera images are prefiltered despeckled and rectified. Then features are detected, e.g. the finger tips. This can be done by edge or corner detection, background subtraction, thresholding, using the color information or motion field analysis. Some of these methods may be combined.

- Advanced processing: Once these features are extracted further processing is needed depending on what the system should do. When

gestural input is needed these gestures have to be extracted from the movement of the detected features.

An example for the use of a virtual touch screen is the PlayAnyWhere project [12]. Here a mobile projector system is presented which can be placed on nearly any table surface. The image is projected by a special beamer system with low projection height. Since the camera is mounted near the projector from a large side angle, image rectification is necessary. PlayAnyWhere includes a gaming scenario and an augmented paper system where normal white sheets of paper are tracked and images or text can be projected on them. Tracking is done by either flow field analysis or IR-shadow tracking.

The Digital Desk [10] project was an attempt to augment the office environment by using a virtual touchscreen system. It is intended for bridging the gap between paper and electronic documents in an office. Paper can be either scanned by the camera or augmented with virtual projected content. Drawing can be done by either real or virtual pens. Additionally there is a calculator application where numbers can be entered by pointing to them on a printed document. Another applied technology in this project is adaptive thresholding for hand tracking. A microphone is used for surface tap detection, though this detection can not distinguish between different hands or pointers.

3.2 Siemens Virtual Touch Screen (SiViT)

The Siemens Virtual Touch Screen (SiViT) [2] is another example for an implementation of a Virtual Touchscreen system. It was originally designed as an information display terminal for use in public settings. Our chair has been donated with an SiViT unit, which is used as a basis for this project. In this section the basic functionality of the SiViT is further described. A user can operate it, utilizing simple finger gestures. The functionality is originally limited to making "tracking" (State 1) input. The original SiViT input system was not capable of real button press equivalents. Although the button click metaphor can be emulated by the following method [1]. When the pointer stays motionless for a given time a click is triggered.

[1] We can only assume this behavior, since the original driver software was not available

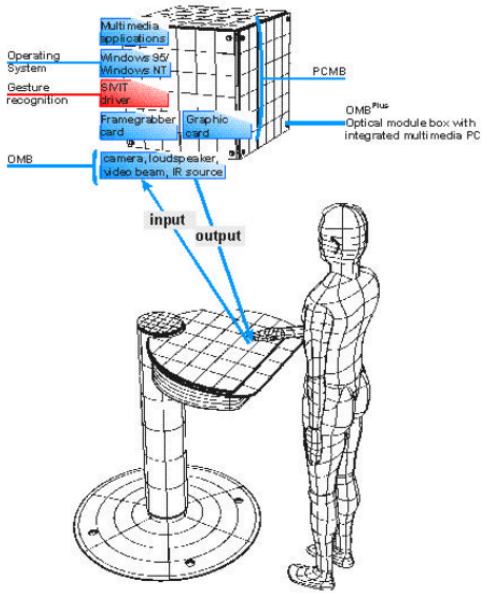


Figure 3.1: *The SiViT*

Original Setup

The SiViT originally consisted of two units: The white table surface mounted on a steel socket, and the Optical Module Box, which can be mounted either on the wall or the ceiling over the table. This OMB contains all the important hardware:

- IR Camera
- Two LED IR Spotlights.
- A Beamer to project the output on the table.
- A PC which originally runs Windows NT. This computer is equipped with an video capture card.
- The driver software library for Windows NT which implemented the image processing routines for the SiViT.

Functional principle

The SiViT uses a simple image processing method to control a system pointer. The application output is projected to the table surface. Since

a b

Figure 3.2: SiViT Components: *a) IR Camera Pentax CV-M50 , b) IR-LED Spotlight*

an IR Camera is used, the output projection does not interfere with the tracking process.

The tracking process works in three stages and with a 50Hz frame rate:

1. **IR - Image acquisition**: An IR camera image is captured.
2. **Thresholding**: This grayscale image is preprocessed, thresholded and converted to binary.
3. **Position detection**: Because the original SiViT driver software was not available we could only guess how this step is done. Possibly It might be this way: The binary image is scanned, top-down, for the occurrence of dark pixels. The first (relevant) occurrence is taken as the pointing tip. This would account for not allowing more than one hand for pointing.

3.3 Acoustic Tracking

As we have seen in the previous section virtual touchscreens and surface-covering input devices have some disadvantages.

In this chapter a different approach for building an tangible surface is described, which had drawn some interest for researchers. Touching, tapping or knocking on a solid surface produces vibration inside the body. This sound waves can be used to build a so called Tangible Acoustic Interface (TAI). Attaching microphones or accelerometers to a surface is easy and relatively cheap. Nearly all surfaces can in this way, turned to a tactile input device. The microphones record the surface vibrations and different techniques are used to estimate the sound source location.

Acoustic tracking approaches have been used in [15]. An interactive shop window is build by the implementation of a passive acoustic tap tracker. Passer-bys can browse informations about the shop or the sortiment, which are projected on the glass by knocking or tapping on the surface. Another project [17] uses acoustic tap tracking for the design of new musical instruments. Computer generated sound is controlled by a TAI. The interaction surface could be a nearly arbitrary shaped object of a appropriate material like steel or glass. Sound parameters can be varied by either pressure or volume of the surface hit or a appropriate location mapping on the surface. In this work different methods like Time Difference of Arrival (TDOA) estimation, Location Template Matching (LTM) and Acoustic Holography are utilized together since they all have different strengths. A commercial product using TAI technology is presented by [18]. *I-Vibrations* offer intelligent shop windows and Table Top Touchscreen systems based on a Tangible Acoustic Interface.

TAI devices are cheap and do not require massive installation of hardware. Unfortunately their accuracy is limited and depends on material constants and dimensions.

3.3.1 Technology

There exist different technological approaches for acoustic tracking. We can distinguish between active methods like acoustic holography and passive methods. Two very common passive methods are measuring the Time Difference of Arrival (TDOA) and Location Template Matching (LTM). Both provide good results for a number of applications and are further described in this chapter.

All technologies require a detection surface which allows sound wave propagation. Most projects have experimented with glass plate and steel whiteboards.

Most systems record the acoustic vibrations with microphones or piezo transducers. Piezo transducers utilize the piezo-electrical effect to measure pressure variations. This effect appears in some materials, i.e. Bariumtitanat ($BaTiO_3$) when pressure is applied to them. An electrical voltage is generated which is proportional to the applied pressure. These sensors are used in this project because they only record the solid-borne sound. Few environment sound is recorded from the air.

Figure 3.3: *Piezo Transducer*

Time Difference of Arrival(TDOA)

The TDOA approach tries to determine the source of sound by measuring either the time differences of the first arrival of a sound wave or the phase shift between signals from multiple microphones. Since we can assume sound speed in a homogeneous medium to be constant, these delays should only depend on the distance differences on the path from the sound source to the detectors. Figure 2.3 d shows the hardware setup for a TDOA estimation system.

Ideally two incoming signals should be similar up to the time shift which depends on the runtime of the sound signals. τ_{mn} is the time the sound wave travels longer to a microphone d_n at distance r_n than to the microphone d_m at distance r_m. c is the speed of sound in the propagation medium.

$$\tau_{mn} = \frac{r_m - r_n}{c} = \tau_m - \tau_n \tag{3.1}$$

With two sensors the source location can be calculated up to a hyperbola. To completely determine the location in 3D Space 4 Sensors are needed. Since we restrict the sound source for a TAI to the tracking surface 3 Sensors are sufficient. With more sensors better estimates could be calculated e.g by least square fitting.

To make this method usable, we have to assume that the medium is homogeneous. Since sound speed depends on the material structure, a homogeneous material is most suited. Inhomogeneous materials cause distortions and make the system imprecise. Sound speed which is about 340m/s in air, is much higher in solid media. To measure the small delays high sample rates have to be chosen. To correctly calibrate such an tracking system, the precise sound speed has to be known. Further we have to

assume a circular propagation of the waves and a nearly lossless medium.

In [15] four microphones are used which are mounted on a glass plate.

Problems arise concerning how to calculate the phase shift efficiently at the different locations. Possible methods to do this could be:

- Simply measure the time values when the signals initial flank arrives at the microphone. This is easy but requires the signal to be at similar amplitude to work correctly. It is likely to be disturbed by different taps occurring at the same time.

- Cross Correlation: This method is very commonly used and give acceptable results. Cross correlation determines the similarity of two signal vectors for a given time shift. This method is used in this work and further described in chapter 3.3.3.

Location template matching (LTM)

This method uses the fact that an impulse signal, like a tap or knock, which is scattered in a media still carries information about its source. According to the time reversal theory, after recording such a signal it is possible to exchange sender and receiver and play a time reversed version of the signal. At the former source location the signal is restored. So different signals can be mapped to their locations.

For a LTM a single signal detector is sufficient. It records the tap signals and compares them e.g.by cross-correlation with several stored template samples. These templates have to be recorded in a calibration process and mapped to their corresponding locations.

The LTM based approach is used in [19]. LTM requires more computational effort and works only for preknown locations. Material inhomogeneities do not interfere with this method, because they cause further local differences which make distinction of locations more easy.

3.3.2 Other methods

There are a multitude of other acoustic localization methods available. Some of them are mentioned in this chapter: One method is Acoustic Holography. This approach is taken in [20]. It tries to measures the two dimensional sound wave field with a microphone array. This information can be used to reconstruct the three dimensional acoustic intensity on a surface. A mathematical estimation to this problem is the Rayleigh Sommerfield algorithm. This algorithm is described in [21]. Though it

performs well, it requires a large number of microphones to get an adequate precision.

Steered Response Power and Steered Beam forming are further techniques. They are applied and described in [22]. When having multiple microphones a beam forming approach is used to build a direction search space. The signals from all microphones are summed for all delays and direction angles. Then this search space is traversed to find the global maximum. Unfortunately it has many local maxima, so the performance depends on the used search algorithm. This technique is computationally expensive and not considered in this work.

3.3.3 Generalized Cross Correlation(GCC)

We decided to implement for our acoustic tap tracking a TDOA approach which applies the Generalized Cross Correlation (GCC).

The GCC efficiently estimates the Time Difference of Arrival τ_{mn} between two microphone input signals. Generally cross-correlation gives a measure for the similarity of two signals. The information presented here is taken from [22] and [23]

The cross correlation of x_i and x_j is defined as:

$$c_{ij}(\tau) = \int_{-\infty}^{\infty} x_i(t)x_j(t-\tau)dt \tag{3.2}$$

In case of ideally identical, but time shifted signals, as in the TDOA scenario, c_{ij} is maximal for an corresponding time shift τ. The correlation, which is here done in the time domain, can be applied also in the frequency domain. As we will see this allows better processing and filtering. Fourier transformation of c_{ij} gives

$$C_{ij}(\omega) = \int_{-\infty}^{\infty} c_{ij}e^{j\omega\tau}d\tau \tag{3.3}$$

By using the convolution properties of the Fourier transform we get

$$C_{ij}(\omega) = X_i(\omega)X_j^*(\omega) \tag{3.4}$$

This is also called the *cross power spectrum*.

X_i is the Fourier transformed signal x_i and X_j^* is the complex conjugate of the Fourier transformed signal x_j. The frequency domain calculation has the advantage, that we can also apply a weighting function $W_i(\omega)$ to emphasize different frequencies.

$$R_{kl}(\tau) = \frac{1}{2\pi}\int_{-\infty}^{\infty}(W_k(\omega)X_k(\omega))(W_l(\omega)X_l(\omega))^* e^{j\omega\tau}d\omega \tag{3.5}$$

One can combine these weighting functions $W_i(\omega)$ into the function $\Phi_{ij}(\omega)$

$$\Phi_{ij}(\omega) = W_l(\omega)X_l^*(\omega) \tag{3.6}$$

Weighting 3.4 with 3.6 and using the inverse Fourier transform gives us the Generalized Cross Correlation.

$$R_{kl}(\tau) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\Phi_{kl}(\omega)X_k(\omega)X_l^*(\omega)e^{j\omega\tau}d\omega \tag{3.7}$$

To increase the performance of the GCC we need to find an optimal weighting function $\Phi(\omega)$.

3.3.4 Generalized Cross Correlation with Phase Transform (GCC-PHAT)

Different weighting functions have been proposed in [23]. Phase Transform (PHAT), Smooth Coherence Transform (SCOT) and Maximum Likelihood (ML) filters are the most important filter functions. In a reverberant free environment the ML weighting optimizes the estimation. Sound is normally reflected inside finite solid objects and the signal is distorted by these reflections. PHAT can be proven to be optimal under these reverberant conditions.

Spectral regions with a low signal-to-noise ration can also be problematic. SCOT suppresses these frequency regions. The problem with SCOT is that it does not adequately prewhiten the cross power spectrum C_{ij}.

Because we can expect massive reverberations in our tap tracking system the PHAT weighting function is chosen. (See [22] for more details.)

It is defined as

$$\Phi_{kl}(\omega) = \frac{1}{|X_k(\omega)X_l^*(\omega)|} \tag{3.8}$$

The PHAT tries to prewhiten the signal, that means all frequency bins are normalized and contribute equally to the correlation. This is appropriate for broadband signals. Percussive knocking and tapping can be seen as such signals [24]. In contrast when using narrow band signals PHAT may overemphasize frequencies with low signal to noise ratio.

The Generalized Cross Correlation with Phase Transform (GCC-PHAT) is therefore:

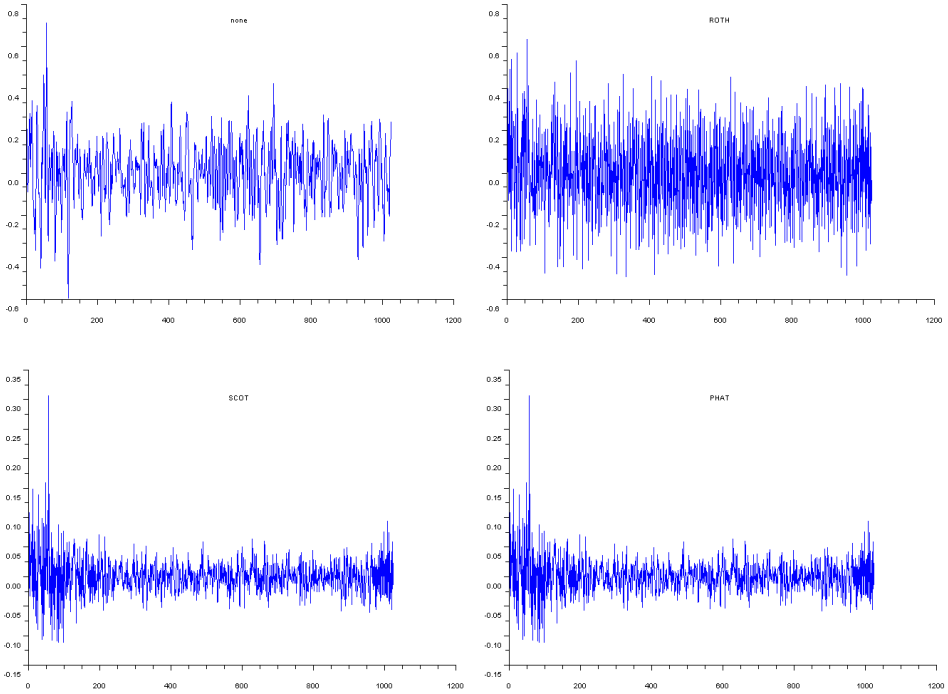


Figure 3.4: General Cross correlation functions with different weighting: *Upper left: no weighting, upper right: Roth Impulse Filter, lower left: SCOT weighting , lower right: PHAT weighting. This data is derived from the input signals shown in figure 3.5*

$$R_{kl}(\tau) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \frac{1}{|X_k(\omega)X_l^*(\omega)|}(\omega)X_k(\omega)X_l^*(\omega)e^{j\omega\tau}d\omega \qquad (3.9)$$

Figure 3.4 shows the result of the GCC $R_{kl}(\tau)$ for the input signals in figure 3.5. The GCC is calculated using different weighting functions $\Phi_{kl}(\omega)$. As we see PHAT and SCOT give nearly identical results. Both show a sharp peak at approximatly 50 samples. This obviously corresponds to the input data. Without prefiltering the correlation tends to be unstable.

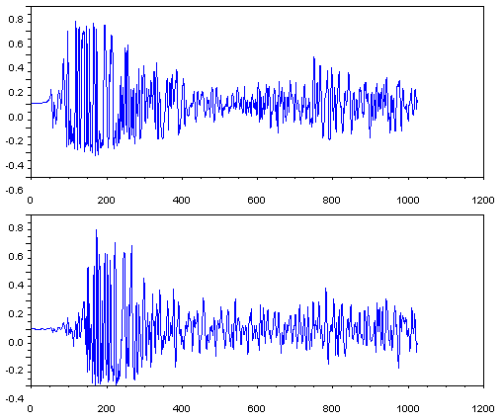


Figure 3.5: *Input data from two microphones which recorded a signal with time delay.*

Summary

In this chapter the possibilites of the two key technologies for this project are discussed. As in this thesis a VIT system is developed. This Computer Vision based approach demands further examination. A single camera approach which uses background subtraction and thresholding was described. Applying this method allows finger and hand tracking and gives good results. Though only X-Y coordinate information is given back. All actions which need selection of objects are problematic in this system. A mouse button press equivalent has to be found. Furthermore, the SiViT as a basis for this thesis was presented. Designed as an information terminal for use in public areas, it was one of the first VIT systems available. Its functional principle was similar to the here presented project.

One of the key ideas behind this thesis is the combination of a VIT with a TAI. Therefore characteristics of acoustic input systems are described. Methods, such as LTM and TDOA can be suitable alternatives to other touch technologies, especially on large scale input surfaces. Unfortunately they have low resolution. Asides from that, a TDOA approach is chosen to support the system. The user should be able to trigger clicks by tapping the table surface. For estimating the TDOA the a GCC approach is described. It calculates the time shift of two input signals by correlating them. Reverbration and noise can cause errors in this estimation. Prefiltering the cross power spectrum with the PHAT function makes it more robust against these problems.

Chapter 4

Problem statement

4.1 System requirements

In this project the development of a a table top gestural interface is described. On the basis of the SiViT (chapter 3.2) this interface should allow input by multiple hands and users. A users operates applications on a projected screen by pointing gestures. General requirements for the system are:

- At least some parts of the SiViT have to be reused.
- Optical hand tracking: Since we want to reuse the SiViT Hardware configuration, an computer vision based solution is needed.
- Mouse movements should be replaced by pointing gestures.
- An applicable activation metaphor has to be found. When using mouse input, objects are activated by a mouse button press. In our system this should be replaced by either pointing on the same place for some time or tapping with the finger on the table surface.
- Standard X applications should work with the hand input without modifications. Most software, which can be operated by a mouse, should stay operable.
- Multi-Pointer System: The requirement to support multiple users, implies the need for multiple pointer input. The tracking system needs to distinguish between at least different hands.
- Multi-User: The System should support more than one user making input at a time. This allows interesting application scenarios.

4.1.1 Tracking and projection

The optical tracking has to be stable, fast, and robust against changing light conditions. The system relies on a single topmounted IR-camera. This is sufficient for pointing gesture detection and has worked well in the original SiViT. Since the used camera provides nearly undistorted images a distortion correction is not necessary. Rectification will be not be considered, since the camera is nearly vertically mounted. The tracking system needs to find the hand and finger shapes and determine pointing fingers. These positions have to be reported to the rest of the system for further processing. For the tracking infrared light is used, the output is projected on the same surface with visual light.

4.1.2 Multi-Pointer Management

Since we use multiple pointers, the raw tracking coordinates have to be managed and translated into movements of system pointers. We want standard applications to be operable without modifications, so the pointer coordinates have to be translated into system pointer movements. Since a normal Desktop System using e.g. Linux and the X-Server Architecture does not allow for multiple pointing devices, special measures have to be taken.The Linux X-Server architecture is suitable for these modifications. We make use of the Multi-Pointer X Server (MPX) developed by Peter Hutterer [25]. This X-Server allows multiple mouse pointers. It supports legacy applications which are written for a standard X-server. To fully support the Multi-Pointer abilities applications have to be specially adapted. In this case parallel use by more than one person is possible.

4.1.3 Clickdetection

Since it is relativly easy to implement, cheap and does not require expensive hardware we decided to use a two microphone acoustic detection system. The optical tracking provides information about the finger locations on the table surface. Feedback about the height above the table is not given. Acoustic tracking in our case is not used to pinpoint an exact location, but to detect a surface tip done by a specified finger. Since we know exact tracking locations from the optical tracking, all we need is an coarse destinction between potential touch points. Using a TDOA approach will limit the possible cursor candidates for a click.

Challanges for this kind of detection might be:

- The appropriate threshold to trigger the detection has to be found. When the threshold is to low the system might confuse noises with a tap. When a high threshold is used the user has to tap very hard, which makes interaction exausting.

- The main problem is, when allowing tap detection with multiple pointers, to detect which finger and pointer has actually performed the tap. Estimating the sound source position by TDOA analysis and comparing this position with the known pointer positions should allow this determination.

- The detection has to be fast to not impede interactive work. Simple tresholding is easy to implement. TDOA analysis with GCC is known as a fast method. When working in the frequency domain

and the Fast Fourier Transform (FFT) approximation is used, it can be performed in O(n log n).

Alternativly clickdetection could be performed by using a No-Motion-click method. A click is triggered when a pointer moves to a location, and stays there for a certain time without moving. We will evaluate which method is better suited for our terminal system.

4.1.4 Calibration

Field of view of camera and beamer might be different and the image regions may not be correctly aligned. Additionally the camera coordinate system may differ from the screen coordinate system. It may be mirrored vertically or horizontally and have a different resolution. So calibration is neccessary. This could be done by transforming the coordinates by a homography which has to be determined in a calibration step using at least 4 point-point correspondences. Additional calibration is needed to align the microphone to the optical coordinate system.

4.1.5 Application

Not all applications require the full features of a Multi-Pointer environment. Most legacy programs are designed for mouse operation and work perfectly with a single pointer. Thou are several scenarios where Multi-Touch and Multi-Pointer is beneficial. To demonstrate the new features of this upgraded SiViT Terminal by now two applications are developed.

- We want the system to be usable to browse the chair website. This may probably not a difficult task and should demonstrate the application of the SiViT system as an information terminal. Text entry might be realized using a virtual soft keyboard.
- The second part should be a puzzle game, which can be played by multiple persons. Puzzle parts can be manipulated, moved and rotated using one and twohanded gestures.

The development of other applications for our terminal system may be part of future work.

Summary

This chapter explains the main system requirements. A Multi-Pointer VIT system is to be built. For this purpose the SiViT and its hardware components are either reused or replaced by up-to-date versions. The system should support both legacy applications and Multi-Pointer aware programs. The latter can handle input form two or more input pointer simultaneously. This should also allow the system to be operable by two or even more persons at once.

A way has to be found, to display multiple cursors in a standard WIMP interface. A special management of pointer coordinates will be neccessary to stabilize pointer positions and assign raw coordinates to system cursors. Additionally a calibration component will be neccessary to adjust different coordinate systems. For click emulation (selection of objects) acoustical tap detection will be implemented. This can be done in mono mode, for a single pointer system but also in stereo mode to differ between several pointers. An alternative is the No-Motion Click method.

Beyond this basic system, a Multi-Pointer aware application is required to demonstrate the new possibilities. As a starting point the input technique can be used to browse websites.

Chapter 5

System Design

5.1 Components / Layers

The basic information our input system provides, consists of pointer positions, pointer movements and table touches (referred to as "clicks" or "taps"). These input information has to be managed and usable for special and general applications. Basically we can split the system into three main components.

- Optical tracking: Captures image data and generated pointer positions.
- Pointer management: The pointer positions have to be organized and allocated to a specific operating system pointer.
- Operating system interface: This includes an interface to the applications. We want to standard applications to run on our system. So the finger cursors should appear the same way, like a normal mouse cursor to an application.

These main components are split into further subcomponents. They can be ordered in a layer model to describe the basic flow of information. These system components are:

- Hardware layer: An IR- Camera captures images from the table surface and is connected via capture hardware (WinTV Card) to the computer.
- TOUCHD: This component determines the finger tip positions from camera image data .
- CALIBD: The CALIBD process allows a calibration from input positions in camera coordinates to output screen coordinates.
- MOUSED: The MOUSED gets the calibrated camera coordinates for further processing. It manages the different pointers and simulates mouse pointer movement for the operating system and the applications.
- MPX: Normal desktop systems are build to be operated by single mouse and keyboard. MPX is a Linux X-Server which supports multiple mouse cursors. It provides the interface for our system to work with standard and Multi-Pointer applications.

- Applications: Applications have to be specially build to fully support Multi-Pointer input. Though legacy applications can be run. These will assume the complete pointer input is done with a single pointer. Painting applications might get confused this way, when drawing with two pointers at once.

Figure 5.1 shows the different component layers with corresponding data flows.[1]

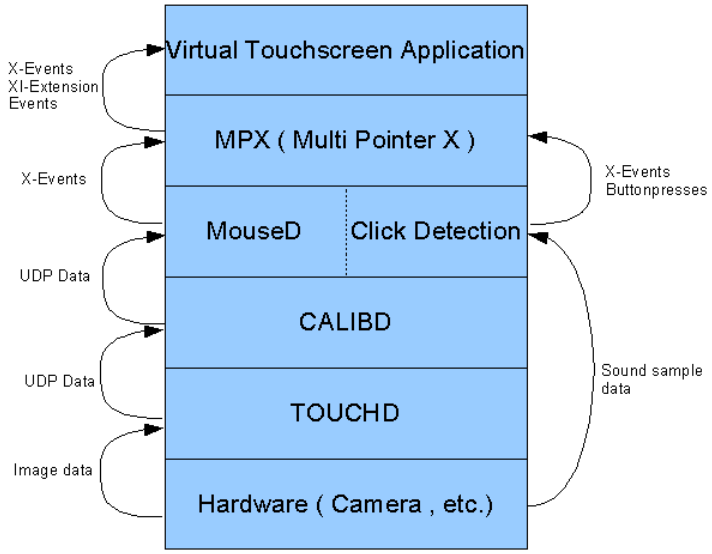


Figure 5.1: *Layer model with data flows*

In the following chapters the design of the single components will be described in more detail.

5.2 Optical tracking system

5.2.1 TOUCHD

The TOUCHD component generates 2D-Tracking positions from camera images. This component has already been used in the TISCH Project by Florian Echtler. In this project it is used to determine IR light blobs from to FTIR input. In our project it works the reversed way. Instead of light blobs the finger shadows are tracked.

[1]The names TOUCHD, CALIBD and MOUSED denote that these processes will run as daemon processes in background.

The following stages are processed in the TOUCHD component:

- The camera images are digitally captured.
- They are filtered and preprocessed.
- A background subtraction is done. This allows the detection of new objects in the image.
- The subtraction image is thresholded to generate a black and white binary image.
- From this threshold image pixel blobs are determined. These are areas which are connected and have a certain size.
- The tip of the pixel blob is determined and filtered. This is done by using the main optical axis. A special algorithm has to assure stable positions. The TOUCHD assigns an unique ID to each detected pixel blob.

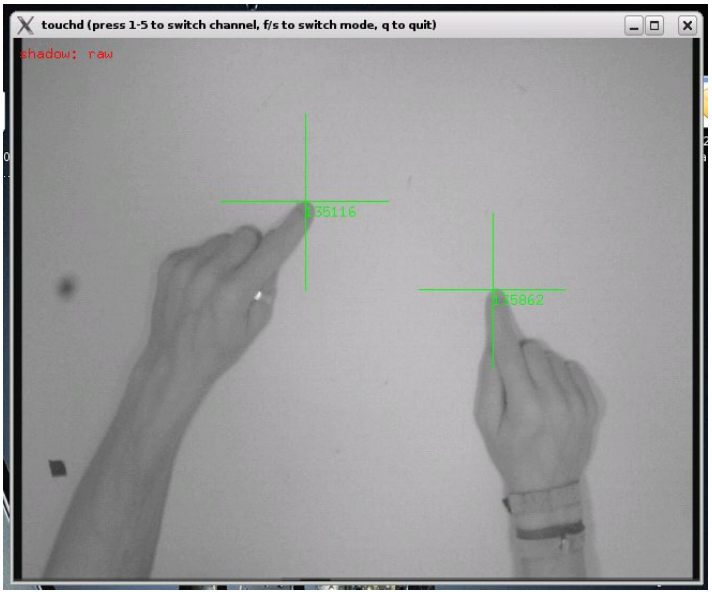


Figure 5.2: *Finger Tracking in the TOUCHD*

5.2.2 CALIBD

The determined coordinates in the camera coordinate system have to be calibrated to the screen coordinate system. This task is done via a separate

component, called the CALIBD. Like the TOUCHD, this component has been originally build for the TISCH project [14] by Florian Echtler. It takes uncalibrated coordinates x_i and sends calibrated screen coordinates y_i to the next layer. The calibration is done by applying a homography H which transforms between the two coordinate systems.

$$y_i = Hx_i \tag{5.1}$$

The homography matrix H can be determined with 4 point-point correspondences. (see [26] p.88ff) In the calibration step the four edges of the interaction space, which have known screen coordinates, have to be clicked. From these correspondences the homography matrix H can be calculated e.g. by singular value decomposition.

5.3 Pointer management

An important part in the layer structure is the management of the pointer data. Position data will be transported from the lower layers. Until now only position, size and ID of a certain pixel blob is known. The ID is changing whenever a blob disappears. These data is assigned to a mouse pointer. The transmitted positions are reached to the X-Server to control the displayed pointers.

Another aspect, that is implemented on this level, is the Click Detection. Both functionalities are implemented in the MOUSED component.

5.3.1 MOUSED

This component receives position data from the Calibration step. Position data is marked with an ID for each pixel blob recognized by the TOUCHD. The MOUSED applies these data to a pointer (mouse cursor). As long as data is send with a certain ID, this data is used to control the cursor. In case of an ID change a new ID will be applied to the cursor.

The MOUSED provides the interface to control the operation system pointers. Additionally it provides Click Detection mechanisms for system. This will be covered in the next chapter. The MOUSED component can be divided into several subcomponents.

- Receive the position data.
- Queue the data for later use. Regularly drop old data. (Timeout)

- Manage Pointer: This is the important step. The best matching, newest data is selected from the queue and applied to a pointer.
- Generate System Events: Pointer movement events are sent for each pointer. In case of a detected click a mouse button press is emulated and the corresponding X-Server Events are sent.

Alternatively calibrated pointer position data can be sent to other processes.

5.3.2 MOUSED Click detection

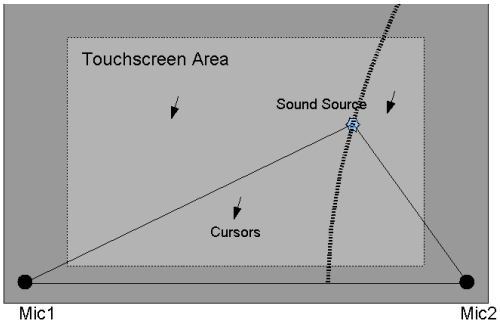


Figure 5.3: *Acoustic Tap tracker setup*

The general problem with the optical tracking is the detection of the users surface touches. We implement three different Click Detection Modes in the MOUSED component.

- No movement: The User triggers a click when he has moved the cursor to a specified location and keeps it unmoved for some time.
- Acoustical detection in "Mono" mode: The user has to tip with its finger on the table surface. All available pointers will be triggered to click. This mode is actually implemented for operation with a single pointer only. With more than one pointer it may prove not applicable.
- Acoustical detection in "Stereo" mode: Using two microphones the tipping position may be detected and the specified cursor may be triggered. The performance of this proceeding depends on the estimation of the TDOA estimation.

Since the last method is the most complicated, the design of the stereo detection process is depicted here.

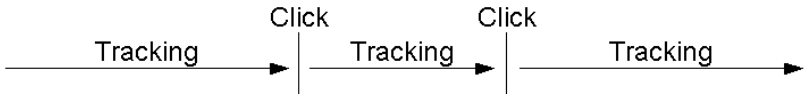
To estimate the correct cursor the following steps are taken:

- Sound sample data is recorded by two microphones.
- The Time Delay of Arrival is calculated using the GCC-PHAT method (see chapter 3.3.4). With known sound speed the position can be calculated up to a hyperbola.
- A calibration (e.g. using a homography) transforms cursor coordinates in microphone coordinate space. All pointer positions are tested for lying near to the hyperbola. The nearest cursor is chosen and activated.

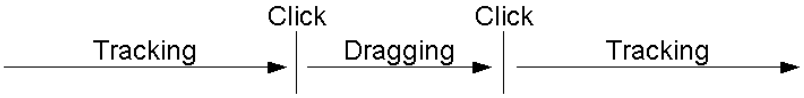
Drag and Drop Mode / Three State model

We want to utilize the three state model in the design of the MOUSED driver. As described in chapter 2.1.3 we consider three states: Out of range (OOR, state 0), Tracking (state 1) and Dragging (state 2). When no blobs are detected in the TOUCHD layer, state 0 is assumed. In this case the pointers remain on their previous position. If blobs are detected a pointer has to be chosen to change to the "tracking state". To fully support all use cases, tracking is not sufficient. For example moving of objects is solved by Drag-and-Drop in most WIMP interfaces. State 3 or the "dragging" state has to be used. When should we change to state 3. On a mouse the button is pressed and held. In our system we do not have the ability to hold the button pressed. To allow drag-and-drop actions we introduce a special drag-and-drop mode. We simply assume that clicking once switch the clicked pointer to the "dragging" state, similar to keeping the mouse button pressed. Another clicking switches back to normal tracking mode. Figure 5.4 depicts this method.

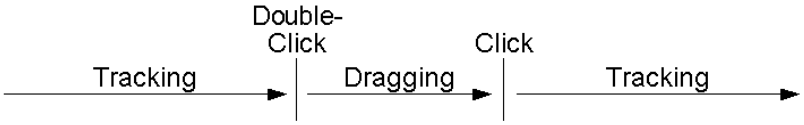
Alternatively the state change could be activated by a double tap. We simply measure the time between clicks of a specific pointer. In case this time is below a specified value, (e.g. 0.2 sec.) we switch to "dragging" state. A simple click would be sufficient to switch back to normal "tracking". In case we want to e.g. move a GUI-window, one could think of taking it up with the double tap, moving it to the target location and letting it drop with a single tap.



a)



b)



c)

Figure 5.4: Tracking and Drag-and-Drop mode: *a) Tracking Mode, b) Drag-and-Drop Mode, c) Drag-and-drop triggered by double click*

5.4 Operating system interface

To allow all applications in principle to be controlled by our input system it is necessary to make use of the standard system cursor input. What we want is to control the mouse cursor with the finger gesture input system. The cursor is displayed as usual. This additionally has the advantage that in case of improper calibration the user still knows where he is pointing to. The MOUSED transfers position data to system mouse events. Now a way has to be found to display these pointers in a normal graphical user interface.

5.4.1 MPX

The operation system of our choice is Linux, which makes use of the X-Server architecture. This allows us to simply apply a X server display system which was designed to allow more than one mouse cursors. MPX [27] is designed to support up to 128 input devices, to support several users each having their own mouse and keyboard. We use this ability to control

multiple mouse cursors and emulating their events. It makes use of the XI Extension events which allow an distinction between different devices. Mouse cursors can be generated dynamically and assigned to a physical device. We will explain MPX in more detail in chapter 6.

5.4.2 APPLICATION

Applications running on our system can be either standard legacy applications or make use of the MPX Multi-Pointer extensions. In the first case nothing special is needed to run the application on the system. These applications make use of the core events send by the slave devices. Though this can cause pointer jumps when more than one pointer is inside the same application window. For the application there is still only a single pointer, which then seems to jump between the real pointer positions. So e.g. for a paint program using only core events our Multi-Pointer environment would not work. But for our first scenario, the user browsing the chair website, it may be right because Multi-Pointer aware applications have to use the XI-Extension events sent by MPX. This allows distinction between the different inputs. To demonstrate this features we designed a puzzle game application. It consists of rectangular parts with numbers on the edges. These parts can be moved and rotated. The goal of the game is to complete the puzzle, all edges with similar numbers have to be adjunct. This game registers for the XI-Extension events. So it can use full Multi-Pointer abilities.

Summary

The presented system can be seen as a layer model. Here we describe how the individual parts work together from a black box view. Input data is captured by the hardware devices , i.e. the camera and sound interface. The TOUCHD component generates blob positions from the camera images. The transmitted positions have to be calibrated to screen coordinates. This is obtained in the CALIBD component. Calibrated coordinate packets are send to the MOUSED. This component is composed of two subparts. Pointer management and click detection. The first part handles the emerging blob coordinates and transfers them to system pointers. Clickdetection is done in three modes. These are the Mono-, stereo and No-Motion click detection. Double tap to activate the drag-mode is proposed here.

The described system should have more than one pointer. The Multi Pointer X-Server(MPX) is a good solution for this problem. MOUSED generates X-Server events which are transported through the MPX layer to applications. These can either listen to Core X Events or to XInput extension events. The latter allows the distinction of pointer devices. Legacy applications e.g. the webbrowser receive the core events and keep operable.

Chapter 6

Implementation and Testing

In this chapter details of the single developed applications are described. Special attention is paid to the pointer management and click detection functionality. In another section the demo game Multi-Touch Puzzle is explained.

6.1 Implementation Stages

The development of the input system was done in several stages.

- Due to limited labspace we decided to build a small test mockup made up of aluminum profiles.(see figure 6.1 c) The camera and an Infrared spotlight were mounted on this rack. Image capture is done with a WinTV Capture card . The "low profile" size of the Dell Computer proved problematic at first. The slot plate of the capture card had to be shortened to fit into the computer case. Fortunately the V4Linux driver fully supports this capture card.

- In a second step the MOUSED was developed and first test runs could be made with a single pointer. Click Detection was not yet implemented.

- The MPX X-Server was installed on the system. Now several cursors are possible. (see figure 6.1 b)

- We decided to utilize the GLUT Library [28] for our new Multi-Pointer application. GLUT already allows the registration of standard X mouse events for graphical window programs. We modified it to also handle XI extension events. New callback functions for these events had been included in the GLUT Library.

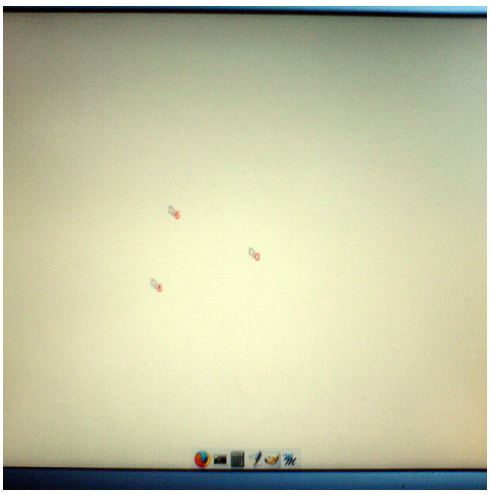
- The Multi-Touch Puzzle Game was programmed. It uses the XI Extension and the new MPX functionality.

- The click detection by acoustic tap tracking was developed and included in the MOUSED program: This is described in more detail in the chapter 6.2.3.

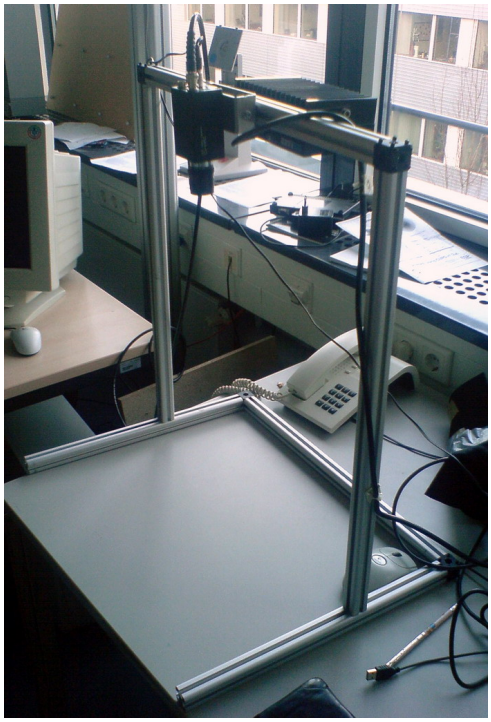
- The final stage was the assembly and testing stage. The system is build into the SiViT frame. The optical components were installed in the Optical Module Box. (figure 6.1 a). Afterward the OMB was mounted in the SiViT Rack.(figure 6.1 d)

For details about the used hardware components consult the appendix 7.2

a



b



c

d

Figure 6.1: *SiViT Assembly*

6.2 Implementation Details

Here the implementation of the single components is described. For performance reasons all code is written in C++.

6.2.1 UDP Data format

As described in chapter 5.2.1, the TOUCHD estimates the tracking coordinates for the optical tracking. These coordinates are sent via UDP packets to the next layer, CALIBD and MOUSED. Different ports (sockets) are used by both TOUCHD and CALIBD so either the uncalibrated or calibrated values can be used. An additional feature that was implemented in MOUSED was, to send managed pointer coordinates via UDP packets to another application. This function has been introduced to support the applications from the TISCH Project.

UDP packets are sent for each frame:

- First a frame initialize packet is sent with the format:
  `frame #number`
- It is followed by the data of all recognized finger shadows. These packets have the following form:
  `shadow focusxpos focusypos size id 0 pointingxpos pointingypos junk junk`

These values have the following meaning:

- `shadow`: This denotes that the tracking has been done using tracking of the finger shadows. The TOUCHD is build also for tracking FTIR finger blobs. In this case the first value would be `finger`.
- `focusxpos, focusypos`: This is the position of the center of a recognized shadow blob. For our project it it not used.
- `size`: The size of the shadow blob. This is important for the tracking. A bigger size may be more likely a users finger, hand or arm. Small blobs are neglected by the TOUCHD and not send to the upper layer.
- `id`: The unique ID of the tracked blob. If a blob vanishes and subsequently reappears its ID has changed. IDs are assigned in a successive way so the ID is always increasing.
- `pointingxpos, pointingypos`: This is the position of the calculated finger tip. These values are used for the pointer control.

6.2.2 MOUSED

The MOUSED program is very important for the system. It is implemented in C++ and comprises two stages, the pointer management and the click detection. In this chapter implementation details for the first part are provided.

The MOUSED manages a number of pointers. If the number of packets with different IDs exceed the number of managed pointers only the biggest blobs are considered. Each managed pointer takes data with a specified ID. So every blob is controlling a pointer position.

Incoming packets are queued and can be chosen from the queue depending on corresponding blob size or on location issue. This queuing has the advantage that in case of a disappearing ID the best blob can be chosen and assigned as a successor. So if a pointer gets no new data with a specified ID, it will soon get new incoming data from another blob with a new ID. One way for doing this assignment would be to take the biggest blob in the queue. We choose this method in case a new blob emerges and there are pointer objects which are currently inactive. It is reliable since bigger blobs are more constantly tracked. The alternate method would be to take the blob with the closest position to the disappeared. In case of the timeout of an active pointer this method is chosen to give the pointer a smaller position jump.

Old data will be discarded after some timeout has been reached. So the queue can not overflow.

Main loop

The main loop of the MOUSED is as follows:

```
Receive packet with ID in blocking mode

If there is a pointer listening to this ID
   feed packet directly to pointer
else
   Push packet to queue with timestamp t
endif

if there are inactive pointers
   Take biggest blobsize packet from the queue
   Assign this ID to the new pointer
   Remove timeout packets from the queue
endif
```

```
if there are any pointers who have been assigned
but idle for more than 0.2 sec.(timeout pointers)
      Assign packet with the nearest location to the old
      pointer position

Process Click Detection

Send Pointer Positions to the X Window system
```

For the timeout value of a given data, an empirical value of 0.2 sec is choosen. This value is short enough not to disturb interactivity. Though when the ID value changes some packets can get lost when there are no inactive or timed-out pointers.

The receiving is done in blocking mode. This is advantageous because in case of no incoming data busy waiting is avoided. Mouse events are generated for the X-Window system every time a managed pointers gets a new position or is called by the click detection after a click event. MOUSED maintains MPX device objects for each pointer. The positions are handed to these devices and XI extension events with a corresponding device ID are generated. This way, applications can distinguish between different pointers.

6.2.3 MOUSED Click Detection

The click detection is the second part of the MOUSED component. The three modi of detection and their implementation are described here.

No-Motion Click

This method triggers a click when the user has moved a pointer to a location and stopped the movement for some time. This is directly implemented in the pointer objects itself. The movement speed is calculated as

$$v = x/\Delta t$$

where v is the movement speed, x is the movement width and Δt is the time between two calls of the calculation.

To make this more insensitive for short stops a sliding median method is applied.

$$movement = movement * 0.4 + v * 0.6$$

If *movement* drops below a certain value a timeout counter is started. When it has elapsed without new movement a click is triggered. The method is very simple but needs some training for the user to work with. Also it is likely to generate false clicks when the user rests his hands on the surface without moving.

Acoustic Click Detection

For our tap detection we use two Harley Benton HB-T piezo transducers. These are actually acoustic guitar transducers, which can be easily applied to the table surface. Figure 3.3 shows an image of the transducer. The transducers are applied in a distance of 60cm to each other on the bottom side of the table. Figure 6.2 illustrates the transducer placement.

Figure 6.2: *Microphone placement on the table*

On the software side the acoustic click detection runs in its own threads. Two threads are used to provide synchronous sound capture and processing.

The capture thread records the signal from the microphones into a buffer. A sample rate of 48kHz is used. Unfortunately the sound card does not support a faster sampling rate. We chose a buffer size of 1024 samples for each channel. This is sufficient for the detection of multiple taps per second. When the buffer is full a system signal is set, and the buffer is swapped with a second buffer. The filled buffer is analyzed in the detection thread. It waits for the wakeup signal and starts detection on the buffer data. It scans through the detection buffer and searches for values exceeding a certain threshold. If the threshold is exceeded a number of times, a click is triggered.

Capture Thread

```
While (true)
   Record Sample Buffer

   if Sample Buffer is full
      Swap Sample buffer with Detection buffer
      Send Wakeup Signal to Detection Thread
   endif
loop
```

Detection Thread

```
While (true)
   Wait for wakeup signal from capture thread

   Search Detection buffer
      if Detection buffer contains more than n values exceeding Threshold
         Trigger Click and eventually start stereo detection
      endif
loop
```

Mono acoustic click detection is relatively simple. But if we can not decide which pointer has clicked. Activating all pointers is inappropriate and will cause error. A solution could be to activate only pointers which are currently not moving.

Unfortunately the cursor is currently still a bit shaky. This could maybe solved by using better algorithms in the tracking system. Future improvement is necessary. Another problem is caused by the finger tip itself. It is hard to hit small targets because tipping on the table causes the finger to move around, mostly in Y-direction. We can solve this problem by buffering the cursor positions. When a click occurs the system uses a previous location some milliseconds before.

Stereo Click Detection

Stereo click detection is much more complicated but provides the ability to detect information about the location of the users surface tip.

The detection process uses again both threads. Like in mono mode the detection is activated by exceeding a certain threshold value, this time on both channels. Figure 6.3 illustrates the process. When the threshold is exceeded a number of times the detection thread starts for copy the sample

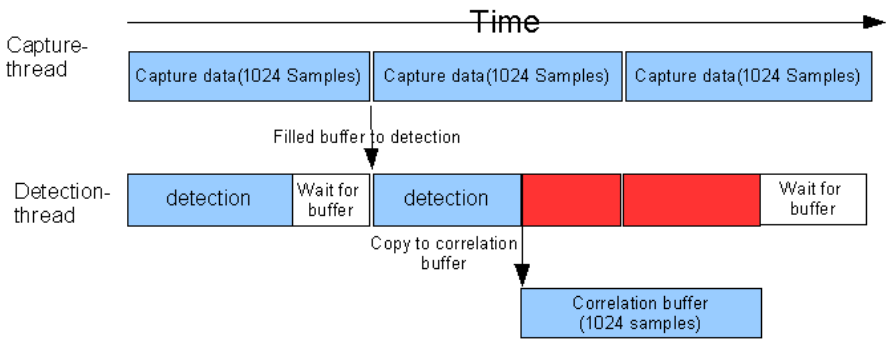


Figure 6.3: *Capture and detection threads*

data into a correlation buffer. It begins with the sample data exceeding the threshold. To completely fill the detection buffer the correlation buffer is filled from the next detection buffer. Now the correlation is done using the GCC-PHAT method described in chapter 3.3.4. All Fourier transforms are done using the *FFTW*[1]- Library. These library provides an implementation of the Fast Fourier Transform in O(N log N) when using a 2^N transformation size.

We search for the time delay τ which maximize equation 3.9. Knowing the sample rate this gives us the runtime difference (TDOA) and sets a hyperbola on which the sound source must have been.

Microphone to Pointer Space calibration

Every pointer with its known position can be tested against this hyperbola. The Hyperbola components are only available in microphone coordinate space. To perform the pointer test we first have to apply another calibration, to map the pointer coordinates (screen coordinates) to them.

The problem is here, that our acoustic tap tracker in the current implementation, does not provide positions but only a single time delay. Therefore our calibration has to rely on a manual measurement of the screen corners. From these four positions a calibration homography can be calculated. Since theses positions do not change in operation we do not need to recalibrate them frequently.

To check the pointers for click events, a first approach would be to calculate a value c for each pointer position. c is proportional to the distance of a pointer to the estimated hyperbola.

[1]Fastest Fourier Transform in the West : *http://www.fftw.org/*

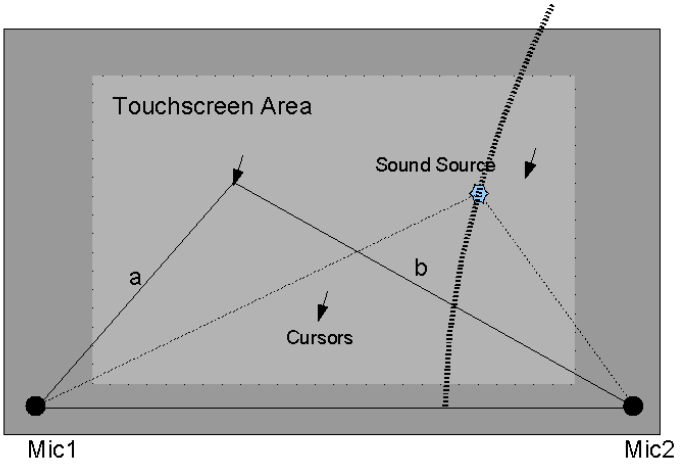


Figure 6.4: *Simple hyperbola test. Each pointer is tested for being close to the hyperbola.*

$$c = (|\vec{a}| - \left|\vec{b}\right|) - |\Delta\tau * Speedof Sound| \tag{6.1}$$

where $\vec{a}$ is the vector from the first microphone to the pointer, $\vec{b}$ is the vector from the second microphone to the pointer and $\Delta\tau$ is the estimated time difference. The pointer with the lowest value c is selected and activated.

Unfortunately this method proved very unstable. Reverberation in the table plate is very high so we had to consider a different method. The problem is that our correlation function $R_{kl}(\tau)$ (equation 3.9) has to many local maxima which makes distinction difficult. Luckily we do not need exact values because we already know the pointer coordinates from the optical tracking system. This gives us possible locations of the local maximum corresponding to our time delay. When a and b are the pointer distances to the microphones and c is the speed of sound then

$$S = \frac{(a-b) * Samplingrate}{c} \pm \Delta S \tag{6.2}$$

gives the desired search range. We look for the maximum at these locations. In our current application we choose ΔS as ± 10 Samples. Considering a maximal time delay of ≈ 48 Samples we can further constrict the search range.

In pseudo code the detection approach can be described as:

```
While (true)
   Wait for wakeup signal from capture thread
```

```
  if correlation buffer is empty
   Search Detection buffer
      if Detection buffer contains more than n values exceeding Threshold
         Copy Detection buffer from beginning with detected position
      endif
  if Correlation buffer is in use
     Copy remaining values to fill correlation buffer
     execute the GCC-PHAT
     Calculate search ranges for each pointer
     Search for the maximum at the calculated locations
     Test Pointers against hyperbola spanned by best result
     Trigger click on best match
loop
```

This approach leads to better result compared to a simple search.

6.2.4 Event generation in MOUSED

As we have see in the System Design chapter, MOUSED generates X-Events to control X-Pointers and applications. The Multi-Pointer X Server has special features which should allow input from various pointers.

MPX event handling

For handling input events there exists two different classes of events in a X Server. Core events are defined in the Core X protocol [29]. XI Events have a device ID attached, with allows an assignment to the event causing device. The XI Events are defined in the XInput Extension protocol [30]. There are also two classes of devices in the X-Server. Physical devices and virtual input points. Virtual input points normally send core events. Applications register with these events. This causes the server to deliver an event to a specific client application. In every X-Server system exists a virtual input device, called the Virtual Core Pointer. Physical pointing devices are linked with it and send core events through it.

The Multi-Pointer X Server handles input events in a special way. MPX also distinguishes input devices in two classes: Virtual devices, which are called Master Devices and physical devices, which are called Slave Devices. Master pointing devices control a graphical cursor in the X-Server. Each master device can be attached to a slave device. In case a slave device is active, three different events are sent. A core event is send by the

slave through its attached master. XI-Events are sent, both by the slave and master. The advantage of this system is the flexible attachment and the legacy application support. Legacy applications register for the Core event. New Multi-Pointer applications register for the XI-Events and can distinguish the pointers by their ID.

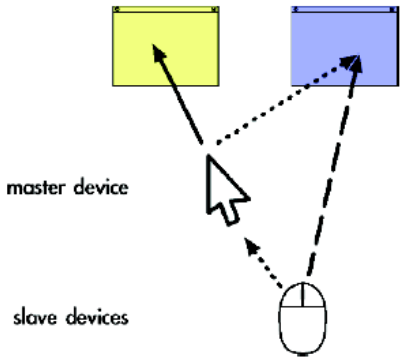


Figure 6.5: *Event handling in MPX*

Problems may arise when applications listen to core events and XI at once. To avoid ambiguities only one event is delivered in this case.

MOUSED Device handling

We want legacy applications to get the core events and MPX-Aware applications the XInput Events. The MOUSED lists the available slave devices and assigns them the pointer data. The slave devices have to be preconfigured in the *xorg.conf* file of the server. We configured four pointers for our terminal. This is sufficient for most use cases. We do not expect more than two people using the system at once. (For a configuration of the X-Servers *xorg.conf* file see appendix 7.2).

At system startup we generate master devices and link them to the corresponding slaves. This is done with the XInput tool which allows this in a fast and simple way.

We now use the functionalities of the *XI*(XInput) and *Xtst* (XTest) libraries. Movement of a pointer is implemented utilizing the *XWarpDevicePointer* function. This generates *XMotionEvent* Core-events and *XDeviceMotionEvent* XI-events and positions the corresponding pointers. Unfortunately the XI library does not contain a function for emulating button presses. Therefore we applied the *XTestFakeDeviceButtonEvent* function from the X-Test library to send button press and release events.

6.2.5 GLUT modifications

For the development of a Multi-Pointer application we needed a toolkit which allows the easy implementation of a OpenGL context window, as well as receiving mouse events in this window.

The GLUT (OpenGL Utility) Library provides many useful functions for such an application. We decided to use this Library, respectively the FreeGLUT project, available at [28], for our graphic application. FreeGLUT is very stable and has hopefully fewer bugs than the standard release. It is also platform independent. which might become important in current project enhancements. Though it is written in plain C, which does not necessarily add to code clarity.

GLUT provides an easy and fast way to apply graphic context windows. This makes it interesting for our demo application. It also allows the registration of mouse callback functions. These functions will be called by GLUT in case of a mouse movement or button press. In the GLUT event loop, X events are fetched from the event queue of the GLUT application. There are a number of captured events. The important core events for mouse movement and button presses are `MotionNotify` `ButtonPress` and `ButtonRelease`. These events are sent whenever the mouse has moved or a button is pressed. All events which are not specified in the GLUT event loop are possible XI Extension events.

These events are handled specially. The device ID of the XInput event allows a distinction between different pointers. Input extension event types are not specified from the beginning but have to be created dynamically at runtime. GLUT now registers these types. It differentiate three types: mouse motion, mouse button presses and keyboard presses. If a XI event is send to the application the corresponding callback function is called.

The new callback functions `glutXExtensionMotionFunc` and `glutXExtensionButtonFunc` receive positions and button presses together with device IDs. This modified FreeGLUT is used in the following demo application.

6.2.6 Multi-Touch Puzzle

The Multi-Touch puzzle is designed to be a true Multi-Pointer MPX application. It uses the functions from the modified GLUT library described above.

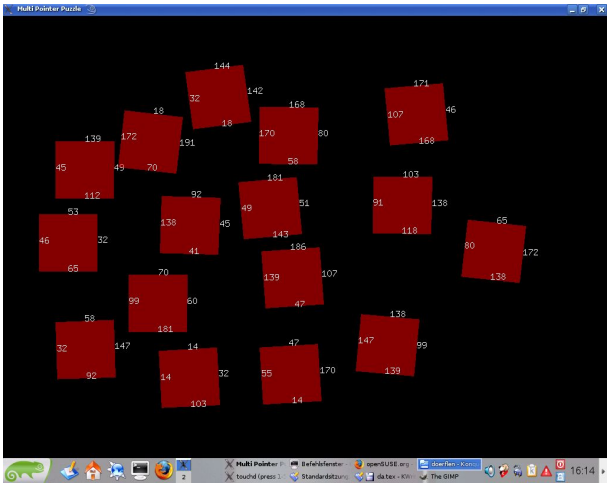


Figure 6.6: *The Multi-Touch Puzzle*

Puzzle part data structure and puzzle generation

In this game a puzzle consists of rectangular parts. Each edge of the parts has a number. Only edges with the same numbers can be combined. The goal of the game is to combine all parts to a rectangular shape, with matching edge numbers.

The important values a puzzle part stores are its position (a vector) and orientation (a matrix). Additionally for each edge the neighboring part for a connection is saved.

When the puzzle is generated first a grid of x * y positions is generated. Then edges between these positions are defined and neighboring relations are set. These edges get random indices which are then adjusted so neighboring edges have the same index. The number of connections for each edge is counted. Then the parts are disconnected. In each part the vertex coordinates are set relative to vertex 0. The parts are distributed and rotated randomly in the window area.

Gesture tracker/ movement and rotation

The Multi-Touch puzzle allows movement with one pointer and rotation with two handed interaction. A simple gesture tracking system was implemented to detect movement and rotation gestures. When a part is activated by a click or tip the received position date is feed into the gesture detection. Each puzzle part has a corresponding gesture tracker which detects gestures applied on this part. The coordinate point where the puzzle part is clicked is called a tracking point. As long as the pointer remains

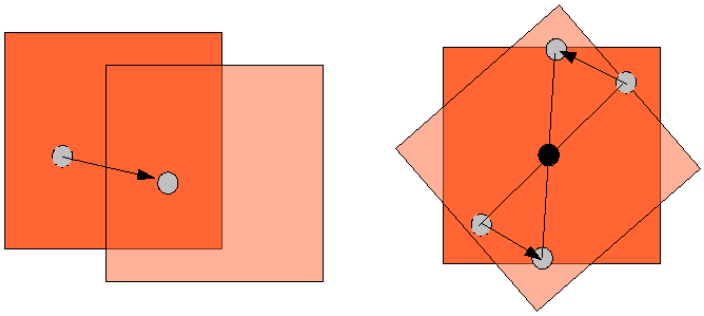


Figure 6.7: *Translation and rotation of puzzle parts*

clicked it transmits input data to the gesture-tracker.

To detect whether a part is clicked the received pointer coordinates are checked against the bounding box of the part. This allows fast and easy detection.

The gesture tracker keeps track of the number of tracking points, feeding input to it. Depending on that, the gesture is chosen. With one tracking point only motion is possible. The relative movement of the tracking point is calculated and the part is moved accordingly. When two tracking points are available, two interactions are possible. Only motion and rotation are implemented so far, though a zooming gesture would be possible. For the rotation we decided to implement two handed interaction. Rotation around any rotation center inside the part is possible. To rotate a part two cursur have to click inside the part. Rotation is then applied around the center of their connecting line. Both rotation and translation are illustrated in figure 6.7.

Part connections, disconnections and win test

The single parts can be connected with their edges. Two parts match when they have the same numbers on their connecting edges. When a part is released close to another part with a matching edge they snap together and change their color from red to blue. Internally this is solved by checking the edges of all parts for the same index and spatial proximity. When these conditions are met the parts are connected. The connection data in the corresponding edge structures are adjusted and the number of connections counter is increased. When a connected part is moved it gets disconnected, the edge connection data on both sides is reset and the number of connections counter is decreased. Since for every part the number of needed connections is reached when the puzzle is solved, the

win test only has to check each part for this attribute. Now a win message is displayed.

6.3 Accuracy and Operation of the system

Here the measurement of the speed of sound and the acoustic tap tracker accuracy are described. Furthermore operation problems are explained.

Measurement of speed of sound

The speed of sound in the table plate is measured with a simple method: We record the stereo signal with an audio recording software [2]. Then we produce a knock sound on the line of the microphones but left or right of both. Now we can count the number of samples N the left or right signal arrived earlier at one of the transducers. d is the distance between the microphones and s is the sampling rate.

$$c = \frac{d * N}{s} \tag{6.3}$$

We measured a difference of 48 samples which gives a speed of sound of 600 m/sec for a sampling rate s of 48kHz and a distance d of 0.6 meter.

Click Detection

As we had already mentioned in the previous chapters, the stereo click detection is quite unstable.

Figure 6.8 shows test detections for different locations along the baseline of the microphones. In this case a simple maximum search in the correlation function has been used.

As we have seen, there are too many maxima in the correlation function. These are caused by the noise and reverberation in the table surface. Choosing the right pointer is more or less a matter of luck. The new method, described in 6.2.3 gives much better results. We counted the number of correct detected pointer clicks for several distances to estimate the error rate of the system. A click is correctly detected if a tap on the table with one hand results in a click of the pointer controlled by this hand. We successively tapped 20 times with one pointer and count the times this pointer is estimated correctly as the clicking pointer. Figure 6.3 shows the results of the test. Expectedly, as the distance gets smaller the error rate

[2] e.g. Audiosity *http://audacity.sourceforge.net/*

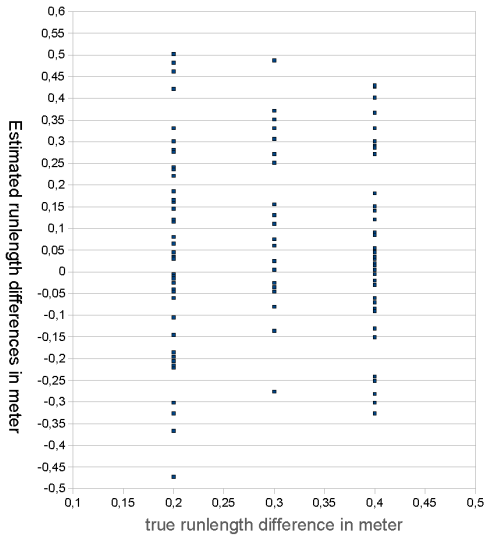


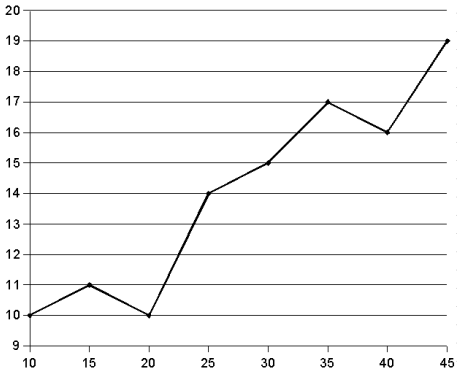
Figure 6.8: *The estimation of the sound run length difference.* $\Delta T * SpeedofSound$ *These estimates are taken with a simple search for the maximum in the GCC-PHAT correlation function.*

increases. Pointer distances under 20 cm are likely to lead to erroneous results.

Usability and known problems

The terminal system that is described in this thesis, allows the operation of basic applications in an adequate way. It is feasible for tasks such as navigating in a website or starting applications from the task bar. The movement of the pointers is fast and stable. Dragging items by double-tapping them is serviceable, let them drop by another click would be an obviously following action. Hitting small targets is more difficult but simplified by the pointer icon. To improve cursor visibility we installed a cursor theme for the window manager. The cursor icons on this theme are bigger than the fingertip and visible. Aside from that, the unexperienced user may need some accommodation time to learn making clicks by tapping correctly. Furthermore, there are some problems which limit the usability of the input system.

Objects on the table can confuse the tracking and grab a pointer for themselves. That means, this pointer is hard to get back to the fingertip of the user, in particular when the user does not see the confusing object at once. E.g in testing, the keyboard, which was places on the desk often



Pointer distance	Correct clicks	Error rate
10 cm	10	50%
15 cm	11	45%
20 cm	10	50%
25 cm	14	30%
30 cm	15	25%
35 cm	17	15%
40 cm	16	20%
45 cm	19	5%

Figure 6.9: *Error rate for the problem of choosing the correct pointer for click. 20 test clicks are done with two pointers and various distances. The pointers were keep motionless.*

confused the pointers and introduced errors in the tracking system. The inexact estimation of the acoustic tap tracking is most problematic. As we have seen pointers which are near each other can not be distinguished clearly. This badly reduces the usability of the system, because it confuses the user a great deal and tends to activate unwanted actions. In addition this impairs the performance of double-clicks and drag-and-drop actions. Unfortunately it is hardly impossible to use the implemented rotation method in the Multi-Touch puzzle game because on the small distance these pointers could not be distinguished. Improving the tap detection would be of particular importance. Chapter 7.2 will go into further detail.

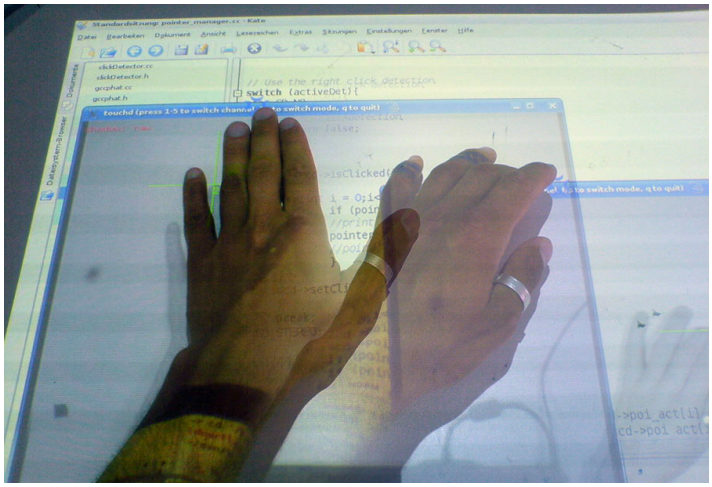


Figure 6.10: *Movement of a window with a gesture (This image is composed of two photos, begin and end of the movement)*

Summary

This chapter describes the implementation of the components as well as necessary accuracy tests of the system. As we had seen in the System Design chapter, data is transported in the layer model from bottom to top. TOUCHD, CALIBD and MOUSED communicate with UDP Packets. The format of these packets was described. Each packet is marked with the ID of the corresponding pixel blob which had been detected by TOUCHD. The MOUSED get calibrated coordinates from the calibration layer and assigns these values to system pointers. Incoming data is queued and the best matching values are chosen, either by blob size or by distance. Old data was cleaned up from the queue regularly.

MOUSED additionally implements the click detection. If No-Motion click mode is set, click events are triggered everytime, a pointer is moved, and then hold motionless on a location. Acoustic click detection is implemented, using two Piezo transducers. Coordinates have to be buffered, because of the pointer movement caused by the tap motion of the finger. Mono mode is only suitable for a single pointer. The stereo detection algorithm tries to differ several pointers. It uses the GCC-PHAT method to derive the correlation function of the two input signals. The time delay should be accessible from the maximum of this function. But it can not be found in a reliable way. So the pointer positions are used to limit the search range for the maximum search. This results in a better detection of tapping locations.

For all movements and click events, MOUSED generates X-Server events. The MPX X-Server manages physical and virtual devices. Two types of events are sent. Core events and XInput extension events. MPX generates both types and delivers them to corresponding applications. Legacy applications get the core events, whereas Multi-Pointer applications like the Multi Touch Puzzle can registrate on the XI events. This demo application is a puzzle game which can be played with multiple pointers and therefore also by two or more partitipants. To provide a toolkit for building these applications the FreeGlut Library is enhanced with this new possibilities.

Finally the accuracy of the presented acoustic tap detection was measured. The speed of sound in the table top was determined. Stereo detection with a simple maximum search proved unemployable. The distinction of pointers using the limited search method was more precise. Still improvement is neccessary.

Chapter 7

Conclusion

7.1 Conclusions

Figure 7.1: The finished terminal system

This thesis describes the design and development of a new input system. We use Computer Vision to build a Virtual Touchscreen Terminal, which allows Multi-Pointer interaction. Two technical decisions had been made. Combining a Virtual Touchscreen with a TAI has proved to be beneficial. On one hand, the one-camera Computer Vision System lacks a adequate method to detect touches, on the other TAI interfaces suffer from bad accuracy. Both methods themselves, are computatively efficient and can complement each other. The implemented TAI in this system works only with 2 audio channels (a standard stereo audio input) and is in this context a supplemental decision tool. It detects the user "clicks" and decides which of the VIT-positions had induced it. Implementing a Cross Correlation algorithm based on the GCC-PHAT, this work shows, that it is possible to make these decisions even in a limited setup. Though error rates are high and improvements on the TDOA-estimation have to be an essential part of future projects.

Another key idea in this thesis is the combination of our Multi-Pointer input system with a standard computer WIMP Interface. Sticking to reliable underlying methods, how input is processed inside the operating system, does not force the user to work with specialized software. Most applications, developed for mouse operation, are operable additionally with the new method. Browsing a website or selecting entries from a menu are

such tasks. Users are accommodated with this type of user interface and having a mouse cursor to make input. Therefore we developed a driver program to control multiple pointer cursors. The implementation of MPX as an X-Server allows application to be truly Multi-Pointer aware. Now simultaneous input is possible. We had presented the Multi-Touch Puzzle as a demo application implementing this functionality.

However the system is far from leaving the developer stage. Improving accuracy and error proneness is mandatory. But even now the new equipped SiViT system can give another perspective on the development of alternative input systems. The research area of HCI gets more and more into common interest, as far as computers is penetrating into more and more aspects of society. Though input methods have not that rapidly. Testing new methods can lead to improving usability of computer systems. Multi Touch and Multi-Pointer input are one of the most promising approaches. Humans are born with 10 fingers and two hands. Why not make use of them?

7.2 Future Work

The developed input system allows an adequate use of pointing gestures. Though we can find many system issues which may need improvement. Since the scope and time of this work is limited many idea are left to future research and engineering.

Most improvements is possible in the acoustic tap detection component. It works well for single pointer tapping but with more than two active pointers it is difficult to choose the right one for clicking. This is caused by the unstable time delay estimation. The resolution is not fine enough. More microphones and a suitable multi-channel audio interface permit full location estimation. Other TAI projects have shown that a much higher resolution (up to +/- 2 cm) is possible when using four or more microphones. Using pairwise estimation of TDOA and a least square estimation would improve higher precision. Considering the GCC algorithm itself, a better suited weighting function may replace the PHAT weighting. adjusted prefilters for the estimation in solid materials might be possible. An extensive frequency analysis of the occurring signals might show a way to gain a better noise suppression for the correlation. More precise measurements of sound speed and a more homogeneous table material might also contribute to better performance results. Thresholding for the tap detection could be replaced or improved by cross correlating with prerecorded tap samples to neglect false activations, caused by noise.

Other methods for the location determination e.g. LTM described in 3.3.1 eventually allow good results. Comparing them to the TDOA estimation will help to find out if better performance is possible. Concerning the optical tracking improvements will in the first place consider better behavior under unstable lightning conditions. On the hardware side this could result in a synchronization of camera and spotlights. The detected finger blobs are unstable which makes it harder to hit small targets, e.g. links or buttons. A better movement filtering has to be introduce in the TOUCHD. The Multi-Pointer X offers the possibility to dynamically generate new pointers mouse pointers. The assignment between mouse input devices and pointer objects can also be controlled dynamically. Until now a fixed number of pointer have to be installed, which allows only a fixed number of hands to work with the system.

To fully benefit from the Multi-Pointer abilities of MPX the employment of a Multi-Pointer aware windowmanager is mandatory. This could be the Multi Pointer Window Manager (MPWM)[1]. Here windows can be resized with twohanded gestures. Finally adjustments in the graphical environment can help to avoid operation errors. Larger symbols and menus may help hitting the correct target.

Further testing has to be done to evaluate issues like usability, user performance and user acceptance of a the virtual touchscreen terminal. New applications have to be built, for further demonstration, entertainment and testing purpose. Other touchscreen computer projects might show possible applications to rebuild and test for our system.

[1] http://cgit.freedesktop.org/ whot/mpwm/

Acknowledgments

This work has been made possible by the donation of the SiViT components. The support of all other utilized hardware materials by the chair had been another important factor. I also thank Peter Hutterer from the University of South Australia for his fast support with all questions and the fast fixing of bugs in the MPX Server.

I am grateful for the encouraged support of my supervisor Florian Echtler who initiated this project. I also thank all other people who gave me helpful comments and suggestions. Their support was gratefully acknowledged. Finally I like to gratefully mention the emotional encouragement and motivation I got from my girlfriend Stefanie Haubold.

Appendices

Class structures

MOUSED

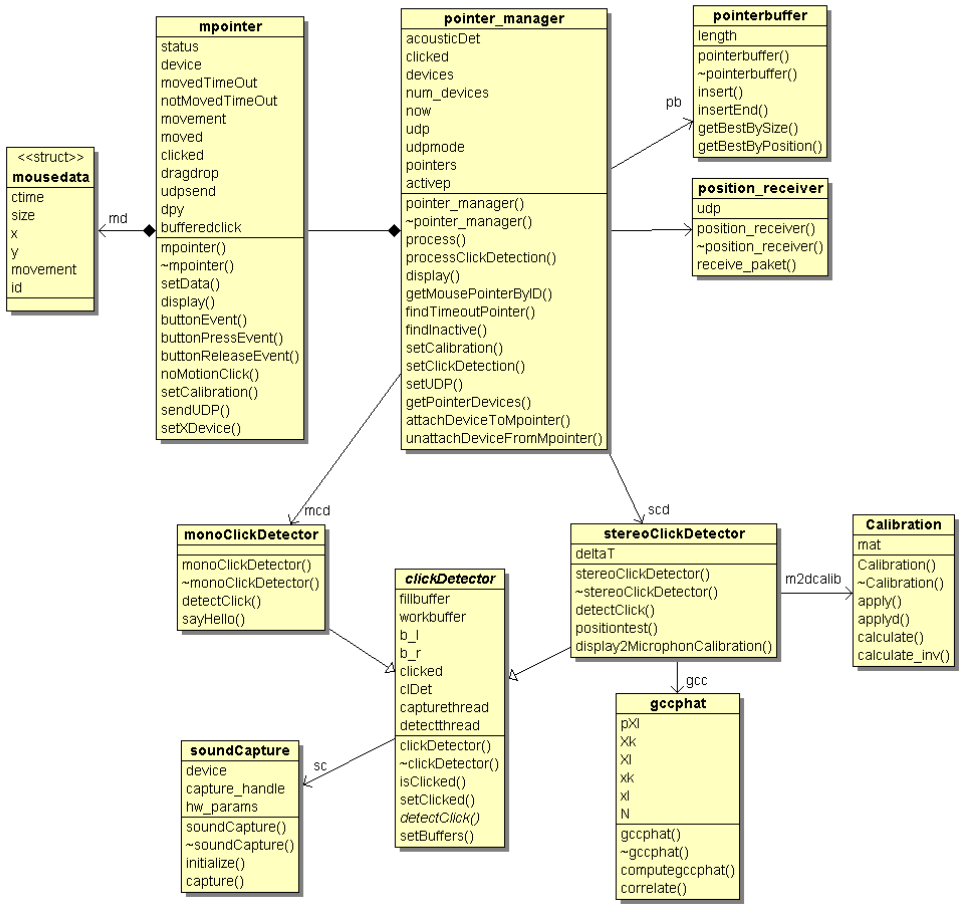


Figure A.1: *UML - class structure of the MOUSED*

Multi-Touch Puzzle

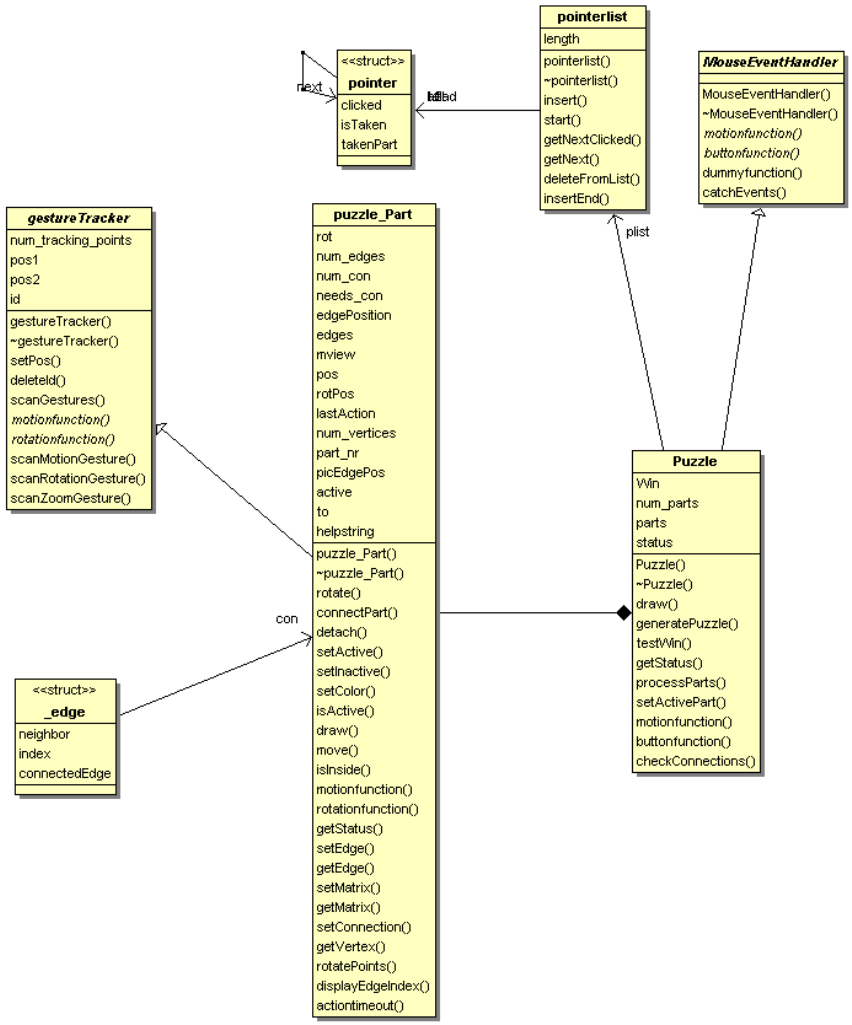


Figure A.2: *UML - class structure of the Multi-Touch Puzzle*

Hardware Components

Computer System	CPU:	Intel(R) Core(TM)2 CPU 6400 with 2.13GHz
	Memory:	2GB
	Graphics adapter	ATI Technologies Inc RV516 Radeon X1300/X1550 Series
	Capture Card	Philips Semiconductors SAA7134/SAA7135HL Video Broadcast Decoder
Camera	Pentax CV-M50 with IR-Filter	
Sound transducers	Harley Benton HB-T	

Cursor Theme "Circle Cursor"

This cursor theme is made by Russell Ambeault and published under GPL licence. It is installed in the test system.

Figure A.3: *Circle cursor theme*

xorg.conf Configuration to get multiple mouse devices in MPX

```
Section "InputDevice"
    Identifier "mouse0"
    Driver "mouse"
    Option "Device" "/dev/input/mouse0"
    Option "Protocol" "ImPS/2"
    Option "Emulate3Buttons" "on"
    Option "ZAxisMapping" "4 5"
EndSection
.
.
.
Section "InputDevice"
    Identifier "mouseX"
    Driver "mouse"
    Option "Device" "/dev/input/mouseX"
    Option "Protocol" "ImPS/2"
    Option "Emulate3Buttons" "on"
    Option "ZAxisMapping" "4 5"
EndSection
.
.
.
Section "ServerLayout"
  Identifier   "Layout[all]"
  Screen       "Screen[0]"

     InputDevice "corekbd" "CoreKeyboard"

     InputDevice "mouse0"
.
.
.
     InputDevice "mouseX"
EndSection
```

List of Abbreviations

AR	Augmented Reality
CV	Computer Vision
DOF	Degrees of Freedom
FFT	Fast Fourier Transformation
FTIR	Frustrated Total Internal Reflection
GCC	Generalized Cross Correlation
GCC-PHAT	GCC with Phase Transform
GUI	Graphical User Interface
HCI	Human Computer Interaction
IR	Infrared (light)
LTM	Location Template Matching
MPWM	Multi Pointer Window Manager
MPX	Multi-Pointer X Server
SCOT	Smoothed Coherence Transform
SiViT	Siemens Virtual Touchscreen
TAI	Tanglibe Acoustic Interface
TDOA	Time Difference of Arrival
VIT	Virtual Touchscreen
WIMP	Windows, Icons, Menus, Pointers

Bibliography

[1] Bill Buxton. Multi-touch systems that i have known and loved. http://www.billbuxton.com/multitouchOverview.html, 2007.

[2] Siemens. Virtual touch screen: A vision-based interactive surface. *User manual*, 1990.

[3] Bill Buxton. Human input to computer systems: Theories, techniques and technology, unpublished book manuscript. http://www.billbuxton.com/inputManuscript.html, 2008.

[4] Alan J. Dix, Janet Finlay, and Gregory D. Abowd. *Human-computer interaction*. Pearson Prentice-Hall, Harlow [u.a.], 3. ed. edition, 2004.

[5] William Buxton, Ralph Hill, and Peter Rowley. Issues and techniques in touch-sensitive tablet input. *SIGGRAPH Comput. Graph.*, 19(3):215–224, 1985.

[6] William Buxton. A three-state model of graphical input. In *INTERACT '90: Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction*, pages 449–456, Amsterdam, The Netherlands, The Netherlands, 1990. North-Holland Publishing Co.

[7] Trond Nilsen. Tankwar: Ar games at gencon indy 2005. In *ICAT '05: Proceedings of the 2005 international conference on Augmented tele-existence*, pages 243–244, New York, NY, USA, 2005. ACM.

[8] Mark S. Hancock, Sheelagh Carpendale, Frederic D. Vernier, Daniel Wigdor, and Chia Shen. Rotation and translation mechanisms for tabletop interaction. *tabletop*, 0:79–88, 2006.

[9] Microsoft. Microsoft surface computer. http://www.microsoft.com/surface/index.html, 2008.

[10] Pierre Wellner. Interacting with paper on the digitaldesk. *Commun. ACM*, 36(7):87–96, 1993.

[11] Jefferson Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 115–118, New York, NY, USA, 2005. ACM.

[12] Andrew D. Wilson. Playanywhere: a compact interactive tabletop projection-vision system. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 83–92, New York, NY, USA, 2005. ACM.

[13] Jun Rekimoto. Smartskin: an infrastructure for freehand manipulation on interactive surfaces. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 113–120, New York, NY, USA, 2002. ACM.

[14] Florian Echtler, Manuel Huber, and Gudrun Klinker. Shadow tracking on multi-touch tables. In *AVI '08: Proceedings of the working conference on Advanced Visual Interfaces*, pages 388–391, New York, NY, USA, 2008. ACM.

[15] Joseph A. Paradiso, Che King Leo, Nisha Checka, and Kaijen Hsiao. Passive acoustic knock tracking for interactive windows. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, pages 732–733, New York, NY, USA, 2002. ACM.

[16] Alex Frank. Dont click it website. http://www.dontclick.it, 2007.

[17] Alain Crevoisier and Pietro Polotti. Tangible acoustic interfaces and their applications for the design of new musical instruments. In *NIME '05: Proceedings of the 2005 conference on New interfaces for musical expression*, pages 97–100, Singapore, Singapore, 2004. National University of Singapore.

[18] http://www.i vibrations.com.

[19] Z. Ji D. T. Pham, Z. Wang. Acoustic pattern registration for a new type of human-computer interface. *IPROMs 2005 Virtual Conference*, May, 2005.

[20] Günther Schäfer Wolfgang Rolshofen, Peter Dietz. Neuartige berührbare schnittstellen durch die rückprojektion akustischer wellen.

Jahrestagung der Deutschen Gesellschaft für Akustik (DAGA2006), 2006.

[21] Düsing C. Wolfgang Rolshofen. Berührbare akustische benutzerschnittstellen. *IMW - Institutsmitteilung Nr 29*, pages pp. 63–66, 2004.

[22] H.F. Ying Yu Hoang Do Silverman. A real-time srp-phat source location implementation using stochastic region contraction(src) on a large-aperture microphone array. *Acoustics, Speech and Signal Processing ICASSP 2007. IEEE International Conference*, 1:I–121–I–124, 2007.

[23] G. Carter C. Knapp. The generalized correlation method for estimation of time delay. *IEEE Trans. Acoust. Speech Signal Process. 24 (4)*, pages 320–327, 1976.

[24] Lejun Xiao, Tim Collins, and Ying Sun. Acoustic source localization for human computer interaction. In *SPPRA'06: Proceedings of the 24th IASTED international conference on Signal processing, pattern recognition, and applications*, pages 9–14, Anaheim, CA, USA, 2006. ACTA Press.

[25] Peter Hutterer and Bruce H. Thomas. Bridging the gap between desktop computers and tabletop displays (poster). In *In Second International Workshop on Horizontal Interactive Human-Computer Systems(TableTop 2007)*, Newport, RI, US, Oktober 2007.

[26] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[27] Peter Hutterer and Bruce H. Thomas. Groupware support in the windowing system. In *In 8th Australasian User Interface Conference (AUIC2007)*, Balarat, Vic, Australia, Januar 2007. W. Piekarski and B. Plimmer, Eds.

[28] The freeglut Programming Consortium. The open-source opengl utility toolkit. http://freeglut.sourceforge.net, 2008.

[29] Robert W. Scheifler. X window system protocol, version 11. RFC 1013, 1987.

[30] George Sachs. X11 input extension porting document, x version 11, release 6.7. http://www.x.org/docs/Xi/port.pdf, 1991.