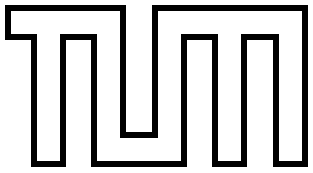


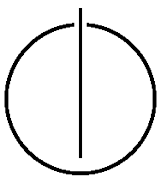
FAKULTÄT FÜR INFORMATIK

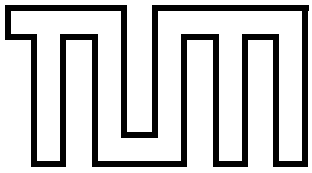
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit in Informatik

Multi-Touch Devices as Conventional Input Devices

Andreas Dippon





FAKULTÄT FÜR INFORMATIK

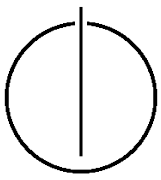
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit in Informatik

Multi-Touch Devices as Conventional Input Devices

Multi-Touch Geräte als herkömmliche Eingabegeräte

Author:	Andreas Dippon
Supervisor:	Prof. Gudrun Klinker, Ph.D.
Advisor:	Dr. Florian Echtler
Date:	September 24, 2010



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.

München, den 24.September 2010 Andreas Dippon

Acknowledgments

I want to thank Professor Gudrun Klinker, Ph.D. for the opportunity to work in such an interesting area of computer science.

Special thanks to Dr. Florian Echtler for being such an inspiring person and the great support of new ideas.

Further gratitude goes to the whole staff of the FAR group, who created an especially good working atmosphere, which made writing such a thesis much more enjoyable. Especially I want to thank the following people for their advice during this thesis: Simon Nestler, Eva Artinger, Tayfur Coskun, Marcus Tönnis, Manuel Huber and Patrick Maier.

Additional thanks to all participants of the evaluation.

Last but not least I want to thank my brother Günther Dippon for some troublesome reading and discussion sessions, and also my parents for their everlasting support during my whole studies.

Abstract

In consideration of the rapid development of displays and multi-touch technologies, many workspaces could feature integrated multi-touch displays in the near future. In order to improve the functionality of such devices, the possibility of using them as input devices for other computers needs to be reviewed. The idea is, to get rid of many different input devices (e.g. keyboard, mouse, multi-touch pad) by using a single multi-touch display. Furthermore the display can be used as an additional monitor to show e.g. toolbars, which can be directly manipulated through multi-touch gestures.

During this thesis, an implementation of this idea was realized by using a selfmade large scale multi-touch table as input device and a windows-based notebook as the main computer. The program provides an adaptive keyboard and a multi-touch pad as well as the option to drag&drop standard Windows widgets onto the multi-touch table, which can be controlled by direct touch input. A small user study was conducted to test the current system and to get information about the further approach to this concept.

Contents

List of Figures

Chapter 1

Introduction

"This is gonna change the way we interact with machines from this point on."

Jeff Han, TED Talk, Feb 2006

Talking about low-cost, scalable, high resolution multi-touch surfaces at the TED Conference 2006 [10], Jeff Han opened up the door to a largely undiscovered research area. Afterwards, multi-touch began to spread rapidly in science, and later also in industry and entertainment areas. Four years later, as small multi-touch devices, especially mobile phones like the IPhone, are already well integrated in modern society, big multi-touch screens still have a long way to go. As mobile phones were already widely spread and multi-touch displays made them more convenient to control, the idea of multi-touch on such devices became very popular. While the integration of small multi-touch displays into daily life was therefore quite simple, big screens are currently rarely used outside of research laboratories. One problem is, that the usability of multi-touch sensors in big standard screens, like TVs or computer monitors, is still lacking useful applications. Rather than directly controlling the device by touching the monitor, it is more convenient to control a TV via remote control from the sofa and a computer via mouse and keyboard at the desk. Thus, big multi-touch screens will probably become additional or independent devices, which will be used for certain specific applications. There is much research going on, about where to use such devices. For example, in the "SpeedUp" Project[8], a big multi-touch table is used to keep track of patients and to coordinate rescue teams in a mass casualty incident, instead of a normal map. Another approach is to use a multi-touch table to explore data and interact with the computer with multiple users and without additional devices, such as a mouse or a keyboard.

For example, the Microsoft Surface[12] is used in some hotels and bars as a public display, where people can gather information or order drinks.

The idea of this thesis is, to bring big multi-touch screens into daily office life in the future. Instead of a normal desk, a multi-touch screen is used. The screen is used as an input device for a normal desktop computer and replaces all ancient input devices. A keyboard can be shown, as well as a multi-touch pad, a drawing table, a piano, a mixing desk and so on. At the same time it can be used as an additional monitor, so that programs and windows can be dragged onto the screen and be controlled with direct touch input. Several advantages are depicted in the following list:

- *changeability*: you can simply change the shown virtual input devices, without having to rearrange your workspace. Therefore you can switch easily between different tasks, always using the appropriate input device.
- *adaptivity*: the virtual input devices can adapt to the current language, program, etc. For example, the virtual keyboard can show the correct keyboard layout for different languages, or provide the user with additional information (e.g. when pressing the ctrl key in a program, the shortcuts of functions are shown on the keys).
- *scalability*: the virtual input devices can be rescaled to meet each user's demands.

As good as this may sound, there are currently still several drawbacks, which are described in the following itemization:

- *haptic feedback*: currently, multi-touch screens don't provide any haptic feedback, which for example makes it harder to write on an on-screen keyboard without looking at it, than on a physical keyboard.
- *size and costs*: big, flat multi-touch screens are still very expensive and therefore not applicable as standard office desks.

There are several ongoing research projects, to get rid of these disadvantages. For example, the *Mudpad*[27] provides haptic feedback via a ferromagnetic fluid, which can be locally stiffened, so that the user can feel it on the surface. This and other projects are described in detail in chapter 2: Related Work.

During this thesis, a basic implementation of the concept was built, providing the following features. The program is capable of showing a keyboard

with different layouts for different languages and on pressing special keys in certain programs. Additionally a multi-touch pad was implemented. A Windows notebook running this program on a secondary screen (which is capable of multi-touch sensing), can be completely controlled by the shown touch pad and keyboard, as well as direct touch input can be used.

After the implementation of the program, a user study was conducted to test the system. As a large scale user study with long testing periods and reiteration would be required to adequately test the whole concept, we decided to do a small scale user study to test the features of the keyboard, in particular the *adaptivity*.

As already mentioned, the following chapter describes some related work, mainly concerning the elimination of the disadvantages. Afterwards the details of the implementation are depicted in chapter 3. In chapter 4 the executed user study is desribed in-depth. Some suggestions for future work can be found in chapter 5. A conclusion and a short review of our work is given in chapter 6.

Chapter 2

Related Work

In this chapter, some of the related work and research is described. The chapter is divided into three different sections. In the first section, research projects about the hardware for big multi-touch screens are mentioned. Afterwards, work about the improvement of the hardware in the future is detailed in the subsequent section. The last section details some related work on similar concepts, as in this thesis, but with different approaches.

2.1 Big Multi-Touch Screens

This section describes the hardware of a big (about 40 inch) multi-touch screen, which is used in this thesis.

Frustrated Total Internal Reflection (FTIR)

While this technique was already introduced in the context of fingerprint sensing in 1965[23], it became very popular for big multi-touch screens after Jeff Han's work in 2005[9]. The basic principle of this technique uses infrared light, which is reflected within a see-through material (e.g. acrylic glass) via total internal reflection. When the surface area is touched, the light will be frustrated at this point and will be reflected and refracted through the finger out of the surface. This effect can be registered by an infrared camera and processed in a computer (see figure 2.1).

Because a device using this technique can be built at very low cost and at any size, this is currently the most popular technique in big multi-touch devices. The standard way of building such a device and much more information on this subject can be found in the technical report of Schöning et al.[16]. As the devices described there all require the space below the dis-

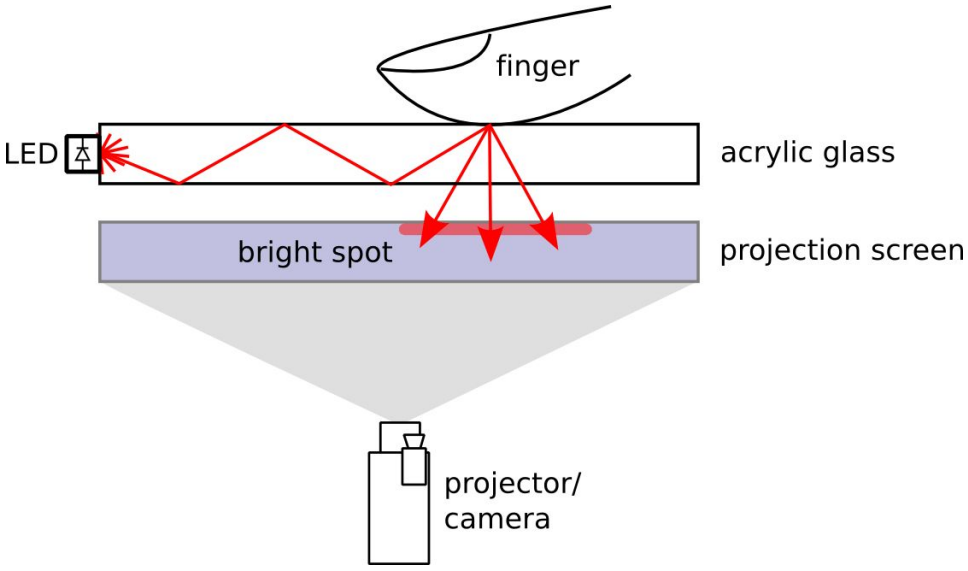


Figure 2.1: FTIR multi-touch principle. Image taken from [6]

play for the projector and camera, they are not applicable in an office desk. An improvement, in terms of required space, was developed by Echtler[7]. In this system an LCD screen is used and the refracted light of the touch point through the finger is captured by an infrared camera above the screen (see figure 2.2).

2.2 Improvements

All currently available big multi-touch devices share one drawback: the lack of tactile feedback. There are solutions for small devices via vibrotactile feedback. One is the Artex project by Crossan et al.[5]. This approach provides different virtual textures for different areas of the display of a mobile phone, e.g. buttons. The textures are created by vibrating the device using a vibration motor attached to the back of a mobile phone. Another solution is the Tactile Pattern Display(TPaD) by Winfield et al.[26], which provides haptic feedback by modifying the friction parameter of the display according to the position of the finger. The small prototype was improved and introduced as the Large Area TPaD(LATPaD)[11]. This version can already be used in a big screen, but it is still only related to one finger instead of multi-touch, and the friction can only be set in certain patterns. Both of these solutions are quite promising, but they currently only give global resonance instead of local feedback. Another solution are malleable

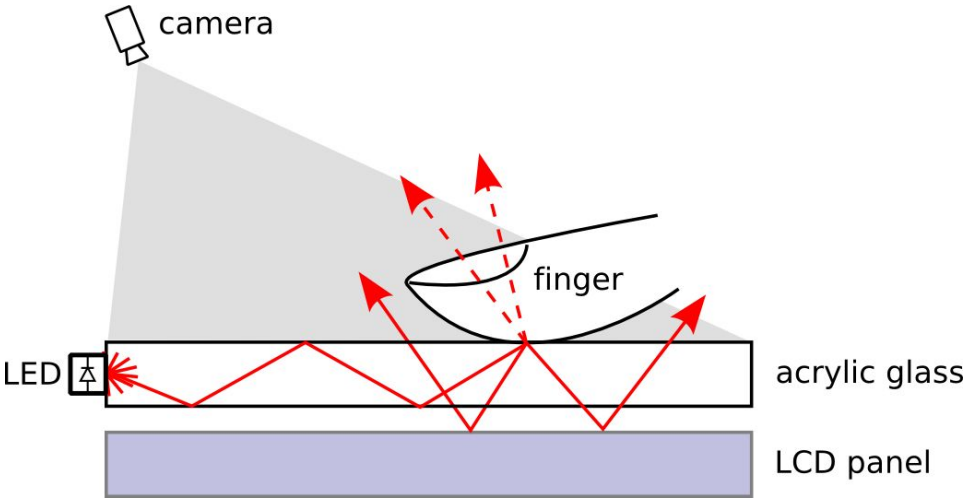


Figure 2.2: Inverted FTIR[7]

surfaces using fluids below the surface. Normally these displays offer passive haptic feedback, like the project presented by Graham et al.[19], and the pressure of a touch can be measured. In order to feel certain keys or a whole keyboard on a display, active haptic feedback is required. The idea of a malleable display with active haptic feedback was therefore presented in the following research project.

Mudpad

The touch surface of the Mudpad by Jansen[27] is a malleable pouch filled with a smart fluid. This fluid is magneto-rheological, whose viscosity can be changed by applying magnetic fields. The fluid contains ferrous particles which build chains along the magnetic flux lines, when exposed to a magnetic field (see figure 2.3). Therefore, an array of electromagnets is installed under the pouch, to dynamically change the viscosity of the fluid (this approach is based on the Actuated Workbench by Pangaro et al.[15]). The resolution of different viscosities within the fluid depends only on the resolution of the magnets under the surface. By using an electro-rheological fluid instead of an magneto-rheological one, the reaction time of different states is also only dependent on the magnets. This system has several advantages. First, the different viscosities are invisible and therefore don't distort the image. Nevertheless, they can be easily felt by the user when sliding across the surface. Second, the fluid stays in the designated form, because the status only changes when the magnets are turned on or off.

Third, this approach provides local feedback, which is very important for multi-touch or multi-user devices, such as big multi-touch screens.

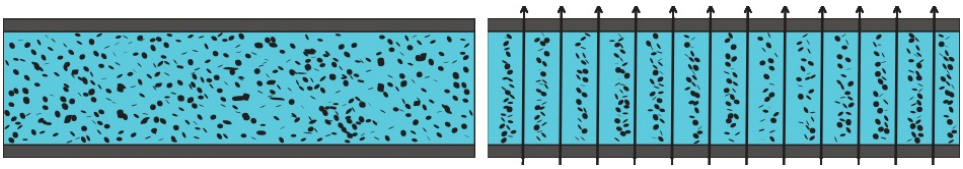


Figure 2.3: The left image shows the ”Off-state”, where the particles flow freely in the fluid. The right image shows the ”On-state”, where the particles are arranged along the flux lines. Images taken from [27].

2.3 Similar Research

This section details related work about bringing displays and multi-touch to input devices instead of the other way round.

Touch-Display Keyboards

One approach is augmenting keyboards with touch sensitivity and small displays in each key. A first commercial product with small displays in each key was presented with the Optimus Maximus Keyboard by Art Lebedev Studio[20]. Therefore, the keyboard layout can be changed through software and can show e.g. different languages or program related icons. The TDK(Touch-Display Keyboard) project by Block et al.[3] improves the concept, by adding touch sensors to each key and using the keyboard as an additional display for the computer. This provides several new features as shown in the following list from *Touch-Display Keyboards*[3]:

- The matrix of key-displays is conceived as a coherent display surface that can extend the primary display in a user interface.
- Graphical elements can be distributed between and moved across keyboard and primary display.
- Mouse interaction is extended across the keyboard display.
- Touch-sensing adds an additional layer and state of input on the keyboard.

These features are similar to the ones, provided by our solution with a multi-touch screen. But this approach comes with several benefits compared to our solution, but also with some other drawbacks. The most important aspect is the maintenance of the tactile feedback, while receiving the gain of an additional display for the computer, as well as the possibility of showing different key layouts on the keyboard. The drawbacks hereby are the fixed size and configuration of the keyboard and also the discontinuity of the surface, because the keys are uneven and there are gaps between them(see figure 2.4).

Figure 2.4: The discontinous surface and the gaps distort the picture. Image taken from [3]

The touch sensors, which are built into each key, provide another feature: touch input on the keys. As it is mentioned in [21], it is a significant ergonomic advantage, that the user can rest his fingers on the keys without triggering input. This resting can now be used for additional input features. For example, when the fingers rest on the keys, the covered areas could be blend in above the fingers to avoid complete occlusion (see figure 2.5). While this can only be used in a very limited way in our hardware (increased size of finger blobs, when applying more pressure), a malleable multi-touch device like the Mudpad, which is pressure sensitive could also implement this feature.

A quite similar project is the Adaptive Keyboard from Microsoft, which will be used for the UIST 2010 Student Innovation Contest[13]. Hereby, the touch sensitivity of the keys is removed, but an additional small multi-touch area is added on top of the keyboard. The keys are transparent and

the screen which is used for the multi-touch area continues under the keys, so it can be used to display different layouts on the keyboard.

Figure 2.5: Finger-occlusion preview: graphical controls occluded by fingers are shown above the covered key.[3]

Curve

Another related project, which deals with the idea of combining a virtual desktop with a real one, is the Curve Project by Wimmer et al.[25]. They constructed a prototype for an interactive desktop, based on the concept of the DigitalDesk by Wellner[22]. The *Curve* blends a horizontal and a vertical interactive surface, which takes existing ergonomics research into account (see figure 2.6).

Figure 2.6: The prototype of Curve. Image taken from [25]

The focus of their work lies in visual ergonomics (design of a surface to ease reading and watching visual content) and touch ergonomics (parameters influencing direct-touch input on interactive surfaces). Referring to several other research work[1, 2, 14, 18], they found, that a digital desk

should offer a more or less horizontal and a more or less vertical interactive surface. E.g. another research group around Morris[14] pointed out, that users prefer a vertical surface for writing using a keyboard, whereas it is strongly disliked for other tasks. According to these findings, it seems to be worth investigating the integration of a horizontal interactive surface into daily office work. If the curved shape can help with this integration, will be tested by the group in future work.

Summary

A review of the related work which was presented in this chapter shows, that there is already a lot of research going on towards the integration of horizontal multi-touch displays or keyboards into standard desktop environments. While the hardware still needs some improvements, first studies on the usability of such systems can be conducted.

Chapter 3

Implementation

In order to test the introduced concept, a prototype was implemented during this thesis. The developed program was written in C++ using Microsoft Visual Studio 2008. For the multi-touch support we used the libTISCH library by Florian Echtler[1]. The prototype features an adaptive keyboard, a multi-touch pad and an area which can be used for direct touch input (see figure 3.1). Those features will be depicted in the following sections. An overview and further details of the implementation can be found in appendix A.

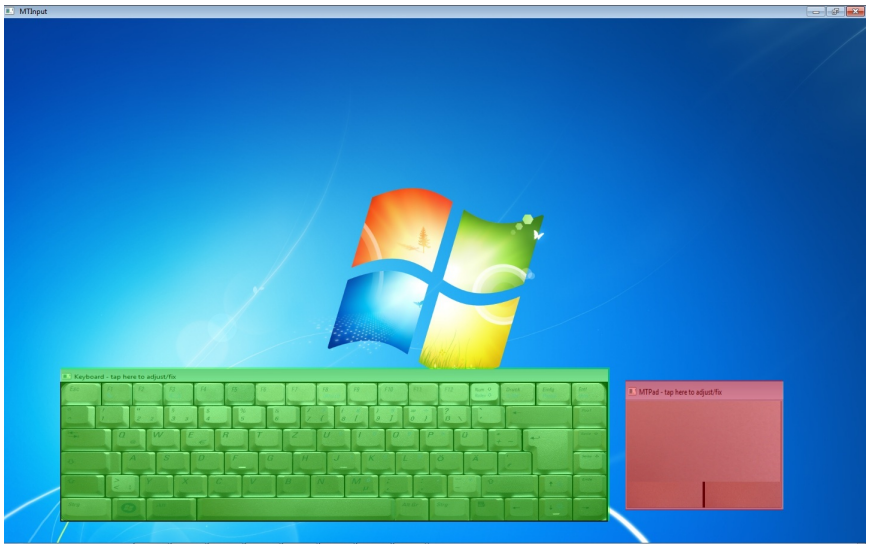


Figure 3.1: The prototype implementation of the concept: MTPad (red), Keyboard (green) and direct touch area (rest).

[1] http://tisch.sourceforge.net/

3.1 Direct Touch

The concept of direct touch input is, that users can drag windows or toolbars onto the multi-touch screen and use their fingers instead of a mouse or a touchpad. For example, the user could navigate through the file system or change colors in a graphics editor by directly touching the according buttons. In order to support this concept, a few issues need to be considered. First, an area for multi-touch input needs to be specified. Then the corresponding *Windows* coordinates of the touch points need to be calculated. Afterwards, a mouse event needs to be invoked at this location.

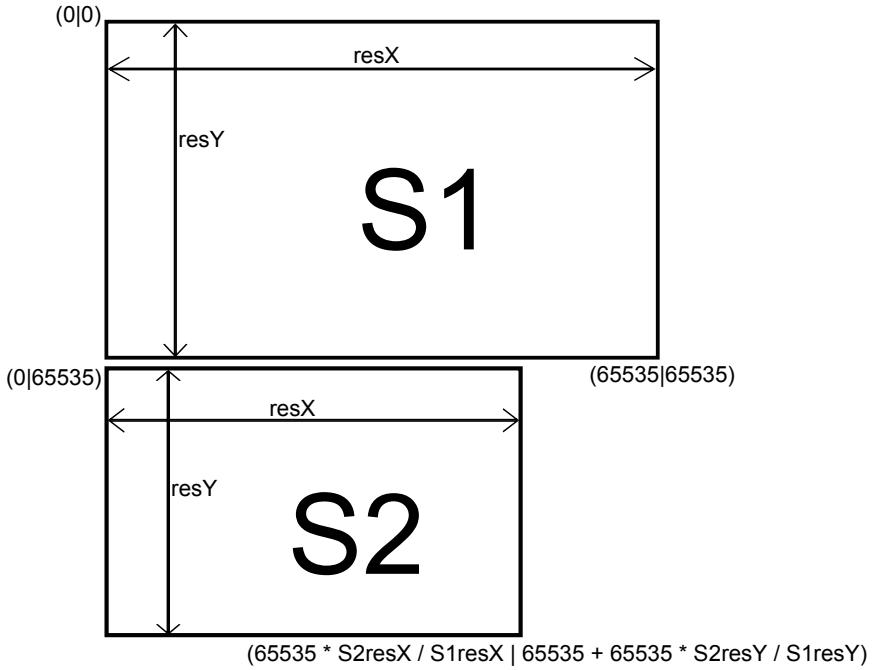
The first issue can be done quite easily with the libTISCH library. Therefore, a background container was implemented, which is based on the *Container* class of the library. The size of this container equals the size of the program window. Thus, all touch events, which are received by this container are already in the local coordinate system of the program window. In order to get the corresponding *Windows* coordinates, several properties have to be taken into account: the position and size of the program window, as well as the alignment of the screens and their resolutions. As the size of the libTISCH program is fixed to the size of the second screen, and the center of the program window is also fixed to the center of this screen, these two properties are implicitly taken care of. In order to calculate the correct coordinates with respect to the other properties, we need to look at the specifications of the different coordinate systems. For visual support, the coordinate systems are also shown in figure 3.2.

The local coordinate system within the libTISCH program is specified as follows:

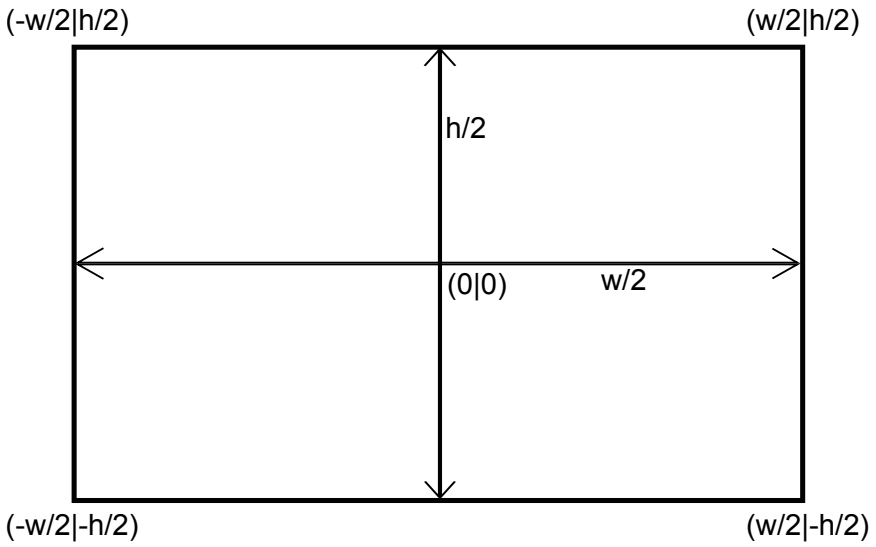
- (0, 0) – center of the program window
- (width/2, height/2) – upper right corner of the program window, whereas width/height resemble the width/height of the program window
- (-widht/2, -height/2) – lower left corner of the program window

The specification of the *Windows* coordinate system is explained in the following itemization:

- (0, 0) – upper left corner of the primary monitor
- (65535, 65535) – lower right corner of the primary monitor



(a) Windows coordinate system (S1/S2: screen 1/2, resX/Y: resolution)



(b) libTISCH coordinate system

Figure 3.2: coordinate systems

The coordinates are mapped to the display surface according to its resolution. In a multimonitor system, the coordinates on the second screen are determined according to the alignment of the screens and the ratio between their resolutions.

In our multimonitor system, the screens are always aligned on top of each other, with the primary monitor being the topmost. With the previously mentioned simplification of the calculation, because of the fixed position and size of the program window, we get the following equations for the calculation of the *Windows* coordinates:

$$x.coord = \frac{touch.x + 0.5 * sm.width}{sm.width} * 65535 * \frac{sm.width}{pm.width} \tag{3.1}$$

$$y.coord = \frac{0.5 * sm.height - touch.y}{sm.height} * 65535 * \frac{sm.height}{pm.height} + 65535 \tag{3.2}$$

The details of these equations are depicted in the following. The first part of equation (3.1) $\frac{touch.x+0.5*sm.width}{sm.width}$ calculates the normalized x-vector of the touch point on the screen. *touch.x* resembles the x-coordinate of the touch point in the coordinate system of the program. *sm.width* equals the width of the secondary monitor. The next step is to get the corresponding *Windows* coordinate of the normalized vector. This is done by multiplying the result by 65535. As the relation of the mapping to the *Windows* coordinates and the normalized vector is defined by the resolution of the primary monitor, different resolutions of other screens need to be taken into account. This is done by $\frac{sm.width}{pm.width}$, whereas *pm.width* stands for the width of the primary monitor.
The equation for the y-coordinate (3.2) is quite similar, but with a few minor adjustments. As the y-coordinate system and the y-coordinates of *Windows* point in opposite directions, the first part needs to be slightly modified: $\frac{0.5*sm.height-touch.y}{sm.height}$. The subsequent part is equal to equation (3.1): $*65535 * \frac{sm.height}{pm.height}$. At the end of the calculation, the displacement to the second screen has to be done by adding 65535 which resembles the lower boundary of the primary monitor.

As the coordinate problem is solved, the next step is to invoke a mouse event at the calculated position. In the prototype, the direct touch area only supports single touch input, because the standard *Windows* environment only supports one mouse pointer. The type of mouse event depends on the action of the user. For the different events, in this case tap, release and move, the libTISCH library provides us with abstract functions, which are automatically called for the corresponding events.

When a new touch id is recognized within the direct touch area, a press of the left mouse button should be triggered at the corresponding location. This is done by moving the mouse cursor to this location, which is realized by a call to the *MouseMoveABS* function (see section 3.4), which defines the absolute position of the mouse cursor. Then the *MouseDown* function is called with parameter *0* to simulate pressing the left mouse button. When a touch is released, a call to the *MouseUp* function with parameter *0* is performed, which resembles the release of the previously pressed left mouse button and therefore the left mouse click is completed.

The movement of a touch point on the direct touch area resembles a click and drag gesture. The previously definition of the mouse click simplifies the implementation of this gesture, as only the move part needs to be programmed. This is done by a call to the *MouseMove* function, which simulates a relative movement of the mouse cursor.

3.2 MTPad

The implementation of the multi-touch pad (or MTPad) in our prototype is capable of moving the mouse cursor, left and right click and scrolling by using multiple fingers. For convenience, the MTPad looks like a standard touchpad of a notebook (see figure 3.3).

The mouse movement is simulated by a call to the *MouseMove* function. The passed movement vector is adjusted by a factor, which is dependent on the move speed of the finger. The speed is defined by the following equation: $vec.x = vec.x * \min(8, \max(3, |vec.x|))$. Due to this equation, the speed of the cursor is always between 3 and 8 times faster than the finger movement on the MTPad.
The MTPad also takes advantage of the large area of the multi-touch display. Other than on a standard touchpad, the user doesn't have to move the cursor in several small steps, because he can start on the touchpad and just keep moving his finger on the multi-touch display till the cursor has reached the desired position. This effect doesn't interfere with the direct touch functionality, because each tracked touch has a unique id, which is always related only to the area, where it was sensed first.
Left and right clicks can be done by touching the buttons below the MTPad. A touch on these buttons calls the function *MouseDown* with the according parameter (*0* for left click, *1* for right click). On release, the *MouseUp* function is called with the corresponding parameter. Another possibility of in-

voking a left click, is to only tap the MTPad for a short moment (less than 50 ms).
We also implemented a multi-touch gesture for the MTPad: scrolling. In order to scroll, the user has to use at least two fingers at the same time on the MTPad. By moving those fingers up and down, the *MouseScroll* function is called with the movement vector as a parameter.

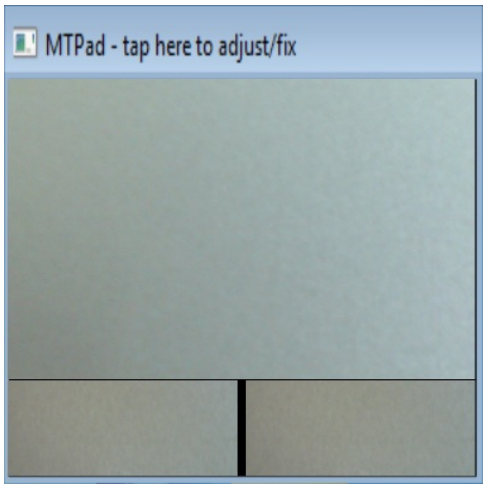


Figure 3.3: MTPad

3.3 Keyboard

The most important aspect of our concept is the traditional keyboard. To keep the adaption level of the user low, we decided to use the shape of a conventional keyboard, instead of testing new designs. The implemented keyboard supports different key layouts (see figure 3.4) and program specific shortcut icons can be shown for several programs: *Inkscape*[2], *Windows Calculator* and *Windows Notepad*.
The *Keyboard* class is derived from the *Container* class of the libTISCH library. The keys are objects of the class *Key*, which is derived from the libTISCH class *Button*. During the initialization of the keyboard, all keys are added and positioned within the *Keyboard* container. Each *Key* object has a parameter *type*, which determines the functionality of the key. All keytypes are stored as hex values in arrays of the *Keyboard* object. The hex values match the *Virtual-Key Codes*[3] and therefore they can directly be used

[2]http://inkscape.org/
[3]http://msdn.microsoft.com/en-us/library/dd375731%28v=VS.85%29.aspx

(a) english layout

(b) german layout

Figure 3.4: different language layouts

to trigger key events later on. Additionally, the key textures were stored in files according to their hex number, so the *type* can also be used to get the right texture for each key. For all keys with different textures in the german and english layout, a fixed value of 256 (or 0x100) is added to the value of the *type* parameter of the german key, so that the correct texture is used, while the *type* variable can still be easily used for the functionality. As different language layouts can be set for different windows, and some programs have special shortcut icons on the keys, a check for an update of the keyboard layout is required, whenever a key is touched, the MTPad invoked a left click or a direct touch event was recognized. This is done by the *update_keytextures* function, which also takes into account, if special keys like ctrl, alt or shift are pressed. This is especially important for the programs which are supported by the shortcut icons on the keyboard, because the layout of these icons changes depending on the pressed special keys (see figure 3.5).

(a) no special key is pressed

(b) ctrl key is pressed

(c) ctrl and shift keys are pressed

Figure 3.5: The layout of the icons in Inkscape with different special keys pressed.

3.4 Functions

The main functions of the program are described in this section. All keyboard and mouse inputs are triggered by using the *SendInput* function[4].Further details of the implementation can be found in appendix A.

MouseMove and MouseMoveABS

These functions need to be called with a parameter *vec*. In the first case, this parameter equals the movement vector of a relative mouse movement. In the second case, it stands for the absolute position, where the cursor should be moved to. As this absolute position is still in the coordinate system of the program window, the calculation described in section 3.1 needs to be carried out to get the absolute position in *Windows* coordinates. The *INPUT* object for the *SendInput* method is quite easy to use for these two functions. The parameter *vec* is written to the *mi.dx* and *mi.dy* variables, in which *mi* stands for *mouseinput* and *dx/dy* contain the absolute position of the mouse, or the amount of motion since the last mouse event was generated, depending on the value of the *dwFlags* member[5]. Therefore the *dwFlags* variable is set to *MOUSEEVENTF_MOVE*, and to *MOUSEEVENTF_MOVE | MOUSEEVENTF_ABSOLUTE* respectively. Finally, the *type* of the *INPUT* object is set to *INPUT_MOUSE*.

MouseScroll

The *MouseScroll* function is quite similar to the move functions. From the passed *vec* parameter, only the y-coordinate is used. The *dwFlags* have to be set to *MOUSEEVENTF_WHEEL* in this case. The *type* is again set to *INPUT_MOUSE*.

MouseDown and MouseUp

These functions require a parameter *type* instead of a vector. The value of *type* defines, which mouse button will be invoked. Thereby the value *0* equals the left mouse button and *1* equals the right mouse button. According to this value and the called function, the *dwFlags* parameter is set to *MOUSEEVENTF_LEFTDOWN*, *MOUSEEVENTF_RIGHTDOWN*,

[4]http://msdn.microsoft.com/en-us/library/ms646310%28VS.85%29.aspx
[5]http://msdn.microsoft.com/en-us/library/ms646273%28v=VS.85%29.aspx

MOUSEEVENTF_LEFTUP or *MOUSEEVENTF_RIGHTUP*. After the *SendInput* function, a function named *always_on_bottom* is called, which asures, that the program window never covers any other window.

Keyboard Input

The input of the keys is directly coded within the *tap* and the *release* function of each key. The key inputs can be simulated similar to the mouse inputs by using the *SendInput* function. The *type* parameter of the *INPUT* object has to be set to *INPUT_KEYBOARD* in this case. The type of the key can be set via the *ki.wVK* parameter, in which *ki* stands for *keyboardinput* and *wVK* represents a virtual-key code. As each key object has its correct virtual-key code stored in its *type* parameter, the stored value can be directly used as the value of *ki.wVK* (the value is calculated with an additional modulo calculation, to take account of the different language types, see section 3.3). The *tap* function simulates a key press, and the *release* function a key release. This is done by setting the *dwFlags* parameter of the *ki* object. For a key press, the default value can be used, and for the simulation of a key release, the value is set to *KEYEVENTF_UP*.

Summary

The implementation of the prototype was depicted in this chapter. Deeper insight into the implementation can be found in appendix A. Using this prototype application, a small scale user study was conducted, which will be described in the following chapter.

Chapter 4

Evaluation

In order to get a first impression of how people react to the introduced concept, a small scale user study was conducted. First, the goals and setup of this user study will be described in this chapter. Afterwards, the exercises of the test are depicted. The chapter is concluded by the outcome of the study.

4.1 Goals

As already mentioned, we wanted to get some information about the acceptance of the system. Our first approach was to do a study of the whole concept. Although the outcome of such a test would be very valuable, there are several problems conducting the test. In order to test the concept for everyday work, the hardware would have to be integrated in a normal desk, so the proband could sit and work there normally. Furthermore, like in the test conducted by Wigdor et al. about *Living with a tabletop*[24], the test subject would have to use the system for a long time, to get used to it and in order to get significant test results. Therefore we decided to do a shorter test about the implemented keyboard functions. The goal was to get information about the utility of the adaptivity of the keyboard.

4.2 Evaluation Setup

The tests took place in a showroom of the faculty of computer science at the Technische Universität München, which is called *ITüpferl*[1]. For the horizontal monitor, we used a big FTIR multi-touch table, called TISCH (Tangible

[1] http://www.in.tum.de/fuer-studierende-der-tum/ituepferl.html

Interactive Surfaces for Collaboration between Humans), which was built by Florian Echtler during his dissertation[6]. In order to match the size of the horizontal plane, a 42 inch television screen was used. The exercises and the protoype were running on a *Windows* notebook, which used the TV as the primary monitor. While the computer of the multi-touch table was used for the touch sensing, its projector was used as the secondary monitor of the notebook. The complete test setup is shown in figure 4.1.

4.3 User Study

The execution of the user study is depicted in this section. First, the probands had to do certain predefined tasks, which are described in the following subsection. Afterwards, they had to fill out two questionaires, that are delineated in the subsequent subsection. As last part of the study, the test subjects participated in a short interview about the whole concept. The description of this interview is given in the last subsection. The analysis and the outcome of the user study will be detailed in section 4.4.

4.3.1 Exercises

The main purpose of the exercises was to introduce the users to the concept and to test some of its functions. Even though the focus was on the keyboard, which was evaluated with the second task, the first task was designed to get in touch with the system. Both tasks were done in *Inkscape*[2].

Task 1: Direct-Touch Input

In the first exercise, several colored rectangles with different colored outlines were shown (see appendix B.1) in the *Inkscape* main window on the vertical display. The adaptive keyboard, the MTPad and the *Filling and Outline* toolbox of *Inkscape* (see figure 4.2) were displayed on the horizontal screen. The probands had to select the left rectangles by using the MTPad. When a rectangle was selected, they had to change the color of the outline and the filling by using direct touch on the toolbox. The goal was to match the colors of the rectangle with the colors of the rectangle next to the selected one.

[2] http://inkscape.org/

Figure 4.1: setup for the user study

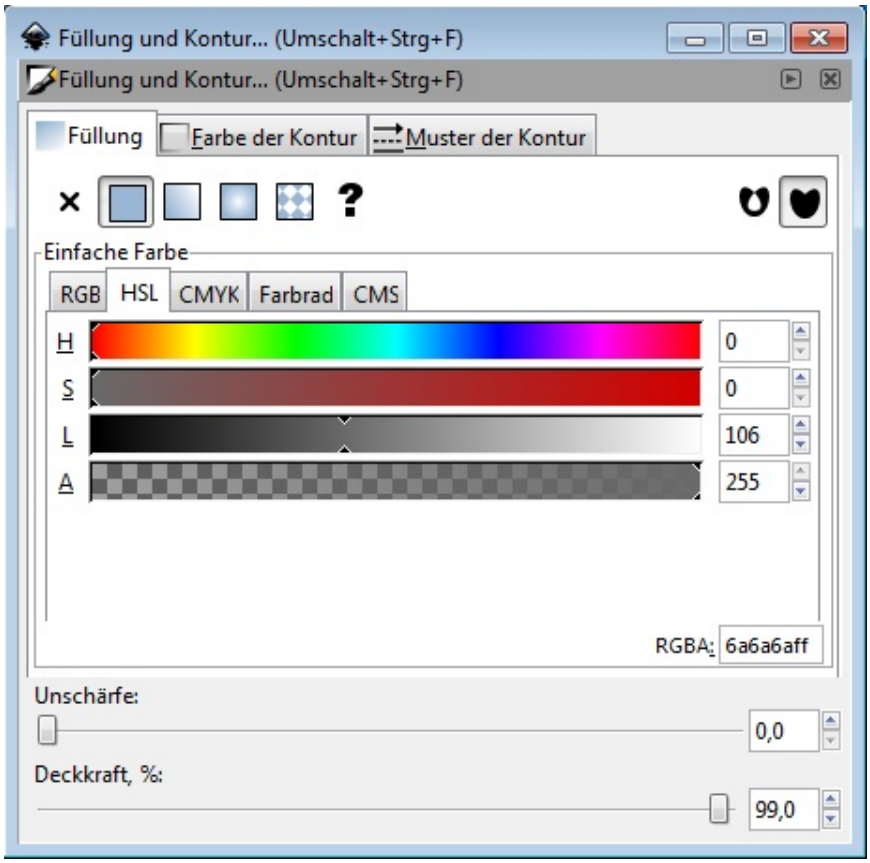


Figure 4.2: Filling and Outline Toolbox of Inkscape

Task 2: Adaptive Keyboard

The second exercise consisted of two columns, where a few combinations of objects were shown in each row (see appendix B.2). The exercise was divided into three subtasks. In each subtask, the probands had to combine the objects on the left by using the given commands to get the same objects as in the right column. Therefore, they had to use the keyboard shortcuts of the commands, which were shown on the keyboard in form of icons (see figure 3.5 on page 20). To make the icons easier to remember, they were also shown between the objects.

In the first subtask, the probands had to use one of the following five commands to combine the two objects on the left: union, difference, intersection, exclusion and division (see figure 4.3). The shown objects looked similar to the icon of the command, which had to be used. Therefore, the user

can easily learn the functionality of each command.

Figure 4.3: Inkscape commands: union, difference, intersection, exclusion and division

In the second subtask, four commands were added to the previous five, so that the user had to use a combination of at least two out of nine commands to correctly combine the shown objects in this part of the exercise. The four new commands were: raise selection, lower selection, flip horizontal, flip vertical (see figure 4.4). In order to increase the learning effect of the first commands, the objects still looked quite similar to the icons of the commands.

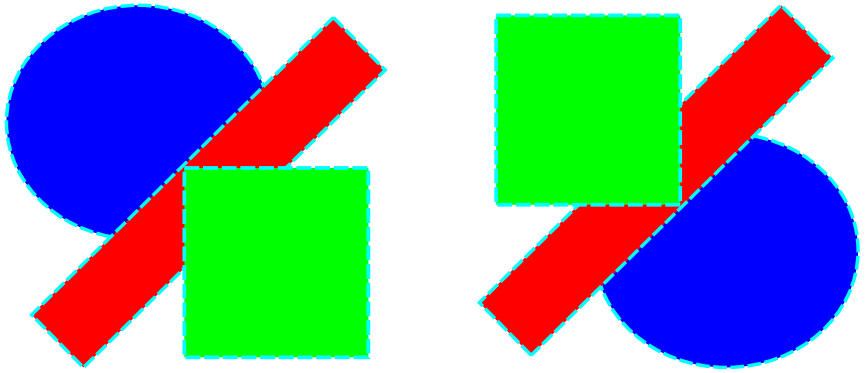
Figure 4.4: additional commands: raise selection, lower selection, flip horizontal and flip vertical

In the third subtask, the probands had to use all the previously learned commands to solve the shown puzzles. The objects had various colors, in order to help the users to choose the correct ordering of the objects, because if two objects are combined with one of the commands from subtask 1, the color of the lower object is used for the outcome.

Following, one example puzzle is described in detail to clarify the users' task.

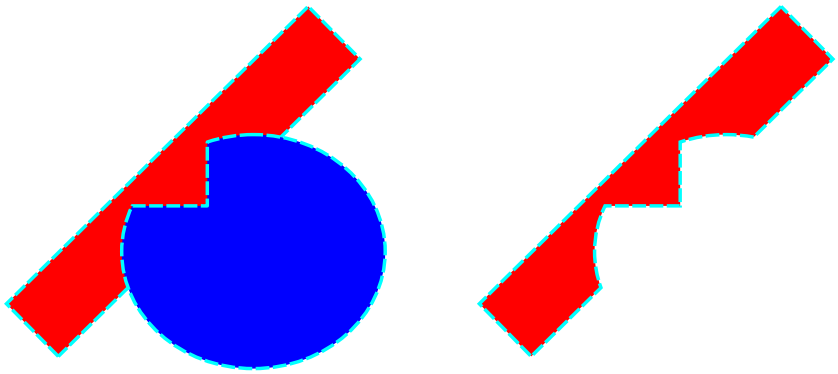
Example: exercise 2, subtask 3, puzzle 3

In this example, the given objects are a green rectangle, a red rectangle and a blue circle (see figure 4.5(a)). The final result has to look like figure 4.5(d). As the remaining color is red, we already know, that the red rectangle has to be the lowest tile. Another hint for the correct transformation is the circular edge on the final result. Because of this, we can first select all objects and flip them horizontally and vertically before we continue (see figure 4.5(b)). In order to get the correct result, we now have to cut the blue circle with the green rectangle. This is done by selecting these objects and by using



(a) start

(b) after horizontal and vertical flip of all three objects



(c) after usage of the difference command on the green rectangle and the blue circle and lowering command on the red rectangle

(d) correct result after usage of the difference command on the blue circle and the red rectangle

Figure 4.5: puzzle 3 of subtask 3 in exercise 2

the *difference* command. As we already mentioned in the beginning, we can also lower the red rectangle, so that the remaining object will also be red (see figure 4.5(c)). Last, we cut out the blue object from the red rectangle by selecting them and by using the *difference* command again. By doing this, we get the correct result for this puzzle (see figure 4.5(d)).

4.3.2 Questionaires

After the exercises, the probands had to fill out two questionaires. The first one was a general questionaire (see appendix C.1) about age, gender, occupation and familiarity with multi-touch devices. This questionaire was

used to get information about the users' experience of working in computer science (all probands were related to the computer science department), as well as their operating experience of multi-touch devices, as this could have an impact in the interview about the whole concept and also on the ability to use the touch screen.

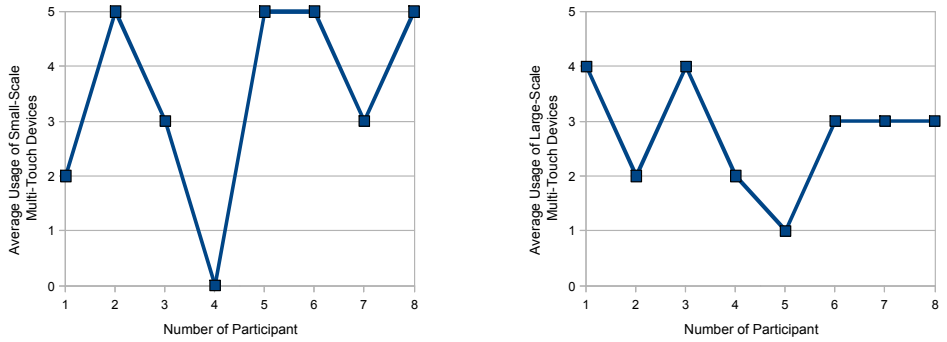
The second questionaire was used to evaluate the concept of showing shortcut icons on the keyboard. In order to get a quantitative result, that could give us an impression of whether the system should be evaluated further, the *Standard Usabilty Scale* (SUS) by Brooke[4] was used. While normally the SUS is used to compare different systems with eachother, we used the result as a hint, if the system is worth to conduct deeper research in this concept. The details of the SUS are described in the following part.

SUS - System Usability Scale

The System Usability Scale (see appendix C.2) consists of ten statements. Five of these statements are positive and five are negative. For each statement, the proband has to choose, how strong he agrees or disagrees to the statement by marking one of five checkboxes, whereat the first one equals *Strongly disagree* and the last one equals *Strongly agree*. Each mark is then given a value between 0 (most negative) and 4 (most positive). While the single values only give insignificant information about the system, the sum, which is multiplied by 2.5 in order to get a more convenient scale between 0 and 100, can be used as a "composite measure of the overall usability of the system being studied" [4]. As already mentioned, a higher result means a more positive feedback from the probands.

4.3.3 Interview

After the completion of the questionaires, the users participated in a short interview. The interview was performed as a discussion about the whole concept of the program. To guide the discussion in the intended direction, the users were asked three main questions. The first one, which also started the interview, was the following: *Could you imagine to use the system for daily work?* It was added, that they should imagine a further progressed solution of the system, which is already integrated in the normal desk. After a short discussion, they were asked: *Do you have any suggestions for improvements?* To get more personal comments on the system, the last question was *Why would you recommend using the system to your friends or collegues?*



(a) Small-Scale Devices (b) Large-Scale Devices

Figure 4.6: Average usage of multi-touch devices of the probands. The values from 0 to 5 equal the different options of the general questionaire with 5 being the most frequent.

The results of the interview will be given in a summarized form in section 4.4.3, as the full notes of the interviews can't be given in detail, because it was declared to the probands, that the recorded data will be kept confidential.

4.4 Analysis

The results of the evaluation will be detailed in this section. The demographic profile of the participants will be given in the first part of this section. Afterwards, the results of the System Usability Scale will be analysed. Last, a report of the interview is given.

4.4.1 Demographic Profile

The probands of the user study were randomly chosen. In total, eight people attended the evaluation. Therefrom, seven were male and one was female. Five were Ph.D. students in computer science, one computer science student, one Dr. rer. nat. and one high-school graduate. The participants were between 21 and 35 years old with an average age of about 28 years. The users' experience of small-scale and large-scale multi-touch devices is shown in figures 4.6(a) and 4.6(b). Most of the probands already had some experience with large-scale multi-touch devices and some use small-scale devices every day.

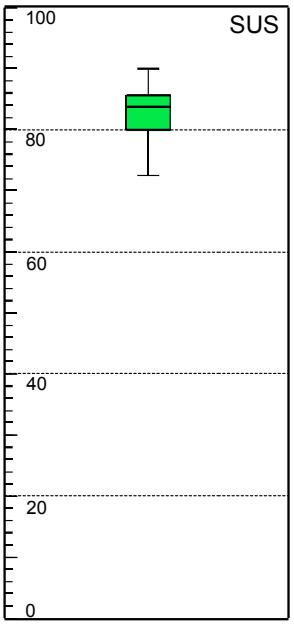


Figure 4.7: boxplot of the SUS score

4.4.2 System Usability Scale

Due to the fact that the System Usability Scale was only intended for evaluating the usefulness of the adaptivity of the keyboard, the probands were instructed to base the rating in the SUS only on this aspect of the system. As already described before, the SUS score ranges from 0 to 100, with 100 being the most positive value. The following values of the SUS score are also visualized in a boxplot (see figure 4.7). The median value of the study was 83.75, the lowest value 72.5 and the highest 90. The lower quartile was 80 and the upper quartile 85.625. While these results are quite positive, the interview is also taken into account, before we draw conclusions about the system. The interpretation of the results will therefore be described in subsection 4.4.4.

4.4.3 Interview

After the questionaires, the probands participated in a short interview about the whole concept of the thesis. The positive and negative aspects, that the users mentioned will be detailed in the subsequent paragraphs. The exact numbers of the most named aspects are listed in table 4.1 and table 4.2.

Positive Aspects

More than half of the users thought, that the adaptive keyboard is very useful. In detail, they talked about the scalability, the multifunctional displaying of key icons and the formability of the keyboard (e.g. natural keyboard). Especially the shortcut icons were considered very useful in order to get used to non frequently used shortcuts. In this connection, a few users also suggested, that the keyboard layout of the shortcuts could be improved in most programs, so that they are shown in context groups rather than associated keys. For example, the first five commands they had to use in *Inkscape* during the tests could be grouped to *CTRL + A/S/D/F/G* instead of being spread over the keyboard with *CTRL* and *CTRL + SHIFT* combinations. Two other users also mentioned, that they could imagine seperate keyfields containing the shortcuts of a program. Altogether, the probands liked the usage of multi-touch gestures (e.g. scrolling on the MTPad) and the flexibility of the direct touch input. Especially the participants, who are frequently using programs with additional toolsets, like graphics editors, game editors or electronic layout editors, could imagine, that the use of direct touch input would improve their efficency at working with those programs.

Some other advantages which were only mentioned by one person each are listed in the following:

- the desk should only react to fingers, not to objects, so you can still use your desk normally
- the shortcut icons could be explained on mouseover
- additional applications like a mixing desk for sound editors
- free space for paper work, when the computer is not used (no keyboard or mouse occupying the desk)
- easier to clean than mouse and keyboard
- additional interaction elements
- programs could be adapted for new multi-touch gestures

Number of Participants	Positive Aspect
5 of 8	adaptive keyboard
5 of 8	increase learnability of shortcuts with keyboard icons
5 of 8	flexibility with additional multi-touch input
4 of 8	good for programs with toolsets

Table 4.1: positive aspects

Number of Participants	Negative Aspect
7 of 8	haptic feedback
5 of 8	separate mouse cursor for direct touch area
5 of 8	modified touch area
5 of 8	solid sensors and calibration

Table 4.2: negative aspects

Negative Aspects

As we anticipated, the most wanted aspect for our concept is the haptic feedback. Nearly all users stated, that they could imagine using the system without haptic feedback for work, where they don't have to type longer texts, but for writing or programming they would prefer a keyboard with haptic feedback. As a small side note, it was very interesting, that the youngest participant thought that maybe he wouldn't miss the haptic feedback and that the users would get used to the software keyboard. More than half of the participants also mentioned that another important aspect that needs to be improved is the sensitivity and calibration of the system, so that unintended touches or mouse clicks are not triggered. Other main problems, which can be solved quite easily, were, that the mouse cursor should not be influenced by direct touch input and that the direct touch area should be limited to the area above the keyboard, so that the user can rest his hand on the screen below the keyboard without triggering direct touch input. One user also mentioned another aspect that could be a problem, namely that the vertical monitor doesn't react to direct touch.

4.4.4 Interpretation

Considering the high SUS score and the positive feedback during the interview, the implementation of an adaptive keyboard on a multi-touch desk

seems to be very useful. Therefore, this concept should be further improved and deeper investigated in further user studies. Additionally, the interview showed, that the users could imagine to rather use a multi-touch desk, instead of a conventional keyboard and mouse.

Yet, the system still needs a lot of improvements in matters of hardware solutions. The main problem that needs to be fixed, before the concept could replace conventional input devices, seems to be the haptic feedback. Furthermore, the sensors need to be very reliable. The hardware also should fit into a normal desk, so that it doesn't constrain the user. Another aspect is, that the multi-touch device should have a reasonable price level.

Summary

In this section, the conducted evaluation of the concept and in particular of the adaptive keyboard was described. First, the goals and setup were detailed. Subsequent, the user study was depicted in-depth. Last, the analysis of the conducted user study was given as well as an interpretation of the results. Ideas for future work will be suggested in the next chapter.

Chapter 5

Future Work

As this thesis has shown the potential of the introduced concept, further investigation in this topic seems to be useful. Some ideas for further reasearch on the processed topic are listed in this chapter.

5.1 Adaptive Keyboard

The adaptive keyboard could be improved for further investigation. The shortcut icon concept could therefore support more programs and additional language support could be implemented. Another aspect is to make the keyboard more flexible, so that each user can define the shape individually (e.g. for users who prefer natural keyboards). The next evaluation step for the shortcut icons could be the comparison between different layouts of the shortcuts (e.g. ordered by menu appearance, groups of similar functions, etc.) and also the comparison between shortcuts on the keyboard versus shortcuts on additional buttons.

5.2 Multi-Touch Input

A big challenge in the future will be the support of multi-touch gestures in standard programs. The task here lies within the finding of fitting gestures, so that programs can really be controlled by multi-touch gestures rather than just mapping the gestures to ancient commands.

Additionally, different input devices could also be implemented. While standard devices like a piano or a mixing desk are already a neat addition, new devices that make use of multi-touch input would be also very interesting.

5.3 Testing the Concept

The whole concept of a multi-touch desk in addition to the normal computer screen could be evaluated in a long term study in order to get information about unforeseen problems. A long term study is required, so that the participants get used to the system in their daily work, where they could eventually identify any drawbacks of the system.

Chapter 6

Conclusion and Review

6.1 Conclusion

In this thesis, the concept of using multi-touch surfaces as input devices for other computers has been introduced. By using a multi-touch surface, which is built into a standard desk, conventional input devices could be replaced through this concept. In order to get a first impression of the usability of the concept, a prototype was implemented during this thesis. In the prototype, an adpative keyboard, a multi-touch pad and a direct touch area was realised. The adaptive keyboard featured the visualization of shortcut icons for several programs. The prototype was then evaluated in a small user study, to gather information about problems of the concept and also to get suggestions for improving the system.

6.2 Review

As a small side note, we wanted to give an advice to the reader for improving future evaluation approaches. In our approach, we implemented a complete prototype before starting the evaluation. During the evaluation, it emerged, that we should've done several unbureaucratic tests prior to the systematic user study. As was also mentioned by Schwerdtfeger[17], a lot of minor bugs can be fixed in a system, if other people try it out in early stages of development. Therefore, the participants in the systematic user study can completely concentrate on the given tasks without being irritated by minor flaws.

Appendix

Appendix A

Documentation of the Prototype

This short documentation of the prototype implementation provides detailed information about important classes and functions. The class diagram of the implementation is shown in figure A.1.

A.1 Class Diagram

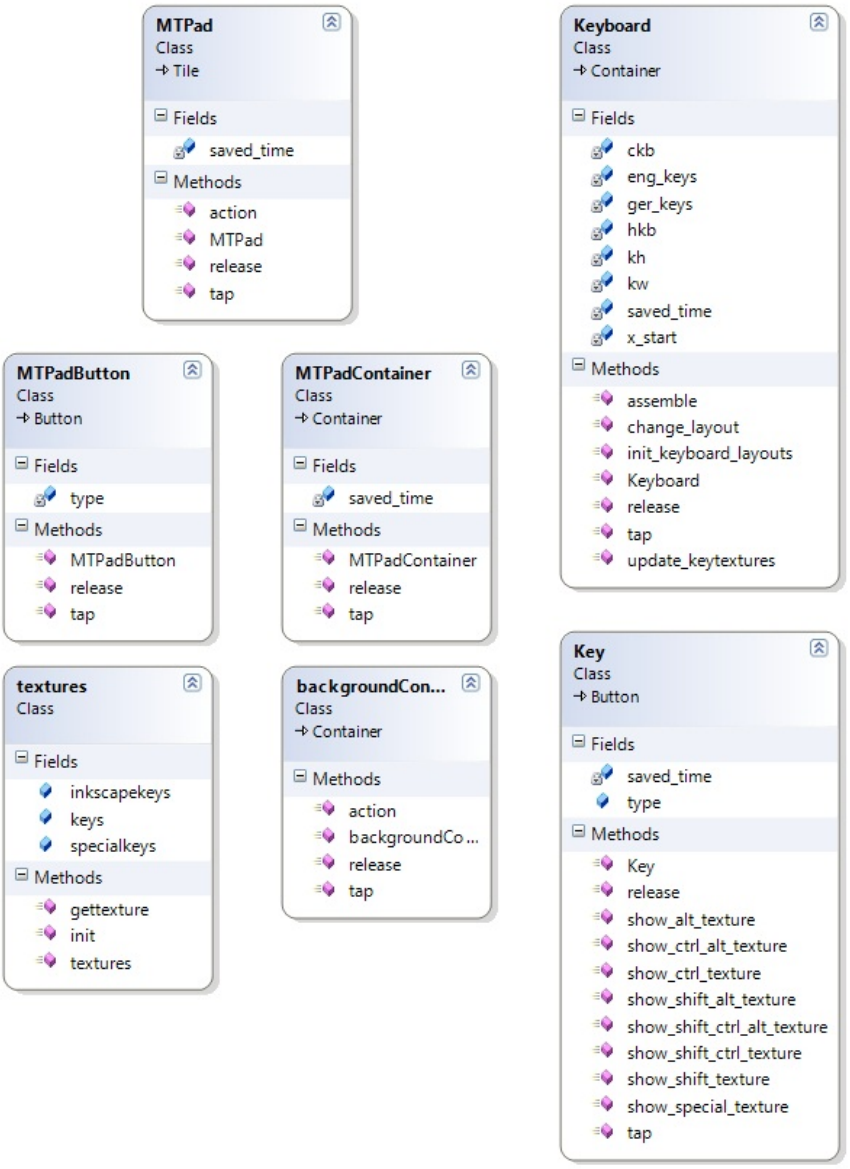


Figure A.1: class diagram

A.2 Classes and Functions

`global`

Description

Contains global functions and variables.

Methods

`void MouseMove(Vector vec)` generates relative motion of the cursor

`void MouseMoveABS(Vector vec)` moves the cursor to absolute position

`void MouseScroll(Vector vec)` simulates scrolling (invoked by move on MTPad with 2 or more fingers)

`void MouseLeftclick()` generates left_mouse_button click

`void MouseDown(int type)` generates a mouse button press (`type == 0` for left_mouse_button, `type == 1` for right_mouse_button)

`void MouseUp(int type)` generates a mouse button release (`type == 0` for left_mouse_button, `type == 1` for right_mouse_button)

`void always_on_bottom()` called on every button event to ensure that the program window stays on bottom

`int check_keyboard_layout()` checks, if the current keyboard layout equals the language of the foreground window. returns `true`, if a change is needed

`MTPadContainer`

Description

Always contains a MTPad widget and two MTPadButtons.
By tapping the visible part of the MTPadContainer (normally title bar of the MTPad), the MTPad can be moved/scaled.
By tapping the MTPadContainer again, the MTPad is fixed and can be used normally again.

Definition

`class MTPadContainer : public Container`

Constructor `MTPadContainer(int w, int h, int x, int y, double angle = 0.0, RGBATexture* tex = 0, int mode = 0)`

Methods

`void tap(Vector vec, int id)` saves current tick count.
`void release()` a "touch" has been detected, if a maximum of 300 milliseconds has passed since "tap"

`MTPadButton`

Description

MTPadButton to simulate left/right click

Definition

`class MTPadButton: public Button`

Constructor `MTPadButton(int w, int h, int x, int y, int _type, double angle = 0.0, RGBATexture* tex = 0)` the variable `_type` defines left or right click

Methods

`void tap(Vector vec, int id)` simulates the keypress of a mouse button
`void release()` simulates the release of a mouse button

`MTPad`

Description

The MTPad works like a normal touchpad. Additionally multi-touch scrolling is supported.

Definition

`class MTPad: public Tile`

Constructor `MTPad(int w, int h, int x, int y, double angle = 0.0, RGBATexture* tex = 0, int mode = 0xFF)`

Methods

`void tap(Vector vec, int id)` saves current tick count

`void release()` leftclick is generated, if a maximum of 50 milliseconds has passed since ”tap”
`void action(Gesture* gesture)` moves mouse cursor, simulates scrolling or calls the `parent->action` method (to scale/move the MTPad)

`Keyboard`

Description
Contains all buttons of a standard Keyboard.
By tapping the visible part of the Keyboard (normally title bar of the Keyboard), the Keyboard can be moved/scaled.
By tapping the Keyboard again, the Keyboard is fixed and can be used normally again.

Definition
```
class Keyboard :  public Container
```

Constructor `Keyboard(int w, int h, int x, int y, double angle = 0.0, RGBATexture* tex = 0, int mode = 0)`

Methods
`void init_keyboard_layouts()` defines the german and english keyboard layout
`void assemble()` places the keys onto the keyboard
`void tap(Vector vec, int id)` saves current tick count
`void release()` a ”touch” has been detected, if a maximum of 300 milliseconds has passed since ”tap”
`void change_layout()` changes the layout to the current keyboard language
`void update_keytextures()` updates the keytextures according to pressed special keys

`Key`

Description
The Key widget generates the predefined input when being tapped.

Definition
```
class Key :  public Button
```

Constructor `Key(int w, int h, int x, int y, int type, double angle = 0.0, RGBATexture* tex = 0)` the `type` variable defines the keytype

Methods

`void tap(Vector vec, int id)` simulates a keypress

`void release()` simulates the release of a key

`void show_special_texture(int active_window_number)` shows special texture for active program

`void show_ctrl_texture(int active_window_number)` shows texture for active program, while ctrl key is pressed

`void show_alt_texture(int active_window_number)` shows texture for active program, while alt key is pressed

`void show_shift_texture(int active_window_number)` shows texture for active program, while shift key is pressed

`void show_shift_ctrl_texture(int active_window_number)` shows texture for active program, while ctrl key and shift key are pressed

`void show_shift_alt_texture(int active_window_number)` shows texture for active program, while alt key and shift key are pressed

`void show_ctrl_alt_texture(int active_window_number)` shows texture for active program, while ctrl key and alt key are pressed

`void show_shift_ctrl_alt_texture(int active_window_number)` shows texture for active program, while ctrl, alt and shift are pressed

`textures`

Description

Contains all textures.

Definition

`class textures`

Constructor `textures()`

Methods

`void init()` loads all textures and stores them into texture maps

`RGBATexture* gettexture(int number)` returns standard key texture of `number`

backgroundContainer

Description

The backgroundContainer generates mouseevents from touches on the screen.

Definition

```
class backgroundContainer :  public Container
```

Constructor `backgroundContainer(int w, int h, int x, int y, double angle = 0.0, RGBATexture* tex = 0, int mode = 0)`

Methods

`void tap(Vector vec, int id)` moves the mouse cursor to the touch postion and generates a left_mouse_button press

`void release()` generates a left_mouse_button release

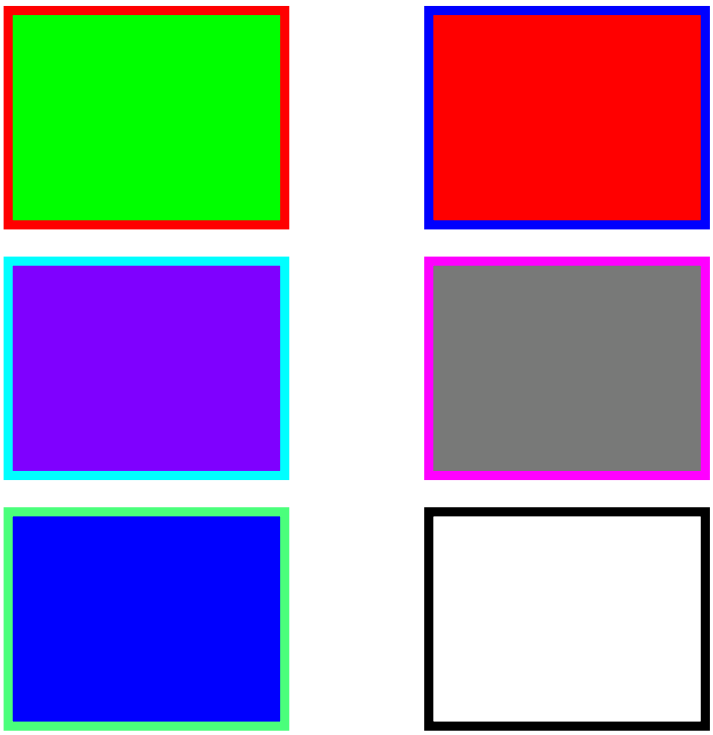
`void action(Gesture* gesture)` moves the mouse cursor relative to the finger

Appendix B

Test Excercises

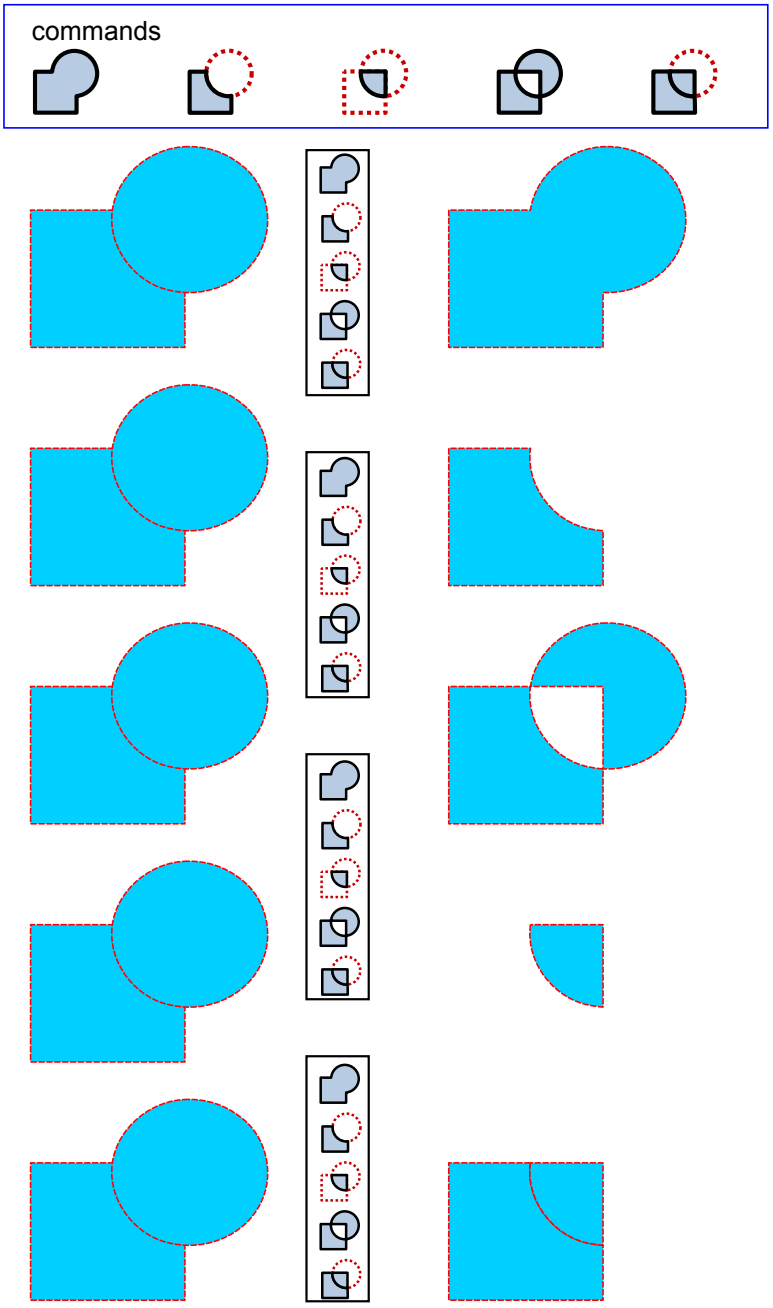
B.1 First Task

Change the color of the filling and the outline of the left objects by using direct touch, so that you get the same image as on the right.

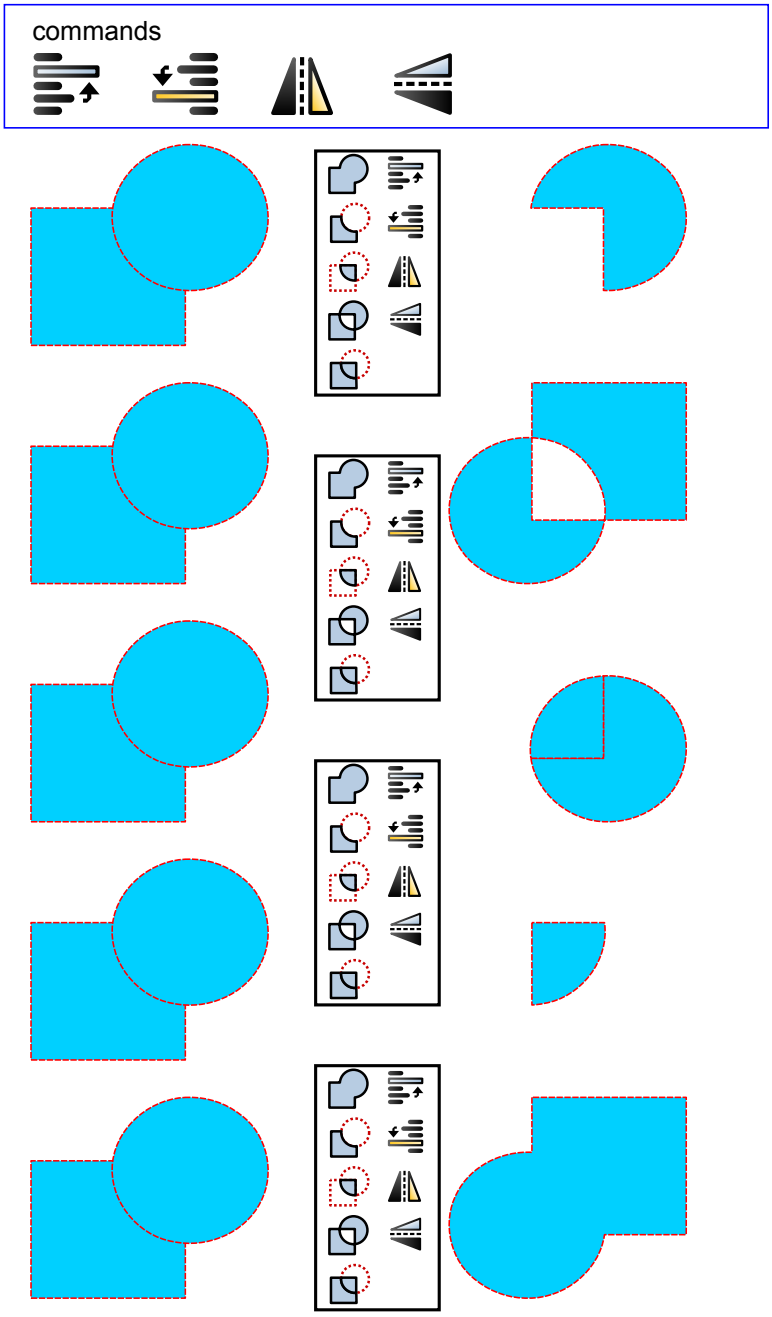


B.2 Second Task

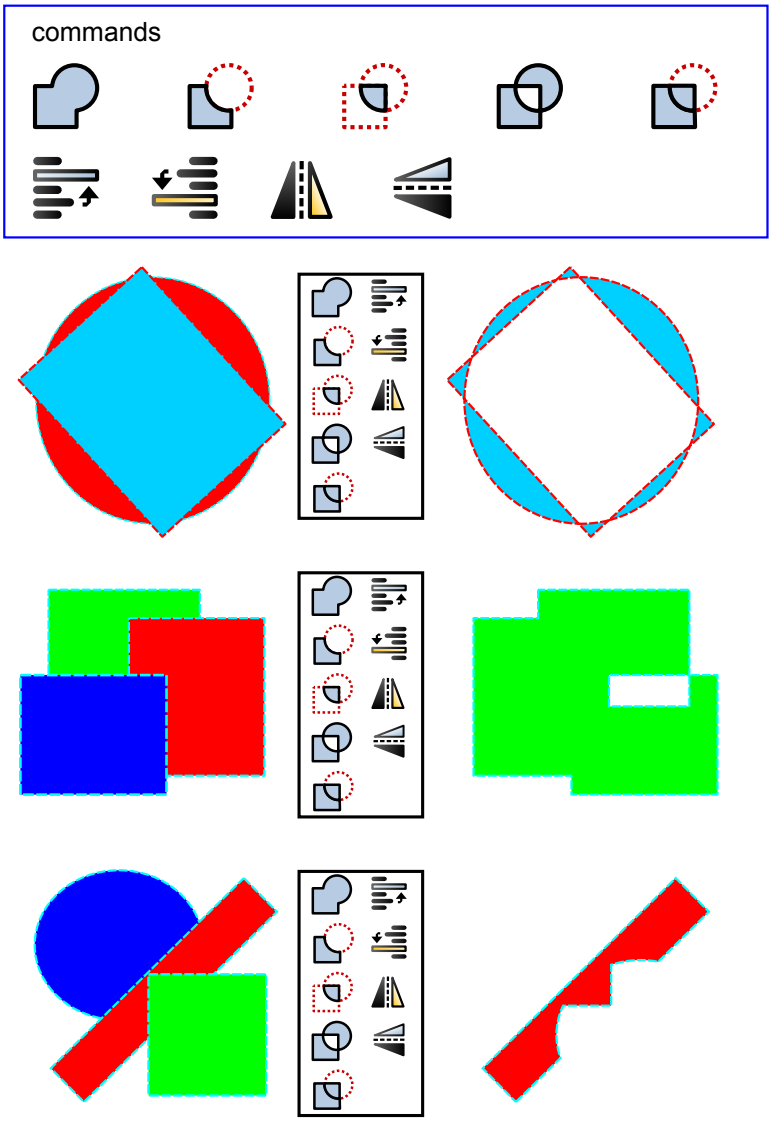
1. Use the following commands on the left objects, to create the image on the right.

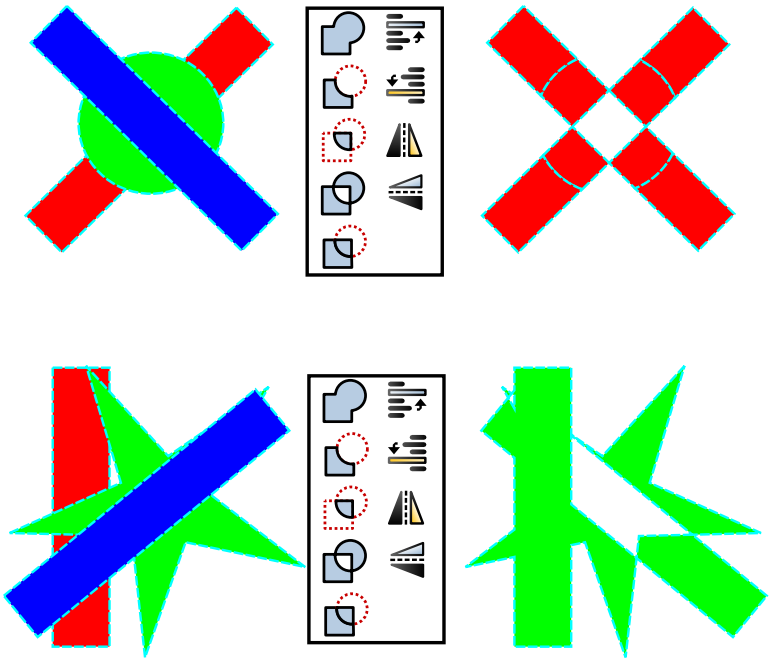


2. For this task, the following commands are required in addition to the previous ones:



3. Try to solve the puzzles using the previously learned commands.





Appendix C

Questionaires

C.1 General Questions

Questionnaire for the evaluation of MTInput

The recorded data will be kept confidential and will only be used for this evaluation. The analysis of the data will be made exclusively by the test administrator.The results will be presented in an anonymous and summarized form, so that conclusions about individual persons are not possible.

0. Participant Number: ☐

1. Age

☐

2. Gender

- ○ male
- ○ female

3. Occupation

- ○ student
- ○ PhD student
- ○ other:

☐

4. How often have you used large-scale multi-touch devices?
(e.g. Microsoft Surface, TISCH)

- ○ never
- ○ once
- ○ a few times a year
- ○ monthly
- ○ weekly
- ○ daily

5. How often have you used small-scale multi-touch devices?
(e.g. iPhone, iPad, Tablet PC)

- ○ never
- ○ once
- ○ a few times a year
- ○ monthly
- ○ weekly
- ○ daily

C.2 System Usability Scale

System Usability Scale

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

Bibliography

[1] Hrvoje Benko, Meredith Ringel Morris, A. J. Bernheim Brush, and Andrew D. Wilson. Insights on Interactive Tabletops: A Survey of Researchers and Developers, 2009.

[2] Dennis B. Beringer and James G. Peterson. Underlying Behavioral Parameters of the Operation of Touch-Input Devices: Biases, Models, and Feedback. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 27:445–458(14), August 1985.

[3] Florian Block, Hans Gellersen, and Nicolas Villar. Touch-Display Keyboards: Transforming Keyboards into Interactive Surfaces. In *CHI '10: Proceedings of the 28th international conference on Human factors in computing systems*, pages 1145–1154, New York, NY, USA, 2010. ACM.

[4] John Brooke. SUS - A quick and dirty usability scale. Usability Evaluation in Industry, 1996.

[5] Andrew Crossan, John Williamson, and Stephen Brewster. Artex: Artificial Textures from Everyday Surfaces for Touchscreens. In *CHI EA '10: Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, pages 4081–4086, New York, NY, USA, 2010. ACM.

[6] F. Echtler. *Tangible Information Displays*. PhD thesis, Technische Universität München, 2009. Available online at `https://mediatum2.ub.tum.de/node?id=796958`.

[7] Florian Echtler, Andreas Dippon, Marcus Tönnis, and Gudrun Klinker. Inverted FTIR: Easy Multitouch Sensing for Flatscreens. In *ITS '09: Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, pages 29–32, New York, NY, USA, 2009. ACM.

[8] Friedrich-Schiller-Universität Jena. SpeedUp Projekt. `http://www.speedup-projekt.de/`, August 2010.

[9] J.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, 2005.

[10] J.Y. Han. TED Talk. `http://www.ted.com/talks/jeff_han_demos_his_breakthrough_touchscreen.html`, 2006.

[11] N.D. Marchuk, J.E. Colgate, and M.A. Peshkin. Friction Measurements on a Large Area TPaD. In *Haptics Symposium, 2010 IEEE*, pages 317–320, 25-26 2010.

[12] Microsoft. Microsoft Surface. `http://www.microsoft.com/surface/`, August 2010.

[13] Microsoft. UIST 2010 Student Innovation Contest. `http://www.microsoft.com/appliedsciences/content/projects/uist.aspx`, 2010.

[14] M.R. Morris, A.J.B. Brush, and B.R. Meyers. Reading Revisited: Evaluating the Usability of Digital Display Surfaces for Active Reading Tasks. pages 79–86, oct. 2007.

[15] Gian Pangaro, Dan Maynes-Aminzade, and Hiroshi Ishii. The Actuated Workbench: Computer-Controlled Actuation in Tabletop Tangible Interfaces. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 181–190, New York, NY, USA, 2002. ACM.

[16] Johannes Schöning, Peter Brandl, Florian Daiber, Florian Echtler, Otmar Hilliges, Jonathan Hook, Markus Löchtefeld, Nima Motamedi, Laurence Muller, Patrick Olivier, Tim Roth, and Ulrich von Zadow. Multi-Touch Surfaces: A Technical Guide. Techreport, Technische Universität München, 2008.

[17] Björn Schwerdtfeger. *Pick-by-Vision: Bringing HMD-based Augmented Reality into the Warehouse*. PhD thesis, Technische Universität München, 2010. Available online at `https://mediatum2.ub.tum.de/node?id=992985`.

[18] Andrew Sears. Improving Touchscreen Keyboards: Design issues and a comparison with other devices. *Interacting with Computers*, 3(3):253–269, 1991.

[19] J.D. Smith, T.C.N. Graham, D. Holman, and J. Borchers. Low-Cost Malleable Surfaces with Multi-Touch Pressure Sensitivity. In *Horizontal Interactive Human-Computer Systems, 2007. TABLETOP '07. Second Annual IEEE International Workshop on*, pages 205–208, 10-12 2007.

[20] Art Lebedev Studio. Optimus Maximus Keyboard. `http://www.artlebedev.com/everything/optimus/`, 2010.

[21] Carlo Tomasi, Abbas Rafii, and Ilhami Torunoglu. Full-Size Projection Keyboard for Handheld Devices. *Commun. ACM*, 46(7):70–75, 2003.

[22] Pierre Wellner. The DigitalDesk Calculator: Tangible Manipulation on a Desk Top Display. In *UIST '91: Proceedings of the 4th annual ACM symposium on User interface software and technology*, pages 27–33, New York, NY, USA, 1991. ACM.

[23] W. White. Method for Optical Comparison of Skin Friction-Ridge Patterns. U.S. Patent 3,200,701, 1965.

[24] D. Wigdor, G. Perm, K. Ryall, A. Esenther, and Chia Shen. Living with a Tabletop: Analysis and Observations of Long Term Office Use of a Multi-Touch Table. pages 60–67, oct. 2007.

[25] Raphael Wimmer, Fabian Hennecke, Florian Schulz, Sebastian Boring, Andreas Butz, and Heinrich Huß mann. Curve: Revisiting the Digital Desk. In *NordiCHI 2010: 6th Nordic Conference on Human-Computer Interaction (to appear)*, New York, NY, USA, 2010. ACM.

[26] Laura Winfield, John Glassmire, J. Edward Colgate, and Michael Peshkin. T-PaD: Tactile Pattern Display through Variable Friction Reduction. *World Haptics Conference*, 0:421–426, 2007.

[27] Yvonne Jansen. Mudpad: Fluid Haptics for Multitouch Surfaces. In *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, pages 4351–4356. Conference on Human Factors in Computing Systems, ACM, 2010.