glueTK:
A Framework for Multi-modal, Multi-display Interaction

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte
Dissertation

von

Florian van de Camp

aus Duisburg

Tag der mündlichen Prüfung:	9. Mai 2014
Erster Gutachter:	Prof. Dr.-Ing. Rainer Stiefelhagen
Zweiter Gutachter:	Prof. Dr.-Ing. habil. Björn Hein

Zusammenfassung

In den letzten Jahren hat es eine verstärkte Entwicklung neuartiger Eingabegeräte verschiedenster Modalitäten gegeben, mit einem Schwerpunkt auf natürlicher und intuitiver Interaktion. Diese Eingabetechnologien basieren darauf, den oder die Benutzer mit geeigneter Sensorik zu beobachten und daraus Informationen, wie Zeigegesten, Personenlokalisierung oder die Schätzung der Kopforientierung abzuleiten. Um diese Informationen der neuartigen Eingabetechnologien bestmöglich nutzen zu können sind neue Konzepte zur Entwicklung von grafischen Benutzerschnittstellen notwendig, um die Eingabedaten zu komplementieren. GlueTK ist ein Softwareframework, das die Entwicklung von Applikationen ermöglicht, die mittels neuartiger Eingabegeräte gesteuert werden und sich über mehrere Displays erstrecken können. Eine zentrale Eigenschaft ist dabei die bidirektionale Kommunikation zwischen der Eingabeschicht und der Ausgabeschicht, die den Eingabetechnologien Informationen über die aktuelle Darstellung zur Verfügung stellt. Ein netzwerktransparentes Signal- und Slot-System erlaubt eine umfassende und flexible Kommunikation im gesamten Framework.
Aufbauend auf der Funktionalität von glueTK ist ein System zur Verbesserung der Zielerfassung für Zeigegestensysteme entwickelt worden. Die so genannten “dynamischen Gauß’schen Kraftfelder” erweitern bisherige Ansätze in mehrfacher Hinsicht. Die Modellierung der Anziehungskraft der Felder mittels einer Gauß-Verteilung macht die Bestim-

mung einer sinnvollen, statischen Anziehungskraft überflüssig. In dem die Orientierung der Kräfte aller Felder in Betracht gezogen wird, kann die Platzierung der Felder in komplexen Anwendungen auch für überlappende Felder generisch gelöst werden. Da Kraftfelder prinzipiell immer zu Irritationen führen können wenn sie nicht gebraucht werden, treffen die dynamischen Kraftfelder Vorhersagen welche Elemente anvisiert werden. Basierend auf diesen Vorhersagen werden dann nur die Kraftfelder anvisierter Elemente aktiviert.
Kraftfelder, wie auch die Mehrheit anderer Systeme zur Verbesserung der Zielerfassung von Zeigesytemen, benötigen Informationen über die aktuelle Darstellung auf dem Display. Diese Informationen sind bei glueTK-basierten Applikationen grundsätzlich verfügbar, allerdings nicht bei der großen Zahl bereits bestehender Applikationen. Für solche Anwendungen, speziell wenn kein Zugang zu dem Quelltext der Anwendung möglich ist, wurde ein System entwickelt, welches mittels Bildverarbeitung dargestellte Elemente automatisch erfasst und so die notwendigen Informationen über das aktuelle Layout liefert. Diese Information werden dann genutzt um Eingabedaten um Kraftfelder zu erweitern bevor diese an die Zielanwendung in Form von Mauskoordinaten weitergeleitet werden. Um die dargestellten Elemente zu lokalisieren wird der Bildschirminhalt erfasst und die einzelnen Elemente werden mit einem Template-Matching-Verfahren gesucht. Um zu verhindern, dass für eine große Zahl von Elementen Templates erstellt werden müssen, werden diese zur Laufzeit erstellt indem das Verhalten des Benutzers beobachtet wird. Durch eine

automatische Generalisierung der Templates werden Modelle für einzelne Elementtypen erstellt, die es erlauben, alle Elemente so schnell zu lokalisieren, dass die gewonnene Information zur Verbesserung von Zeigegesten für beliebige Anwendungen verwendet werden kann.
Mit Hilfe der dynamischen Gauß'schen Kraftfelder und dem automatischen Lokalisieren von Elementen in beliebigen Applikationen ist die Zielerfassung präzise genug, um neue sowie bestehende Anwendungen zu bedienen. Sobald das Zielelement erfasst ist, muss es darüber hinaus aber auch noch aktiviert oder angewählt werden. Eines der großen Vorteile der Zeigegestensysteme ist ihre Natürlichkeit, da Benutzer keinerlei Geräte tragen oder halten müssen um zu interagieren. Es ist daher erstrebenswert, auch für die Zielaktivierung auf zu tragende Geräte zu verzichten und diese Information ebenfalls aus der visuellen Erfassung zu gewinnen.
Durch eine neue, umfassende Taxonomie wird ein umfassender Überblick über Möglichkeiten gegeben, Ziele ohne zusätzliche Geräte, allein mit dem zeigenden Arm und der Hand zu aktivieren. Dafür werden mögliche Aktivierungsgesten in Arm-, Hand-, und Fingergesten unterteilt, um eine systematische Auswertung zu ermöglichen. In einer Wizard-of-Oz Studie werden alle Klassen der Taxonomie untersucht. Die Ergebnisse erlauben über die Empfehlung einer optimalen Geste hinaus auch Hinweise für einen sinnvollen Einsatz der anderen Gesten.
Das Design von grafischen Benutzerschnittstellen ist ein wichtiger Aspekt von glueTK Anwendungen. Dies ist zum einen darin begründet, dass viele neuartige Eingabegeräte

nicht an ein einzelnes Display gebunden sind und Interaktion über Displaygrenzen hinweg erlauben. Zum anderen gibt es eine große Zahl von Displays mit verschiedenen Größen, Auflösungen und Seitenverhältnissen: Faktoren die alle bei der Gestaltung von Anwendungen berücksichtigt werden müssen. Bei der Gestaltung hat vor allem die Größe der Elemente einen großen Einfluss auf die Interaktionsleistung, also darauf, wie schnell ein Benutzer eine Anwendung bedienen kann. Für eine genauere Analyse wird die Interaktionsleistung in zwei Phasen unterteilt. Die Zeit, ein Element visuell zu lokalisieren und zu identifizieren, wird hier als Wahrnehmungsleistung bezeichnet. Die Zeit, dieses Element dann mit einem Eingabegerät zu aktivieren, wird als Eingabeleistung bezeichnet. Um bei der Vielfalt von Eingabegeräten und Anzeigegeräten Vorhersagen über geeignete Elementgrößen machen zu können, wird das Konzept eines Referenzsystems entwickelt. Basierend darauf lassen sich Aussagen darüber treffen, wie die Interaktionsleistung sich auf einem beliebigen Zielsystem verhalten wird. Die Vorhersage der Interaktionsleistung ermöglicht die Entwicklung gut bedienbarer Anwendungen für jede Kombination von Eingabegerät und Display bei minimalem Entwicklungsaufwand.

ABSTRACT

In recent years, many novel input modalities have been explored with a special focus on natural and intuitive interaction. These modalities focus on the user and on deriving input data from the user's actions such as pointing gestures, person tracking, or headpose estimation. To make use of these modalities in the best possible way, novel concepts for the development of user interfaces are necessary to complement the input. GlueTK is a framework, which allows the creation of applications that can be controlled using novel input modalities, and can spread across multiple displays and machines. A central property is the bidirectional communication between the input and output layers, which makes interface layout information available to input modalities. A network-transparent signal and slot system provides a coherent communication mechanism within a glueTK application.

A pointing enhancement technique for improving the target acquisition of a pointing gesture recognition system builds upon the functionality provided by glueTK. The technique, called dynamic Gaussian force fields, extends previous force field variants in several aspects. The Gaussian modeling of the field strength solves the problem of defining a single fixed strength, which is always a trade-off between help and irritation. By allocating for multi-directional forces of overlapping fields, a generic solution to the overlap problem is presented, which allows force field placement even for complex user interfaces. As force fields can always be

a source of irritation when not needed, dynamic Gaussian force fields predict if an interface element is targeted from the users pointing data and dynamically turn fields on and off.
While the layout information required by target aware pointing enhancements is available for glueTK applications, the same is not true for other applications. A computer vision based approach to automatically detect the user interface layout of those applications, even without access to the source code, solves this problem. By acting as an intermediary between the existing user interface and the input data of an input device to be integrated, force fields can be used for existing applications without the applications knowledge. The approach involves capturing the screen and using template matching to localize targets. To avoid the need for prior training or configuration, target templates are created on the fly by observing the user while he interacts. By automatically generalizing target models, using multiple acquired templates, the localization works fast enough to enhance the target acquisition of pointing gestures for any existing application.
With the ability to improve the speed and accuracy of target acquisition not only for glueTK applications, but also for existing applications by utilizing the gained knowledge about the layout of the user interface, target acquisition is fast and accurate. Once the target is acquired, however, it has to be selected as well. Since one of the advantages that make pointing gesture recognition natural and intuitive, is the lack of any devices the user is required to wear or hold, target selection should be device free as well.

A novel taxonomy gives a comprehensive overview of device free options to trigger a selection of a target in mid-air by systematically categorizing different arm-, hand- and finger gestures into respective classes to cover all possible ways of target selection. In a Wizard-of-Oz study, all gestures from the taxonomy are evaluated, and the results give recommendations about which gestures should be used for selection as well as which are suitable for secondary interaction tasks.
The design of user interfaces is an important aspect of glueTK applications. This is due to the fact, that modalities like pointing gestures are not bound to a single display and allow for interaction in multi-display environments. At the same time, the different display and modality properties have to be taken into account. When it comes to the affect the user interface has on the interaction performance, two distinctive parts of the interaction can be identified. Before an input device is even used, a target has to be visually acquired by the user. This perceptual performance has been studied for text but not for graphical interface elements. Therefore, the concept of a reference system is introduced, which allows developers and interface designers to put design decisions they make into perspective by allowing the calculation of required element sizes for a target system to achieve the same perceptual performance as on the specific reference system they use. The other distinctive part of the overall interaction performance is the input performance, the time it takes a user to select a target with a given input device, once the target is visually acquired. An adaptation of the well known Fitts's law allows for pre-

dictions with respect to a reference system for the input performance with similar accuracy as the predictions for the perceptual performance. In addition, the dependency between perceptual performance and input performance is analyzed in detail and allows for predictions of the overall interaction performance.

Acknowledgements

On April 4th 2005, I sent an E-mail to Prof. Dr. Stiefelhagen asking if I could participate in a seminar on multimodal rooms as part of my university studies even if I could not attend the first meeting. Little did I anticipate that his agreement would be the start of years of research in this very field for me. I would like to thank Prof. Dr. Stiefelhagen for this opportunity and his guidance and support over the years, especially throughout the course of this thesis. I would like to express my gratitude towards Prof. Dr. Hein as co-referee of this thesis. I would particularly like to thank my colleagues at the Fraunhofer IOSB, Alexander Schick, Joris IJsselmuiden and Dr. Michael Voit. All of our work within the scope of the Attract project complemented one another so well that it allowed each of us to pursuit our own specific challenges with the results culminating in the SmartControlRoom. I would like to thank all of you for

countless discussions, proofreading and companionship.
I would like to express my gratitude towards Dr. Geisler and Dr. Peinsipp-Byma, who have not only shown great interest in my work, but also created the frame conditions that allowed me to work on my thesis as well as on many interesting projects. I want to thank Prof. Dr. Beyerer, without whom the Attract project would not exist. Though not directly involved in this thesis, I would like to thank my parents. From an early age on my interest in technology was always met with encouragement, and on the way from there to here I have received much support, morally and financially. Finally, I would like to thank Marleen, who has always been there for me. Always.

Contents

1 Introduction

Since the introduction of the first mouse based desktop computer with a graphical user interface, the Xerox Alto in 1973, not much has changed in the way humans interact with desktop computers. The “windows, icons, menus, pointer” (WIMP) - paradigm that was introduced to complement the then novel pointing input has been prevalent in the computing world for more than thirty years.
One reason is the generality of the paradigm that extends to a wide range of applications for desktop computers and was assisted by commercial widget toolkits on the rise that followed the WIMP-paradigm. Another important reason

is that the mouse alongside the keyboard are an extremely powerful pair of input devices that allows for high efficiency in the use of a desktop computer. The combination of these circumstances have left little demand for change.
In recent years, new computing devices and environments have surfaced. Mobile devices have become an inherent part of everyday computer use and are still on the rise, while the number of desktop computers is stagnating [Gar]. Mobile devices are not well suited for mouse and keyboard input, which has led to a rise of touch and multi-touch input as the dominant input modality for these devices. While there are already many commercial solutions and products in the mobile device market, another category of computing systems, large displays and multi-display environments, is also on the rise but not yet as pervasive as mobile devices. What the two categories have in common, however, is that mouse and keyboard are not perfectly suited for large displays and multi-display environments either, since these are commonly not used sitting down. Large displays are commonly used from several distances depending on whether the user wants to get an overview or take a look at details. In multi-display environments, users usually walk from one display to another depending on which system is best suited for the task at hand.
To provide the required flexibility, different input modalities have been explored and several novel input devices have recently become mature enough to be used in such environments. For developers, however, it is still difficult to utilize these devices as they all have different requirements, interfaces and partially provide very different input

data. Besides integration, the accuracy of many novel input devices poses a problem as well. Since they are usually not bound to a single display to allow for flexible use in multi-display environments and aim at providing a natural and intuitive user experience, most of them focus on observing the user's actions to derive information for interaction. While this approach has the great advantage of omitting the need to carry a hardware device, these devices are quite inaccurate compared to a mouse due to sensor noise and measurement errors. Even if sensors get better, modalities that focus on measuring movements of users are inherently inaccurate due to the human physique. For pointing gesture recognition, for example, does neither the extension of an arm necessarily reveal the exact, intended target the user points to [NS07], nor is a user able to keep the arm perfectly still.

To compensate for the inaccuracy of many novel input devices, interfaces have been adjusted by using larger elements where possible. A visual adjustment of interfaces is, however, only possible to a certain degree, as larger elements severely limit the possible functionality and complexity of user interfaces. Other techniques for improvement on the side of the interface exist, but rely on information about the layout of the interface. Traditionally, input devices have been unidirectional, meaning they have no knowledge of the user interface and no means of accessing this information. The more natural input devices are adopted, the more important the interface becomes, as it directly affects the user's actions, which in turn are captured by many input devices.

The following two sections will give an introduction to the challenges involved with the integration of new devices as well as the development of applications for multi-display environments, followed by an overview of the goals and contributions of this thesis.

1.1 New Input Modalities

The term "input modality" is ambiguously used in literature. Its use ranges from describing a specific device such as a mouse [NKG13] to denoting a very broad category such as computer vision based systems [TH13]. Within the context of this thesis it is used to define a class of input devices, such as touch based devices. An input device is a specific representative of that class, such as a capacitive touchscreen.
In recent years, many new input devices are starting to mature far enough to be used in applications, but there exists a large variety of interfaces along with different types of input data that is provided. While a first approach for many new input devices seems to be to emulate mouse data, this severely limits the full potential of the input modalities. In addition, many input modalities are not bound to a single screen or system, they are systems in their own right and provide data relative to their own coordinate system - which requires data transformations and projections before the interaction with screen content is possible. Because of the user and body focused approaches of many modalities, the inherent inaccuracy has to be compensated to allow for

meaningful interaction with real world interfaces. The user interface and its layout directly affect the user's behavior and while often thought of as mostly independent of the input modality, there has always been a close correlation between input modality and user interface.
For the first personal computers in the 1970s, text based interfaces were dominant and the only available input device was the keyboard. While the limitation to textual display was due to technical boundaries in the beginning, it was in perfect correspondence to the keyboard as an input device which was capable of only text input and very basic, typewriter like, navigation.
When Xerox introduced the Alto and later the Star in the mid 1970s [JRV+89], they introduced the computer mouse to be used as an input device in addition to the keyboard. Along with this new input technology, they also introduced a new concept of user interfaces, the graphical user interface (GUI) based on the WIMP-paradigm. The mouse would have been of little use in a purely text based user interface, the concept of windows, icons and menus allowed the mouse to prove its strengths. After this introduction, the basic concepts for both input devices as well as user interface have stayed the same for desktop computers until this day. There have been improvements and minor modifications like the switch from mechanical mouses to optical ones or the introduction of the scroll-wheel in the 1990s, when scrolling became a dominant task due to the dawn of the world wide web, but the devices and concepts have remained the same.

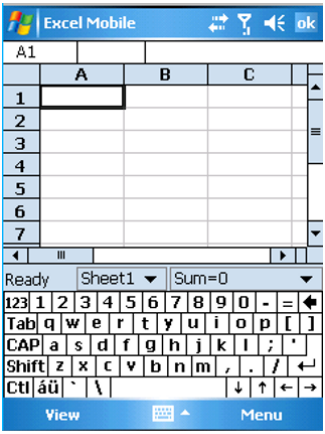


Figure 1.1: Microsoft Excel on Windows Mobile 5.

This changed when the first smartphones appeared in the early 2000s. The mobility of the device itself as well as limited space due to the small size rendered the use of a mouse impractical, which led to the widespread introduction of touch screens. The first user interfaces in widespread use for touch based smartphones were, however, not adapted to this new input modality and its properties. User interfaces were merely scaled down variants of desktop software, never intended to be used with any input device but a mouse. Figure 1.1 illustrates this with an example screenshot of Excel on Windows Mobile 5. Several years later the iPhone was introduced, which used touch input as well, but alongside a user interface that was specifically tailored towards the properties of this input modality. Larger ele-

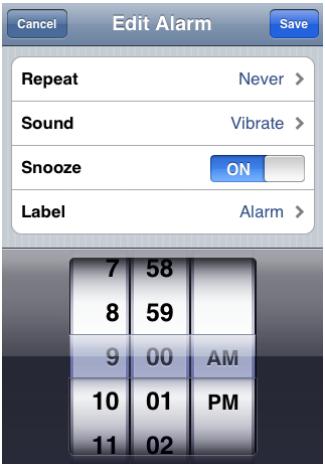


Figure 1.2: Setting an alarm on the iPhone (iOS 4).

ments to account for the lower accuracy as well as interface elements that reacted not to simple touch but to a swipe across them better utilized the expressiveness of the touch modality (See Figure 1.2). The adaptation of the interface to go along with the new modality made the device easier to use and led to explosive growth of the smartphone market that is still ongoing.

In 2010 the Microsoft Kinect [Kin] was introduced, and allowed the control of games with gestures and body movements. To make use of this input technology, the interface has been adapted to it as well. Because of the fact that the Kinect is an addition to a game console and intended for play, there are still many open problems to make use of this natural, intuitive input in professional applications.

1.2 Multi-display Environments

While mobile devices have broadened the range of computing systems used in everyday life with displays smaller than desktop computers in recent years, there are also more and more systems that use much larger screens than common desktop computers. While large displays, digital tables and videowalls are not common for personal computing, they become more and more common in professional settings. In many cases, these large displays are used alongside regular desktop computers as well as mobile devices, which leads to an increase of installations of multi-display environments. Especially in combination with novel input modalities, like pointing gestures, which are not bound to a single system or display, the demand from users for inter-display interaction grows. Users usually do not care about the underlying technical details but about applications. Since large displays and mobile devices are in fact, however, completely different machines, simple tasks like moving an object from one display to another can become a technical challenge.
Interacting with multiple displays also creates new expectations towards the interface design. Not only should interfaces provide a coherent user experience across displays and devices, but they should also be usable. The large variety of screen sizes, resolutions, ways of use, and new input devices that start to establish themselves, leads to very different requirements regarding the user interface to allow for the same level of usability. In most cases, interfaces have to be adapted to account for the visual acuity of normal sighted persons at a certain distance or for the

accuracy of an input device. The basic approach of simply increasing the size of textual and graphical elements is not always an option. In addition, screen estate is a very limited resource, which is why elements should never be larger than they have to be to allow for a certain level of interaction quality. Another reason why the design of interfaces for such novel interactive systems is usually a tedious and time consuming task, is that developers still use desktop computers to design these interfaces. This means, both input device and screen properties differ from the target system, which makes it hard to develop a feeling for "right" element sizes, particularly if there are more target systems with different properties.

1.3 Goals and Contributions

Within the scope of this work, a framework for multimodal, inter-display interaction was developed. The framework lays the necessary ground work to allow for the implementation of several contributions in the field of human computer interaction. The framework intends to be able to incorporate a wide variety of novel input devices and abstract from their specific interfaces to allow for easy integration. It needs to provide means of communication between input devices and interface elements across multiple machines to enable interaction across multiple displays, connected to different machines. To allow for interfaces tailored towards the properties of input modalities, displays and users, a flexible output system is required to render cus-

tom interfaces and elements. As the interface itself plays an important role in the utilization and improvement of many novel input modalities, providing interface layout information throughout the framework is an essential property.
As pointing gesture recognition is the most common novel input modality due to its versatile use cases, it has to be of special focus within the context of the application of the framework.
Given the display layout information by the framework, target aware pointing enhancements should be utilized to improve the accuracy and speed of target acquisition. While this can significantly improve the input performance of pointing gesture systems, it is only available for applications utilizing the framework. There has to be a way to make use of the framework's input layer along with the pointing enhancement technologies it provides or existing applications without any modification or prior configuration.
With target localization for existing applications and therefore target aware pointing enhancement, it is possible to effortlessly acquire a target in any application. For actual target selection it is important, which options exist to trigger a click with device free pointing interaction in mid-air and which options are optimal choices.
With a functional, reliable pointing gesture recognition comes the desire to use it across displays as it is not bound to a single one. The transfer of interface elements across systems needs to be provided by the framework, but latencies can always occur. For those cases, it is important to know how to bridge the latencies to improve the overall

user experience.
The interaction across devices and displays requires usable interfaces on all displays. Besides the flexibility of the output of the framework to allow for such a variety, there needs to be a concept that allows a developer using the framework to choose element sizes for arbitrary interactive systems and to easily understand their implications on the interaction performance.

Contributions:

- An application framework that allows for rapid development of multi-modal and multi-display applications by providing a network transparent signal and slot system and bidirectional communication between input modalities and user interface, including its layout [vdCS13c].
- A novel force field variant for pointing enhancement, which solves the problem of overlaps for all interface layouts, eliminates the need to define a field strength and detects when fields are needed to dynamically activate them [vdCS13b].
- A computer vision based approach to locating interface elements without prior knowledge to apply target aware pointing enhancements to existing applications [vdCS13a].
- A taxonomy of hand gestures for triggering click events, which was used in an empirical study to systematically determine the best possible gesture to allow clicking in mid-air [vdCSS13a].
- A methodology to predict element sizes for arbitrary interactive systems to retain the interaction performance from a reference system to any target system [vdCSS13b].

1.4 Outline

This thesis is structured as follows. Chapter 2 discusses the state of the art and related work. This covers related frameworks and approaches to multi-modal interaction as well as pointing enhancement technologies, the integration thereof and the design of interfaces for a multitude of display devices and cross display interaction. Chapter 3 presents the architecture of glueTK and discusses the properties that allow for easy integration of new input devices, multi-display applications and raise the interface layout as a source of information to an equal level as the input data of input devices. Building upon the technical possibilities offered by the glueTK architecture, Chapter 4 describes several contributions to enhance the integration of novel input modalities with a special focus on pointing interaction. Chapter 5 describes two applications created with glueTK, one with a focus on multi-modality and a multi-display environment and the other with distance dependent display adaptation. Chapter 6 summarizes this thesis.

2 Related Work

Human computer interaction has a long history of research and the following sections will discuss work related to this thesis to give an insight into the current state of the art. As the glueTK framework is the necessary ground work for several advances in human computer interaction introduced in this thesis, Section 2.1 will present some of the frameworks used today as well as the progress made towards supporting novel input modalities and multiple displays in recent years.
While glueTK supports various data sources as input, a main focus of this thesis lies on pointing gesture interac-

tion as it is a versatile input modality in multi-display environments, especially if large displays are involved. To incorporate pointing gesture interaction in both existing and new applications, several challenges have to be addressed. Section 2.2 outlines work related to enhancing pointing interaction in general, as well as integrating these enhancements into existing applications. An additional challenge addressed, deals with progress made towards one of the core challenges of pointing interaction, intuitive clicking.
Inter-display interaction is not only about input, it is also about the displays and user interfaces. For large multi-display environments it is common that users can move freely between displays, which creates challenges as well as opportunities. Section 2.3 and Section 2.4 discuss the current state of the art of inter-display interaction and interactive displays respectively.
One key concept of glueTK is the equality of input and output. The display content and state are just as important for interaction as the input. For human computer interaction it is of key importance that the interface is adapted to the display hardware as well as to the user. Section 2.5 describes related work in the field of display adaptation and interaction performance.

2.1 Multi-modal Input and Multi-display Environments

A framework is software that provides generic functionality that can be customized by developers to create specific

applications. A key property that differentiates a framework from other software tools such as libraries, APIs or toolkits, is that the framework dictates the flow of the applications and not the developer [RG98]. While glueTK is a framework, there are many related projects that do not provide a framework but provide similar functionality in other ways. The following subsections will therefore give an overview of frameworks, libraries, APIs, toolkits and projects that are related to the functionality provided by glueTK. The first subsection deals with projects that focus on input alone. While all of these lack great parts of what makes up glueTK, they address similar problems with regard to the integration of multi-modal input. The second subsection addresses digital tables. While projects related to digital tables usually focus on few input devices and a single display, they produced some early related work that addresses input and output within the same context. Finally, the third subsection focuses on projects that put emphasis on the visual output.

2.1.1 Multi-modal Input

The need for applications that can handle novel input modalities dates back to the first multi-modal systems. Bolt's "Put that there" [Bol80] is one of the first examples of multi-modal input, in this case, speech and gestures. It used speech to issue a command and a pointing gesture to provide the intended spatial locations. One of the main problems with multiple input modalities was data fusion on different levels. Flippo et al. [FKM03] present a framework

for rapid development of multi-modal interfaces. The goal is the fusion of input data from different modalities and to pass the result on as input data as well. The framework is limited to only handling input data. A similar framework is MUDRA [HDS11]. Its goal is the fusion of multiple modalities in a generic fashion with a special focus on input data with different levels of abstraction.
Fusion has become an important research topic of itself; an overview of fusion engines for multi-modal input can be found in [LNP+09].
The OpenInterface Framework [SNL+08] has the goal of aiding the development of multi-modal applications. The focus is enabling the use of multi-modal input and also to provide a framework that makes applications reusable. This allows for an easy exchange of these applications as well as modules or components. An overview of typical problems and settings of multi-modal systems as well as an overview of frameworks along with their commonalities and differences can be found in [DLO09].
While these frameworks deal with multiple modalities, they do not include any means for output. The intelligent room project [BDB+97] is an example that shows that multi-modal input alone can create an intelligent environment. It is an early approach to allow for intuitive interaction with a room. It does not utilize any displays but offers speech recognition as well as intelligent services based on observations of user actions in the room. The integration of multiple modalities is accomplished by "agent-based layers", in which each agent offers an abstraction of the component it wraps, so the application developer does not have

to deal with it in detail. The goal of the QuickSet system [CJM+97] is to use novel input modalities, speech and gestures, to allow the control of a complex application on a small screen. Multiple QuickSet systems, each running on a PDA, can communicate using a blackboard based agent system. While very flexible, the focus is neither on a generic integration of modalities nor the creation of actual user interfaces.

2.1.2 Digital Tables

Digital tables are computing systems that use large displays as well as novel input modalities. While they do not deal with a large range of display sizes and modalities, they are a popular platform that differs significantly from the traditional desktop computer.
Such tabletop systems have gained much interest initially due to the possibility of placing objects on the screen that could be used for interaction. These so-called tangible objects close a gap between real world interaction and virtual screen content. ReacTIVision is a computer vision framework for detecting fiducial markers on a digital table to enable interaction with tangible objects [KB07].
TUIO [KBBC05] is a communication protocol, which was initially designed for tracking data of tangible objects but has become a de facto standard for multi-touch data as well. It is based on Open Sound Control (OSC) and allows the integration of multi-touch data into applications. This is necessary, as the default interfaces of most applications can only deal with mouse and keyboard data. Multi-touch

data, however, is richer and incorporates more information than a mouse. Therefore, just emulating a mouse would waste useful information. The protocol is specifically tailored towards multi-touch data all the way down to the format of messages, which limits its use for the integration of other modalities.
Echtler et al. [EK08] use a similar, message based approach for communication, but also add automatic geometric transformations to provide pixel level coordinates from touch recognition systems that do not provide these natively.
An important aspect of tabletop systems is the multi-touch input. Common approaches to detect these touches are usually based on computer vision, such as Touchlib [WZX11], which includes blob detection and tracking. Similar to Touchlib is "LightTracker", a computer vision toolkit for easy touch detection and tracking to build custom touch trackers, tailored towards a specific system [GLH10]. DiamondSpin is a system by Shen et al. [SVFR04], which goes beyond multi-touch by creating a tabletop system that is usable for multiple users at the same time. The goal is to allow users to collaboratively work at a single display. Much effort is spent on correctly transforming display content towards the respective users by rotating interface elements depending on the users' position at the table.
Bader et al. present a tabletop system that does not only use multi-touch but also incorporates a second modality, eye gaze, to demonstrate several interaction concepts [Bad11].

2.1.3 Visual Output

The tabletop systems described above were early adopters of novel input modalities for a limited space, but there were ambitions early on to make whole rooms interactive and also increase the number and size of displays. With multiple displays and multiple machines driving those displays arises the need for inter machine communication. There exist many middle ware solutions that allow for easy inter-machine communication and messaging, many of which have been compared in [DCE+11] and have been used to send data of input devices to a single display [FGA+07]. While these solutions allow for effortless communication, there is always a clear distinction between messages that are explicitly sent to remote machines and local functionality such as methods and functions. The event based middle ware "Hermes" brings remote messages and local methods conceptually closer by using a type- and attribute-based publish/subscribe model [PB02]. For the Java language exists a solution called RMI (Remote Method Invocation) [WRW96], which is extended by the Java-Party [PZ97] for distributed computing, such as clusters of workstations and makes remote methods accessible while reducing the programming overhead required for RMI. For all approaches, however, the developer needs to be aware of the locality of methods and decide which need to be accessed remotely.
Meaningful interaction in multi-display environments poses more challenges than inter-machine communication.
Johanson et al. [JFW02, JF02] describe their interactive workspace project as a project to investigate interaction

with large high resolution displays. Their prototype system, called iRoom, relies on a software framework called iROS. Their main focus is not on the integration of multiple input modalities or interface adaptations to the large displays, but on making existing WIMP applications usable at different displays. Due to multiple displays and systems driving them, a coordination model based on tuplespaces [GC92] is used for the integration of multiple applications. An "Event Heap" is used as a central entity to distribute events between all machines.
The "EasyLiving project" by Krumm et al. [KHM+00] is one of the few systems that incorporate not only pointing gesture modalities. While touch, as well as pointing gestures, are novel modalities that require custom tailored interfaces for best results, they can be forced to be used as mouse replacements, which is why non mouse-like modalities are usually harder to utilize for human computer interaction. They use the identity and location of users to display information, like instant messages, at the display closest to the user. Their main focus, however, is not the integration of tracking data in an interface or interaction framework, but the challenges with respect to the person tracking itself posed by their living room scenario.
Melchior et al. [MGVVR09] describe a toolkit with a special focus on distributing applications across multiple machines. While neither novel input modalities nor adaption to large displays play a role in this framework, they support the transfer of data as well as the transfer of parts of applications across multiple machines. They make use of traditional WIMP applications and corresponding input,

but different parts of the same application can be used at different displays and by different users at the same time. The ROSS API [WMJ+12] is a toolkit that allows communication between multiple devices to exchange input and output data. The main focus is a structure to organize input and output devices in a hierarchical order. This allows for making the input data of one device available at another system. Output, in the sense of creating user interfaces, visualization or visual adaption for modalities or displays, is not part of the framework. While there are many frameworks that deal with different aspects of multi-modal input or the utilization of multiple displays and machines, very few make the actual creation and rendering of interfaces part of the framework.

Most application interfaces today are implemented by utilizing a framework that follows the WIMP (Windows, Icons, Menus, Pointer) paradigm. There exist several such frameworks for most operating systems [Har99, War00, TL02]. While applications created with these frameworks are dominant today, they are specifically tailored towards the traditional desktop computer with keyboard and mouse as input. The limits of these frameworks become apparent when creating applications for large displays or novel input modalities.

A very flexible solution for large display visualization is the open source programming language Processing [Pro]. As it is a programming language, much functionality required for multi-modal and multi-display environments needs to be implemented manually. Communication support allows sending and receiving of bytes over a network, but func-

tionality built upon that is not part of the language.
Large displays are often intended for data visualization. Vvvv [Vvv] is a commercial framework for such visualizations. It also includes the management and control of multiple render nodes and supports the control of many displays, usually as a single surface. It has limited support for some non standard input devices. The creation of actual applications is, however, not the focus.
One of the few examples of frameworks that incorporate novel input technology and include dedicated output is PyMT [HHV+09] by Hansen et al. They describe how today's toolkits for interface creation are not well suited for creating interfaces for many novel input modalities. In particular, they try to build applications tailored towards the potentials of multi-touch as input. The integration of input handling and output handling allows to build interfaces that can exploit the potential of multi-touch data unlike many of the existing frameworks that internally map all pointing like data to mouse coordinates. While PyMT does control input and output, information about the interface is not globally accessible or used.
While many of the challenges involved with multi-modal and multi-display interaction have been addressed, input is always seen as independent of the output, which is why most frameworks do not include any means for output at all. Those which do, are designed to control the displays in a one-way fashion with no feedback from the output layer to the input layer. A similarly strict distinction is made between in-application and inter-application communication. While there are many approaches to spread applications

across multiple displays and machines, they are thought of as different applications instead of a single application, embracing a whole room of multiple displays.

2.2 Pointing Gesture Enhancements

The mouse is a very powerful input device for graphical user interfaces. The ability to quickly indicate any desired position on a screen allows for powerful interfaces, which is why many novel input modalities generate similar input data. Pointing gesture recognition systems, for example, generate locations on a screen but differ from a mouse in that they can be used freely in a room without the need for a flat surface or holding an actual device. The advantages of device and surface free use come at the cost of lower accuracy, however. This is true for most input modalities that rely on measuring body movements. While novel input devices will become more accurate, even perfect measurement will not result in perfect accuracy as users are not able to make perfectly accurate movements. The accuracy of an input device and the resulting ease or difficulty of hitting a target with respect to the target's size has been studied extensively, most notably by Fitts [Fit54] resulting in Fitts's law 2.1. Fitts's law quantifies the relationship between the size of and distance to a target and the difficulty of hitting the target.

The most important conclusion drawn from the law is the fact that the bigger a target is, the easier it is to hit. This insight is used to improve user interfaces for mouse input,

$$t = a + b \cdot \log_2(1 + \frac{2d}{w})$$

Figure 2.1: Fitts's law in the Shannon formulation. a and b are input device specific parameters, d is the distance to the target and w is the size (width) of the target. t is the movement time.

but is especially useful to counteract inaccuracies of many novel input modalities. Since simply increasing the size of targets restrains the interface design, pointing enhancements were developed to increase the virtual size of a target. The key idea is to influence the cursor of a pointing input in such a way that targets are easier to hit than Fitts's law would predict for their visual size. An example for such an enhancement are Area Cursors [KB95]. While most cursors have a single point (or pixel) "hot spot" which is actually used for target activation, area cursors have a larger spot. This means a target can be activated even if the "hot spot" (usually the tip of an arrow) is not exactly on a target. This way, the area available for target activation becomes larger than the visual area of the target, which results in faster acquisition.
Worden et al. [WWBH97] compare area cursors and so called "sticky icons" as well as their combined use. Mouse movements are usually not directly mapped to screen coordinates. Therefore, a small movement of the mouse can result in a large cursor movement. The transformation between mouse movement and cursor movement is usually

dynamic and depends on the speed. The faster the mouse is moved, the more "gain" is applied to the cursor movement. Sticky icons are designed to reduce this gain factor when the cursor is above the icon. This increases the effective size of the target, making it easier to stop the cursor on the desired icon.
Cockburn et al. [AC05] compared sticky icons to non-speech audio and tactile feedback in a user study. They found that while every feedback improves the interaction in terms of speed, sticky icons perform best. A disadvantage found are negative influences of sticky icons when there are interferences between neighboring targets.
An improvement of area cursors for target acquisition is the Bubblecursor [GB05]. Its functionality is depicted in Figure 2.2. The top left illustration shows the typical operation of an area cursor, with its much larger activation area. This can cause problems if the activation area of the cursor covers multiple targets, as it is unclear which target is the intended one (top right). The bubble cursor solves this problem by dynamically changing the size of the activation area depending on the surrounding targets as shown on the bottom left. The name stems from the bubbles it can create from the original area cursor if the cursor location and target alignment can not be accounted for by size adjustment alone, as depicted in the bottom right.

Area cursors have been studied and extended by other researchers as well. Findlater et al. [FJS10] introduce several additional functionalities based on area cursors. One is their "cross and click" technique, which transforms all icons in the area of the cursor to targets on a circle around

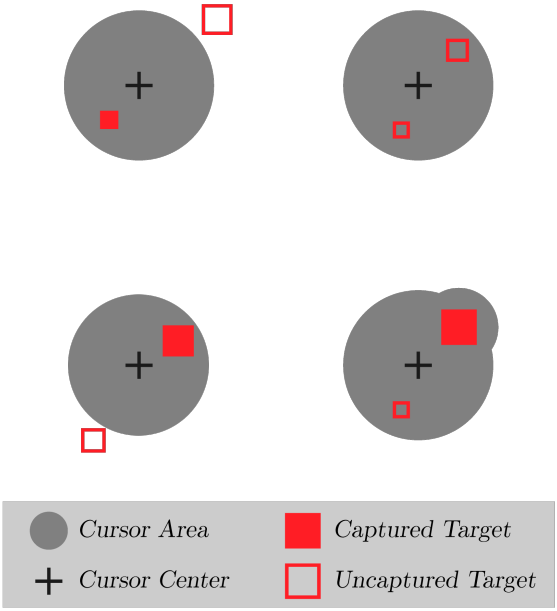


Figure 2.2: The Bubblecursor adapts its size and shape to the surrounding targets.

the location of the cursor, which just have to be passed to be selected. Additions like this can further enhance interaction beyond target acquisition, but do not only require knowledge about the interface layout but also means of manipulating it.
An extreme variant of utilizing the knowledge about the interface layout for pointing enhancement is "Objectpointing" by Guiard et al. [GBBL04]. The cursor is manipulated in such a way that it can no longer be freely navigated, but jumps to whichever target is closest to the cursor. This causes the cursor to jump from target to target as it is moved. This behavior results in very fast target acquisition times, but prevents several typical uses of a pointing input like freehand drawing or scaling. The authors suggest to understand Objectpointing as an optional mode rather than an alternative to the common "Bitmappointing". To make use of the advantages of Objectpointing, an additional toggle interaction is therefore required.
An enhancement technique called the "Beam cursor" [Yin06] is intended for a digital pen as input but could be adapted for most pointing devices. Figure 2.3 illustrates the idea. When the cursor comes close to a target (1 to 2), a beam is displayed from the target to the cursor (3). If the pen is lifted in this state, the target is selected without having to move all the way to the target (4). Whether the cursor is close to a target is determined by areas around the targets, calculated using a Voronoi diagram. While most enhancements aim at being invisible, the beam cursor adds a visible element to assist users. The target size, however, is just virtually increased as the target itself remains unmod-

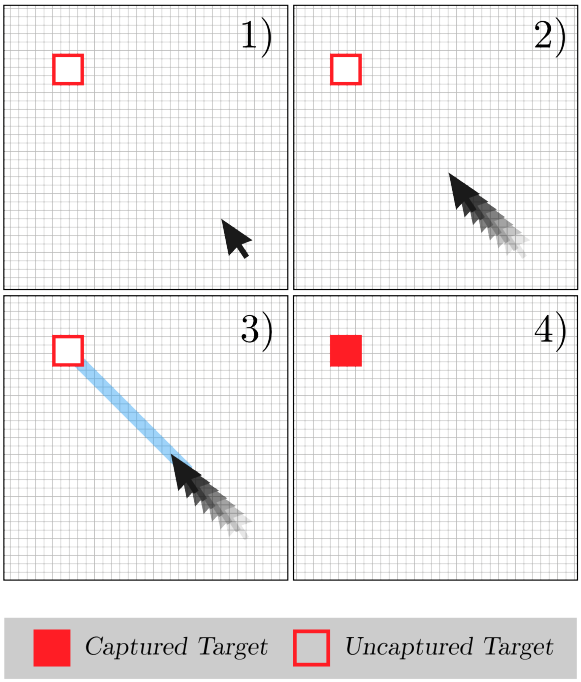


Figure 2.3: The Beam cursor allows to target activation before the cursor reaches the target.

ified. There are also approaches that actually increase the visual target size. One example are the visually expanding targets by Brock and Cockburn [Bro05, CB06]. This technique increases the size of targets as the cursor comes closer and decreases the size as the cursor moves further away. The technique is illustrated in Figure 2.4 and has

become well known due to its use in the Apple OSX operating system.

Figure 2.4: Visually expanding targets on the Apple OSX dock.

Another example of visual modification for pointing enhancement is the “Pointing Magnifier”
by Jansen et al. [JFW11]. The effect of this technique has to be manually triggered. Once activated, the area around the cursor is magnified and the user can interact within the scaled up area. Since the magnification and the size of the magnified area can be adjusted, this technique is particularly well suited for users with motor impairments.
Ahlström et al. [AHL06] present an enhancement technique called “force fields”. The key idea is that in a certain area around a target the cursor position is influenced by moving it towards the targets center. The offset applied to the original cursor position towards the target is called the strength of the force field. They evaluate force fields in two

user studies and find that they outperform sticky icons.
The original force fields have several limitations, one of which is lacking support for overlapping force fields. Specht et al. [SSG+10] use force fields with a joystick as an input device and solve the problem of overlapping fields for a limited number of cases. Using the movement direction of the cursor and a set of rules, they can decide in several cases which force field should be used. There are, however, cases that are not covered, in which case no force fields are used as a fall-back solution. The use of a joystick is one of the few examples where pointing enhancements are used for other input devices than the mouse. Another example is the application of force fields, cursor speed reduction and cursor warping to eye tracking by Zhang et al. [ZRZ08]. Eyetracking is, just like pointing gestures, an inherently inaccurate modality as the eye jitter can not be avoided and has to be counteracted for precise interaction. In a user study, they find that force fields and cursor speed reduction significantly improve the pointing performance.
The angle mouse [WFL+09] is an atypical example as it does not rely on information about the layout of the interface like all other techniques described above. The idea is to modify the cursor's gain depending on the angular deviation of the cursor's movement. If it is low, the gain is kept high, if it is high, the gain is dropped. This results in fast movements when the cursor is moved along a straight line and slow movements otherwise. While being target agnostic, the configuration is complex and does not aid users in all cases.

As the knowledge of the location and often also the size of targets in the user interface is key for many pointing enhancements, it is important to know how this knowledge can be obtained. This is especially difficult for existing applications with no access to the source code. In these cases there exist essentially two approaches. The first approach relies on operating system specific functions to reconstruct elements of the interface and their respective locations. This is limited to default elements and is not supported on all operating systems, but has been used to integrate a computer vision based touch system into existing applications [BYCH05] as well as for automatic software testing [MBN03].
The other approach to localizing interface elements is based on computer vision. The idea is to capture the screen content and locate targets in the resulting image. This approach is very versatile but requires additional processing power. Yeh et al. [YCM09, CYM10] developed a tool called "Sikuli", which can be used to automatically query documentation databases using screen shots of windows or dialog boxes as input. The computer vision based query relies on template matching and local features. A similar approach is used by Dixon et al. [DF10, DLF11, DFW12]. Unlike Yeh et al., their goal is to utilize the knowledge of the interface layout for target aware pointing enhancements. Their "Prefab" system uses a database of target templates to compare to the screen content. As this database has to be created manually, they suggest, however, to allow for an easy exchange of databases to minimize the need for manual database creation.

While pointing enhancements can significantly improve the speed and accuracy of target acquisition, to actually activate a target, a trigger - such as a click with a mouse - is required. For many novel input modalities this trigger has not yet been studied in detail, there is nothing ubiquitous like the button of a mouse for most modalities yet. In addition, many novel input devices are capable of different trigger gestures or movements, which can be used for secondary functions.
In the context of a touch based interface, Woobrock et al. [WMW09] conducted a study to find natural gestures for touch interaction. Twenty non-technical users were presented with the effect of a gesture and were subsequently asked to perform the effects' cause. More than 1000 gestures were collected this way for 27 commands. Interestingly, the number of fingers involved in the gestures had no effect on the preference, however, single handed gestures were more often used than gestures involving both hands. While all gestures of the study were limited to touch and, therefore, a two dimensional surface, Vogel et al.[VB05] used the touch of a finger tip and the thumbs tip to trigger a click event for distant hand pointing. A more recent work by Nancel et al. [NWP+11] evaluates several mid-air gestures for pan-and-zoom interaction. The remote pointing towards a large display wall seems to affect the users' preference for the category of gestures, as two handed gestures were preferred over single handed gestures. In addition, the results of their user study show that linear gestures were preferred over circular motion gestures.

Quek et al. [Que94, Que95] have studied computer vision based input modalities and present a first, basic taxonomy of gestures, which allows a general categorization of static gestures and motion gestures. A similarly generic taxonomy can be found as part of the elaborate taxonomy [GW07] of interactive table top systems, which also includes the display variations and physical properties of the table.

2.3 Inter-display Interaction

The previous section has described progress towards improving novel pointing technologies. Unlike a mouse, many of these are not bound to a single display. Gesture recognition is often independent of any display and the 3D pointing data has to be mapped to the surfaces of displays. With such a display independent input modality quickly arises the desire for cross display interaction. This does not only involve the mapping of the input data to different screens but also the handling of content on multiple displays, especially the transfer of interface elements beyond the boundaries of a single system.
Bragdon et al. [BDHM10] use gesture interaction in an office environment to allow collaboration between users, utilizing a large, shared display as well as mobile devices. Their focus is on the social acceptability of gestures in such a public context.
Dynamo [IBR+03] is a public interactive surface for cooperative sharing as well as the exchange of media. Instead

of users exchanging data from their personal devices, they do so in this collaborative workspace, which makes the distribution of files and users easy to understand. These examples show that cross display interaction is an active field of research, and easy and intuitive transfer of items across screens has several applications. Cross display and especially cross system transfer of data always comes with technical limitations, which most of the time present themselves as latencies. If latencies can not be avoided, it is important to find ways to mitigate their effects.
Bederson et al. [BB99] have found that animations improve the users' ability to remember spatial relations of interface elements. Stasko et al. [SBL93] have studied the benefit of using animations to aid students in their understanding of algorithms by smoothly animating transitions between different states. While these studies focused on using animated transitions to aid users in a specific task, animations can be used in many places of user interfaces. Baecker et al. [BS90] give a summary of many of the ways in which objects can be animated in user interfaces.

While most animations in today's operating systems are rather subtle, Chang et al. [CU95] argue that the use of more elaborate animations would aid in the understanding of many interactions. They use cartoons as a base for comparison, which are easy to understand due to complex animations and suggest the use of such animations in user interfaces as well. An example for such an elaborate animation is the so called "Genieeffect" in Apple's OSX, used when minimizing a window as illustrated in Figure 2.5.

Figure 2.5: The Genieeffect applied when minimizing windows to aid the users spatial memory.

2.4 Proxemic Interaction

When it comes to utilizing novel input modalities, person tracking is especially interesting for implicit interaction. The location of users relative to a display can be used to realize many interesting applications. Brignull et al. [BR03] have studied the use of displays in public spaces and discovered that fear of social embarrassment keeps many users from interacting with such a display. By using location information, a fluid change between users interacting with one another and users interacting with the display can be accomplished, which they suggest.

Marquardt et al. [MDMBG11, MBB+12] present an interactive system for the exchange of digital content by utilizing proximity information. Both visibility of content and transfer are affected by the location of devices relative to each other. They utilize fine-grained measures of proximity to trigger different functionality instead of a binary information about the presence of a person.
In addition to location, the orientation of persons is used to control an interactive media player in [BMG10]. If two persons in front of a TV look at each other, for example, the media playback is paused. As soon as they look at the TV again, playback is continued. In this application there are defined actions for defined locations or orientations instead of a direct mapping of the location to an adaptation of the interface.
With their "Range" white board, Ju et al. [JLK08] explore the switch between implicit and explicit interaction. They use the distance to the white board as a cue to when a person actually wants to interact with it instead of simply reading its content. They utilize Hall's distance zones [Hal90] to define different levels of privacy at different distances.
A similar concept of different zones is implemented in the "Hello.Wall" project by Prante et al. [PRS+03].
The Hello.Wall is considered informative art as it emits information using different light patterns. The distance is detected using RFID transponders and automatically switches from an "ambient zone" to a "notification zone" when a person comes closer to the wall. Right in front of the wall it switches to the "interaction zone", where users

do not only receive information by means of individual light patterns but can also interact with the light emitting cells of the wall.
A similar concept has been realized by Vogel et al. [VB04]. Instead of RFID transponders, they used marker based person tracking for localization and added a range of options for interacting with a display by means of several hand gestures. As an example application a calendar is used which can not only switch between different overview and detail modes but also displays personalized information depending on who stands in front of the display.

2.5 Interaction Performance

Interaction performance can be divided into perceptual performance and input performance. Perceptual performance means the time it takes to visually acquire a target and the input performance describes the time it takes to select a visually located target by means of an input device. The most essential work on input performance is Fitts's law [Fit54], which has been described above.
MacKenzie et al. [Mac92] offer a detailed study of Fitts's law, along with a comparison of multiple variations and studies of the law that have been published over
time [CEB87, Dru75, JM85, KE88]. Their conclusion is that Fitts's law does not allow for perfect predictions, but is a very usable tool in human computer interaction research. To make it easily usable they created a tool [MB93] for the rapid evaluation of input devices using Fitts's law.

The tool also provides predictions by the several variations of the law so it is easy to see which predicts the devices performance best.
Fitts's law, in its original form, always assumes direct movements from the current position to the desired target. For certain tasks in a user interface, like menu navigation, this assumption is false. Accot et al. [AZ97] describe an adaptation of the law, which improves predictions for steering based tasks and is now known as the steering law. It has been extended later on [AZ01] to study how different sizes affect the steering law tasks. Not only the size of interface elements can affect the input performance but also different display sizes. Kostakos et al. [KO08] study these effects and derive generic rules for the adaptation of cursor speeds for displays of different sizes.
Fitts's law is not only used to analyze input devices, but also to evaluate the layout of interfaces. Sears et al. [Sea93] studied the use of existing interfaces to find the most common operations. They then suggested layout modifications which would optimize the size and distances of interface elements in favor of the most common operations, therefore improving the overall input performance. CogTool [JPSK04] is a tool that allows to create mock ups, based on web technology, of interfaces and analyzes the resulting input performance. The interface can then be adjusted and multiple variants can be compared. Automatically optimizing the layout of user interfaces is usually not possible as it requires context information that has to be provided by a user.
While input performance has been studied in detail, there

is little work on perceptual performance. One of the few is a system by Gajos et al. [GWW07] which shows how they can adapt traditional desktop user interfaces for users with motor and vision impairments. While the interface is automatically adjusted to the motor abilities, the adjustments for the visual impairments have to be made manually by the users.

While common, usable font sizes for desktop computers are well known, new devices always raise the question of suitable sizes for optimal legibility again. A study of multiple age groups in [DGBG05] determines an optimal font size for the small screens of personal digital assistants.

Today, graphical interfaces are dominant and the percentage of text elements decreases. Icons, or graphical elements in general, can be of great benefit for the human machine interaction [Byr93] but it is hard to classify graphical elements or determine the factors that make them easier or harder to recognize [MCdB99]. So unlike text for which ISO standards [ISO11, ISO08] exist for evaluation, scaling graphical elements to achieve optimal perceptual performance is a problem not yet solved.

3 Framework

This chapter will describe the glueTK framework in detail and how the challenges posed by interactive, multi-display applications are addressed. These challenges fall into three categories. First of all, the large number of input devices which vary in their interfaces as well as in the kind of data they provide. Different preprocessing has to be applied, and the actual information needs to be made available in a generic fashion. Second, when interacting across multiple screens, this often involves multiple machines that drive the displays as well. This requires communication across applications, displays and machines. Third, displays

vary greatly in physical size, resolution and aspect ratio, which calls for flexibility to adapt graphical user interfaces to these different configurations.
After an initial overview, every component will be described in detail. The focus will be on the framework itself and its internals. Functionality enabled by, and build upon glueTK is discussed in Chapter 4 and applications created using glueTK are described in Chapter 5. In addition to the architecture, this chapter also includes a performance evaluation of the framework as well as a user study conducted with developers previously unfamiliar with glueTK.

3.1 Architecture

Both technically as well as conceptually, glueTK is split into two separate parts: glueInput and glueOutput. The input layer, glueInput, takes care of abstracting from input device specific interfaces as well as preprocessing and adaptation to present the developer with a single coherent interface, independent of the modality and specific device. The output layer, glueOutput, allows for the creation of user interfaces tailored towards a wide variety of displays from smart phones to large video walls, connected to one or multiple machines. While it is optimal to create an application using both glueInput and glueOutput, in many cases this is not a feasible option, because applications already exist and rewriting or porting them is not reasonable when the only goal is the integration of a novel input modality. In these cases it is possible to use glueInput independently to

offer developers an easy interface to new modalities without any limitations. This approach is taken to an extreme in Section 4.2, where glueInput is used with a closed source application without its knowledge.

3.1.1 Overview

To get an initial overview and to show how the components that make up glueTK work together, Figure 3.2 shows the framework's architecture along with a few sample input devices. At this point, the intention is to illustrate the relationship in-between components, while their inner workings will be described in the following sections. The communication between all the components of glueTK is accomplished by a network-transparent signal and slot system. It allows to easily connect a signal of one component to a slot of another component even across different machines, which contributes to the flexibility of the framework.
The input devices (in light gray) are connected using different, device specific interfaces. The *FaceID* component provides information about a person's identity as well as details about the identification progress and sends its information over the network. The *Persontracker* sends coordinates of the locations of multiple persons in a room via the network. *Pointinggesture* is a system for articulated body tracking that provides the 3D pointing direction of stretched out arms of persons. While all three devices are connected via the network, the format as well as the type of data they provide differs greatly. Finally the *Gyromouse* is a hardware device similar to a regular mouse but usable

in mid-air and is connected via a USB-port as a human interface device (HID)[1].
For each input device, there is a corresponding event handler (in yellow). While each event handler is customized to the input device at hand, they have the common goal of abstracting from the device specific interface as well as data and can also add functionality if needed. At this point, any data input is converted to signals that are passed to the event manager so that the different interfaces are abstracted from at the earliest possible step (signal connections are indicated by solid arrows, all other communication is indicated by dashed arrows).
The event manager keeps track of all event handlers but also deals exclusively with all network connections. This has the advantage, that if multiple handlers need the same data stream, it is only requested once by the event manager and then distributed among those subscribed to it. In addition to event handlers, context handlers can be registered with the event manager. These subscribe to signals to generate additional, high level information. The 3D pointing information, generated by the *Pointinggesture* is converted to pixel coordinates this way, for example.
Connected to the event manager is the signal manager. To avoid deep interweaving of the two layers, a single, clean interface between the layers is defined here and makes the separation of glueInput for independent use possible. The signal manager keeps track of signal connections using a mapping table and passes incoming signals on to subscribed widgets and also provides information about the state and

[1] http://www.usb.org/developers/hidpage

layout of the user interface back to the input layer. In addition, it takes care of rendering the interface by displaying all registered widgets. Widgets are interface elements created from building blocks provided by glueTK. A widget can offer slots that can be connected to any signal, be it for communication with other widgets, e.g. a button-click triggering the display of an image, or for reacting to data from input devices, e.g. moving the widget to the display position provided by the *Gyromouse*, to create a cursor.

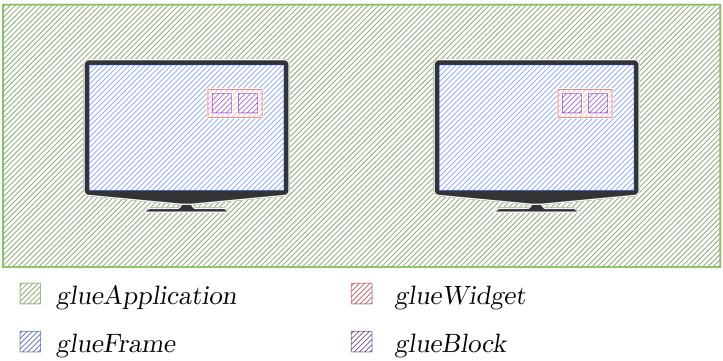


Figure 3.1: Illustration of the terminology for the different parts of a glueTK application.

Because glueTK allows the creation of applications involving multiple displays that can be connected to multiple machines, Figure 3.1 illustrates the terminology of the different parts of an application created with glueTK. The application as a whole is called a glueApplication. The interface on a single display which corresponds to an ap-

plication in the traditional, technical sense is called a glueFrame.[2] The glueFrame itself contains the user interface which is made up of glueWidgets, which are in turn built from elementary blocks, called glueBlocks.

3.1.2 GlueInput

The goal of the input layer is the abstraction from device specific interfaces and to provide the developer with a single coherent interface based on signals. In addition, the input layer can provide auxiliary functionality and improve input devices by utilizing context knowledge about the state and layout of the user interface.

Event Manager

The central part of the input layer is the event manager. Unlike the event handlers, it is not responsible for a single input device, but for making the information of all input devices available, as well as providing them with context information. As the central communication hub within a glueFrame, it also represents the interface to other glueFrames within the same glueApplication by passing incoming signals from remote glueFrames to the local glueFrame and vise versa. As many event handlers, as well as context handlers, require the same information or access to the same resources, the event manager takes care of handling

[2] Note that multiple displays connected to a single machine and configured as a single desktop are also handled as a glueFrame.

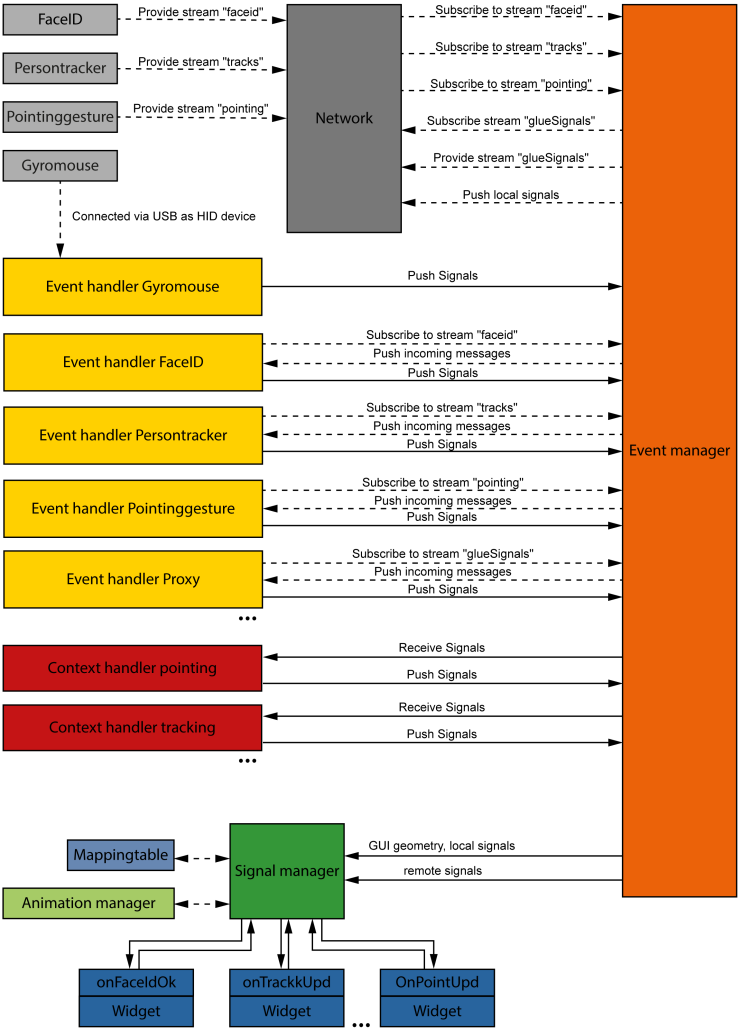


Figure 3.2: glueTK architecture overview with exemplary input devices and corresponding event handlers.

access to information and resources, as this avoids duplicate requests and reduces communication overhead. One example of information that many event handlers need and that is handled and distributed by the event manager is the context information of the user interface, which includes properties of interface elements such as their position, size and state.
This information is provided to the event manager by the signal manager via the single interface between the input layer and the output layer. This strict separation and clear interface definition allows the input layer to be used independently of the output layer by allowing developers to use the same interface to the event manager as the signal manager. Context information can be provided manually, if the input layer is used in conjunction with an existing application that does not automatically provide this information as glueApplications do.

Eventhandler

The support of many, diverse input modalities requires the support of a large variety of interfaces as there are not only custom interfaces and device specific SDKs but also a variety of standards like HID[3], TUIO[4] and MPX[5]. The range of supported devices include both devices directly connected via USB, either requiring special drivers or emulating a mouse, and systems running on different machines,

[3] http://www.usb.org/developers/hidpage
[4] http://www.tuio.org
[5] http://www.x.org/releases/X11R7.6/doc/inputproto/XI2proto.txt

providing input data via a network connection. The variety of interfaces is difficult to handle and makes the use of multiple devices of different modality cumbersome for the developer. This is why glueTK abstracts from them at the earliest possible point using event handlers. Many established input devices only need to be wrapped by an event handler, so that their input data can be accessed in a generic fashion. There are, however, many new input devices that can be improved in the event handler.
To illustrate how input devices can be improved in the event handler, consider a pointing gesture recognition system as an input device. Since pointing gestures are usually recognized by computer vision based systems, the results are often imprecise due to sensor noise and measurement errors. Filtering can be applied, but also reduces the responsiveness of the cursor. To avoid this, the filtering can be applied in a context sensitive fashion by taking the layout of the user interface into account. By only applying filtering around targets, the user gets assistance with controlling the cursor when needed but direct and fast responses everywhere else. This concept is also applied to make use of pointing enhancements (see Section 4.1) for more advanced improvements relying on context information.
Besides improving input devices in the event handler, it is also possible to add new functionality. The *Pointinggesture* in the architecture overview (Figure 3.2) only provides a location on the screen and therefore allows users to acquire a target. To actually select or click a target, additional functionality is required. A popular way of implementing a selection trigger for a pointing gesture is a dwell timer.

The idea of the dwell timer is to trigger a click if the user keeps the cursor at the same location for a certain amount of time. Usually, users are provided with visual feedback about the click progress as illustrated in Figure 3.3. Such a dwell timer could be implemented as part of the gesture recognition system. As it has no knowledge of the interface layout, however, a click would be triggered whenever the cursor is kept stationary, even if the cursor is not on a click-able element such as a button. The visual feedback going along with the click progress can be very irritating to users if they did not intend to trigger a click. By implementing the dwell timer based click functionality in an event handler, this can be avoided. As event handlers have knowledge of the current layout of the interface, the dwell timer is only started if the cursor is within reach of a click-able element, creating a much more robust user experience.

The proxy event handler is a special event handler and

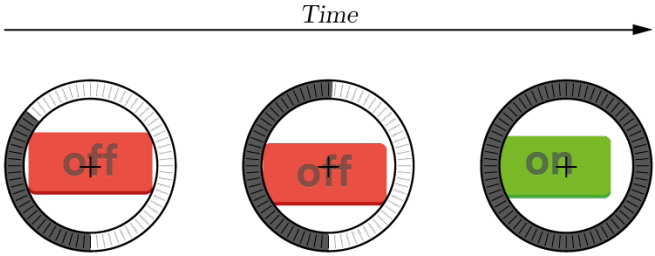


Figure 3.3: Visual feedback for a dwell timer implemented within an event handler. As long as the cursor stays within a certain deviation from the button's center, the progress is increased. After the delay, the click is triggered.

part of every glueFrame. Unlike all other event handlers, it does not abstract from a single input device to provide its data as signals to the framework, it rather distributes signals across multiple glueFrames. All signals generated within a frame, no matter if originating from an interface element or an input device, are sent to other glueFrames of the same glueApplication by the proxy event handler. In turn, the proxy event handlers of other glueFrames receive these signals via the network and make them available as signals to the local glueFrame. This makes any signal generated at any point within the glueApplication available to all glueFrames and allows to completely ignore the fact that interface elements and even input data might be distributed across machines. Because there is no difference between a locally connected input device and an input device connected to a different machine from the developer's point of view, it is possible to develop applications with a multi-display environment in mind without taking the underlying hardware into account.

Contexthandler

In glueTK, one of the main responsibilities of event handlers is the abstraction from the specific interfaces of input devices by translating all input data to signals and making them available throughout the framework. In many cases, however, the raw input provided by novel input modalities, especially those that do not provide mouse like data, is of little use for an application developer. To abstract from the rather low level information provided by many input

devices, context handlers subscribe to the signals provided by event handlers to generate higher level information that can be utilized directly. The additional layer of abstraction is optional and only applied when necessary, but avoids the need for additional logic in the output layer, which would interfere with the clear separation of input and output layer.
There are two designated methods to provide higher level information from low level data: using additional information or fusion of multiple low level data inputs. An example of additional information is the room layout in conjunction with the *Persontracker*. Using the coordinates of locations of persons in the room, the layout information can be used to derive spatial relations between persons and displays. The context handler can then provide signals in the style of "`person_x_left_display_y`" or "`person_y_at_display_z`". Such signals can then be used in applications without further analysis. The spatial relationship signals between persons and displays, for example, are used to allow a personal workspace to automatically follow its owner across displays, as described in Section 5.1.
Fusion of multiple homogeneous input devices is possible within a context handler but not described in detail here, as there are already many existing approaches [FKM03, HDS11, LNP+09]. Fusion of orthogonal input data as provided by *Persontracker* and *Pointinggesture* within a context handler, however, has some interesting applications. A common problem with the automatically following workspace mentioned above is that it is hard to select items within the workspace with a pointing gesture when it is moved by the

person tracker at the same time. By creating a context handler that subscribes to the signals of both input devices, it is possible to create a new signal for controlling the position of the workspace that automatically locks in place if the user points at the workspace and therefore making it easy to interact with it. Another context handler, subscribing to the same data, can be used to differentiate between multiple possible interpretations for touch input as illustrated in Figure 3.4.
Another very common use for context handlers is the conversion between 3D room coordinates to pixel coordinates to allow interaction with the user interface. The management of the required calibration data and coordinate system transformations, however, are residing in the geometry module.

Geometry

One of the goals of glueTK is to support any input device, but especially novel input technologies. Unlike a traditional mouse, which is bound to the machine and display it is connected to, many new input devices, such as the Kinect [Kin] and the Leap Motion [Lea], are display independent. They output data relative to the hardware of the input device, not relative to any display. In addition, glueTK supports multiple displays, connected to different machines so that display independent modalities can be used to interact with a user interface on any one of those displays. To put such input data in relation to the displays, glueTK needs information about the location of input devices and displays.

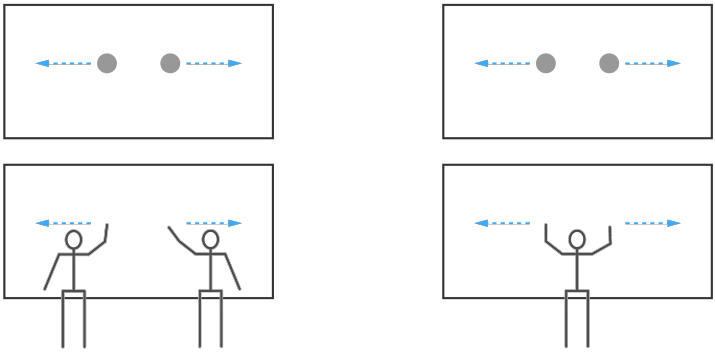


Figure 3.4: Differentiation between a multi-point gesture by a single person and two single point gestures by different persons by fusing *Pointinggesture* data and *Persontracker* data in a context handler. In both cases, the data received from the *Pointinggesture* alone is the same (top). The cause, however, is a different one (bottom).

These locations can be configured by initializing a glueFrame or an input device with the origin and rotation vector of their coordinate system. For displays, the x, y-plane is assumed to be the display's surface. This configuration only has to be done once for each glueFrame and each input device. GlueTK will then automatically calculate the corresponding rotation and transformation matrices for each combination of input device and display and distribute this information among all glueFrames using the signal and slot system. This way, there is no need for a global configuration, each glueFrame and each input device just provides their own location and this information will be made avail-

able throughout the glueApplication.
The availability of this location information already allows some new functionality. The higher level information generated from *Persontracker* data in a context handler described above is one example. Many new input devices try to replace the mouse and therefore generate pointing data. For several devices this means generating a pointing direction within their own coordinate system. But in order to use this input data for interaction it needs to be converted into pixel coordinates of a display. Using the generated transformation matrices, the pointing information can automatically be transformed from the input device's coordinate system to any display coordinate system. In addition, intersection points are calculated and if any are found, pixel coordinates relative to the intersected display are generated and sent out as as signals.

3.1.3 Communication

The fact that glueTK is built in a modular fashion and allows the creation of applications that run across multiple machines makes communication within the framework a crucial topic. The mechanism used for communication extends the signal and slot concept [WG00] by being network-transparent. The following sections explain how this mechanism allows the use of input data at any point in a glueApplication and how communication between glueWidgets within a glueFrame as well as across glueFrames is accomplished.

Signals and Slots

The signal and slot concept is a communication mechanism that allows to connect components in a flexible fashion, while keeping the components itself autonomous and modular. Any component providing information emits the information via a signal and any component requiring data offers a slot. For a signal to reach a specific slot, the two need to be connected explicitly. Any signal can be connected to any slot, assuming they use the same signature. This allows to replace components at any time and can create a completely different behavior by simply updating connections.
Traditionally, this concept is used in object oriented programming to allow for flexible communication between objects of the same application. GlueTK extends this concept by automatically distributing local signals over the network in every glueFrame. As all other glueFrames receive these signals, the ID of each glueFrame is attached to the signals before they are sent over the network as messages using a middle-ware [VvdCI+13]. This way, all receiving glueFrames can distinguish true remote signals from the ones they sent out themselves to avoid an infinite signal duplication. The true remote signals are then converted into local signals and made available via the event manager. Because a glueApplication can be comprised of many glueFrames, the way signals and slots are connected differs from the way it is handled in previous implementations, for example Qt [Qt2] or Boost [boo]. In these implementations, a signal of one object is connected to the slot of another object as

follows:

```
connect(Object1, Signal1, Object2, Slot2)
```

To incorporate the fact that objects in glueTK can reside within different glueFrames, the connect command is extended:

```
connect(glueFrame1, glueWidget1, Signal1,
        glueFrame2, glueWidget2, Slot2)
```

The above command connects the signal of one glueWidget, residing on one glueFrame, to the slot of another glueWidget, residing on a different glueFrame. Connecting glueWidgets within the same glueFrame works the same way:

```
connect(glueFrame1, glueWidget1, Signal1,
        glueFrame1, glueWidget2, Slot2)
```

In case the glueWidgets have globally unique identifiers, the explicit declaration of the respective glueFrame can be omitted. It is important to note that all identifiers in the connect command are strings. This means that it is not necessary to have access to the objects in code when connecting them. This way, it is possible to connect a signal and a slot of glueWidgets that reside on one machine by executing a connect command on another machine.
In a lot of cases, many connections are required and it would be tedious to set them up individually. To overcome this problem, glueTK uses the asterisk (`*`) as a wildcard character within the connect command:

```
connect(glueFrame1, glueWidget1, Signal1,
        *, *, hide())
```

All interface elements in glueTK have several default slots to allow for basic, common manipulations. One of these slots is the `hide()` slot, which makes an interface element invisible until the `show()` slot is called. The above command connects a single signal to the respective `hide()` slot of every interface element in every glueFrame throughout the glueApplication. With this connection, a single signal will trigger the hiding of all interface elements. This avoids even having to know about all existing interface elements and having to connect every single element individually.
Communication within glueTK is always based on signals and slots, and the network-transparency of this particular extension allows an implementation that abstracts from the fact that different parts of the application run on different machines.

Data Transfer

The network-transparent signal and slot system enables communication across machines without having to worry about underlying technical details. The same should be true for data transfer, more specifically, the transfer of interface elements across screens. A cursor, for example, is at its core a simple image displayed on the screen. With modalities that are not bound to a single display, the cursor image needs to move across displays, even if these displays may be connected to different machines. Having to deal

with multiple machines makes the movement of a cursor much more complex than in the case of a traditional, single machine application. The same is true for drag-and-drop operations, which the user expects to be possible across screens as he or she is presented with a coherent application across displays. Since this is, just like communication, a key part of glueApplications, it should be trivial for users but also developers to interact across devices. Therefore, transparent data transfer is required.
Unlike signals, which are usually just a few bytes in size, interface elements can be significantly larger because they often require binary data like images or videos for rendering. This prohibits the use of signals for transferring data as they are designed for high performance and low latency. Interface elements are therefore transferred using a separate mechanism. The interface element is serialized to JSON data, with binary data attached as encoded strings. This data is then sent over the network and deserialized to create an identical copy at the target glueFrame. A signal, triggered upon successful transfer, is used to remove the original interface element, therefore creating the impression of a move, rather than a copy operation. When transferring interface elements, especially while hiding the technical complexities in the background from the user, latency is an important issue. The JSON format has been used over XML as it has shown better performance [NPRI09]. Compression does not reduce latencies reliably as the time it takes to compress the data often outweighs the time saved on transferring less data. Section 4.4 discusses this problem in more detail and evaluates a possible solution. If

minimal latency is of paramount importance, a synchronization flag can be set for an interface element. This will cause any changes made to this element to be constantly synchronized across glueFrames. This way, each glueFrame keeps an invisible, always up to date copy of the interface element and latencies for transferring it are eliminated. As this can cause much data transfer in the background, it is not activated by default for all elements, as it would quickly exceed network limits and is seldomly required.
The transfer of an interface element across glueFrames is triggered by setting its position. Either, by explicitly setting its position within a target glueFrame (Figure 3.5):

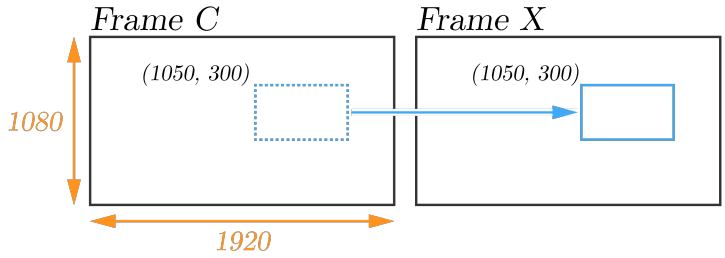


Figure 3.5: Element transfer by explicitly defining the target glueFrame.

```
element.setPosition("FrameX", 1050, 300)
```

Or it is transferred implicitly, by setting its position beyond the boundaries of the current glueFrame (Figure 3.6),

```
element.setPosition(2970, 300)
```

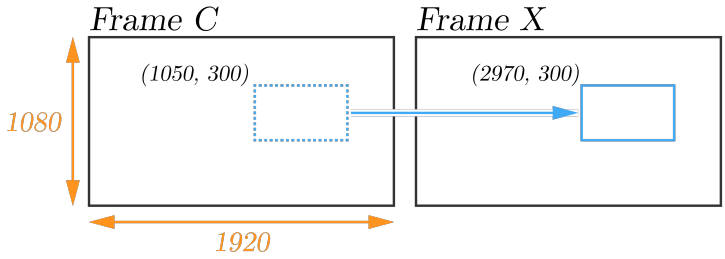


Figure 3.6: Element transfer by implicitly defining the target glueFrame.

given that another glueFrame is next to the current one. The relative position of glueFrames is known to glueTK, as each glueFrame is configured with its origin and rotation as described above.

Signalmanager

The signal manager connects the input layer to the application content. This can be either an existing application that uses glueInput to include a new input device, or a glueFrame within the context of glueOutput. All signals from input devices or remote glueFrames are passed from the event manager to the signal manager. A mapping table, that keeps track of all connected signals and slots, is used to pass on incoming signals to the connected slots. In addition, the signal manager gathers layout information about all interface elements including position, size, rotation and opacity, and passes this information to the event manager,

which in turn makes it available to the event handlers that can use this context information to improve the usability of input devices.
When using glueInput with an existing application, not created using glueTK, this information can not be gathered automatically, but developers can provide this information to glueInput manually to make use of the same enhancements. If glueInput is used in conjunction with glueOutput, the signal manager also takes care of driving displays as well as rendering interface elements.

3.1.4 GlueOutput

GlueOutput provides what is necessary to create and render user interfaces for a large variety of displays. Despite the fact that different input modalities have different requirements towards the user interface and displays with different sizes and resolutions put different restraints on interface creation, the goal of the output layer of glueTK is to provide an easy way to quickly build usable interfaces for any input modality and any display. Section 4.3 shows an example of how this quick interface creation can be used for prototyping, and Section 4.5 presents a solution to the problem of creating interfaces for target displays with differing properties. To allow this flexible creation of interfaces, glueTK provides elementary building blocks that can be used to create more complex widgets. This requires additional work, but puts no restraints on the creation of interfaces and allows for user interfaces specifically tailored towards the input devices and displays at hand.

GlueBlocks

Many existing GUI toolkits offer so called widgets, user interface elements with a clearly defined functionality. The advantage of offering these widgets is twofold. First, it avoids reinventing common interactions, because developers can simply use existing widgets to achieve the required functionality. Second, it makes it easier for users to get used to new applications - created with the same toolkit - as the application is new but many of the interactions

are well known because of the reuse of widgets. In essence, users can carry skills acquired at a standardized interface from one application to another. This reuse is possible because most established toolkits or frameworks are focused on creating applications according to the WIMP paradigm, with a well known range of display sizes and no other input devices but mouse and keyboard.
For glueTK, which supports a wide range of input modalities and displays, it is not possible to create a set of widgets that are guaranteed to work with any input device and display. Any such attempt would result in a compromise that would not be able to take full advantage of the specific properties of each modality and display. The approach of glueTK is, therefore, to offer elementary building blocks that are customizable and combinable. Some of the building blocks provided by glueTK are: image, animated image, video, text, container, map and web-view. All of these glueBlocks have default properties such as position, size, rotation and opacity, which can be used for customization and are also accessible via default slots. By combining several glueBlocks, arbitrarily complex glueWidgets, tailored towards a specific input modality or even input device and display, can be created.

GlueWidgets

GlueWidgets are compositions of one or more glueBlocks. The glueWidget as a whole has the same default properties and slots as every glueBlock, which allows to treat the glueWidget as a single interface element. In addition to

the default slots, a glueWidget can define additional slots to allow modular access to its functionality. A simple example of a glueWidget is a toggle button which allows the user to toggle between two states, on and off, for example. This kind of glueWidget can be created from an image glueBlock, a text glueBlock and a container glueBlock to hold the other glueBlocks. Figure 3.7 illustrates this composi-

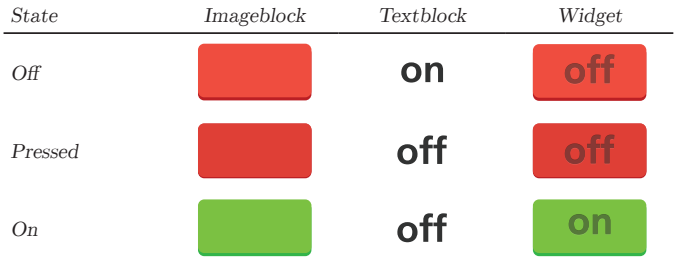


Figure 3.7: A toggle button glueWidget created from glueBlocks for pointing input.

tion. The image glueBlock switches the associated texture depending on the current state and the text glueBlock updates itself as well. By emitting a signal with every state change, this glueWidget could easily be used in any application. Note that the container glueBlock is invisible as it has only organizational purposes. Figure 3.8 also depicts a toggle button. Unlike the previous button, which is meant to be used with a pointing device, this button can be toggled using a keyword via speech recognition. It needs a slot that will be connected to the corresponding signals of

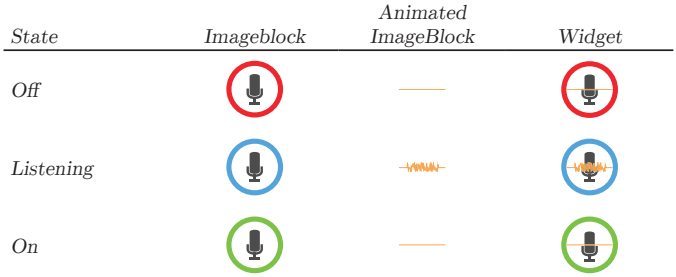


Figure 3.8: A toggle button glueWidget created from glueBlocks for speech input.

a speech recognition event handler and uses an image glueBlock and an animated image glueBlock, again, contained within a container glueBlock. These examples show that the reuse of glueWidgets tailored towards specific modalities is limited when it comes to different modalities, but no problem across applications.

Animationmanager

Animations are not crucial to the core functionality of glueTK, but animations are an important aspect of user interfaces as they can reduce confusions, give helpful feedback and enhance the overall user experience. Especially for latency prone operations, animations can be used to alter the perception of latencies as described in Section 4.4.
GlueTK does therefore offer a convenient way of applying

animations. All changes to default properties can optionally be animated using a variety of easing modes. An easing mode is a mathematical function that describes the rate of change of a parameter over time (see Figure 3.9 for two examples). In addition, glueTK allows to build custom animations by concatenating any number of these animations with arbitrary, individual easing modes and durations.

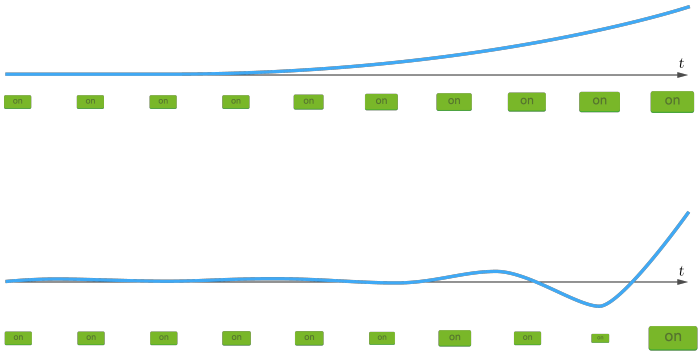


Figure 3.9: Two examples of a button, being scaled to 200% of its original size. The animation on top uses a "Quad" easing curve, while the bottom one uses an "Elastic" easing curve.

3.2 Evaluation

Many aspects of the glueTK framework are unique compared to existing frameworks, while other parts are comparable. The novel and unique parts are what differentiate

glueTK from previous frameworks and enable such a variety of new functionality that it is hard to compare it to existing frameworks, even more so for an evaluation. Instead, this evaluation focuses on two central questions:

- Is the performance sufficient to deal with several novel input devices and multiple displays at the same time ?
- Is the framework usable from a developers point of view ?

The next two sections describe the results of an evaluation and a user study with regard to these questions.

3.2.1 Performance

When it comes to the performance of the framework, the key factor is communication. Everything that happens within the framework relies on the network-transparent signal and slot system. The only other significant part is the rendering and display of user interfaces. However, glueTK utilizes an existing library for rendering [Qt2], which supersedes the need for evaluation within the context of this work.
There are four kinds of signal and slot connections:

- From an event handler to a glueWidget within the same glueFrame

- From one glueWidget to another within the same glueFrame
- From an event handler to a remote glueFrame
- From a glueWidget in one glueFrame to a glueWidget in another glueFrame

Technically, however, the only difference between these is whether the signal has to be serialized or not. Therefore, the evaluation distinguishes between serialized signals that have to be sent over the network as *remote signals* and signals that do not leave the glueFrame as *local signals*. Signals can vary in size depending on their content. For input devices and typical inter-glueWidget communication, the size of signals is rather small. For evaluation, a typically sized signal of 342 characters ($\sim$*0.33kbyte*) and for comparison a 76480 character signal ($\sim$*74kbyte*) were used. Remote signals were sent between two machines in a switched gigabit ethernet network. 100000 local and remote signals were transferred respectively, which was repeated ten times. The time from emitting the signal to triggering the connected slot was measured in all cases. Table 3.1 shows the average number of signals transfered per second for the two signal types and signal sizes.
Most input devices produce updates at 50Hz or less. For local signals this means that glueTK could handle thousands of locally connected input devices. For remotely connected devices, the number of possible devices is still well above a hundred. Of course, actually using this many devices at the same time is not likely, even in a large multi-display en-

	342 characters	*76480 characters*
local	115500.11sps	111844.31sps
remote	6690.75sps	407.84sps

Table 3.1: Signal performance for local and remote signals in signals per second (sps).

vironment. However, glueWidget communication also uses signals and will reduce these numbers slightly. Nevertheless, the results from this performance evaluation show that the performance of glueTK leaves enough room for many devices connected locally or remotely.

3.2.2 Developer Survey

A framework is primarily aimed at developers to allow the creation of a specific category of applications. While a main goal of a framework is to make the development of applications of that particular type easier and faster, it still requires developers to learn a usually complex structure.
To get an insight into the use of glueTK by developers, a user study was conducted with ten C++ developers, aged 25 to 33. None of the developers had any prior experience with glueTK. The C++ proficiency among the users was high. On a scale from 0 (= No knowledge at all) to 5 (= Expert) the average was 3.85 ($\sigma = 0.58$). All participants are active programmers with at least 15 hours programming per week ($\mu = 22.85, \sigma = 6.38$). The experience with other GUI development was low in comparison, on a scale from 0

(= No GUI development experience) to 5 (= GUI development on a regular basis), the average was 1.95 ($\sigma = 1.38$). From this data it can be assumed that the C++ knowledge required to fulfill the tasks of the user study were no challenge for the participants, at the same time, no one had significant GUI development experience.

Set-up and Tasks

The developers were presented with four tasks that reflect some of the core concepts and functionality of glueTK. The first task required users to place an image at the center of the screen and then place another object relative to the first image (Figure 3.10).

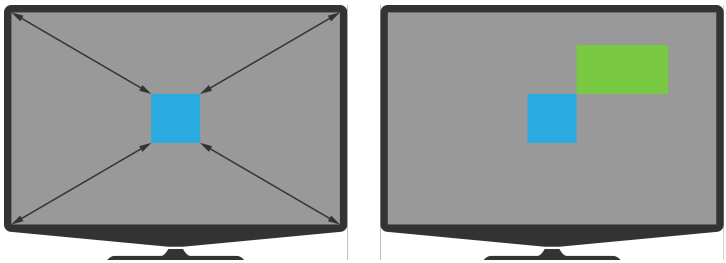


Figure 3.10: Task 1: Placement of images.

While this task could be solved by manual measurement, glueTK provides a powerful anchor point system that makes the placement much easier.
The second task involved the control of the position of an

image using the mouse (Figure 3.11).

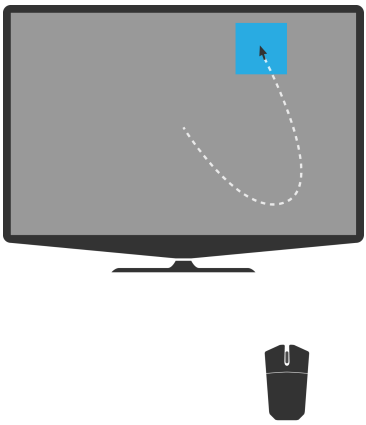


Figure 3.11: Task 2: Mouse interaction.

Task 3 was identical to task 2, except that the position of the image had to be controlled using the Leap Motion [Lea] as depicted in Figure 3.12. The Leap Motion is an input device that detects a user's hand above the device and provides a hand model in 3D. While the mouse data in task 2 is provided by glueTK directly, the Leap Motion required participants to use a dedicated event handler.

Since multi-display environments are a main focus of glueTK, in addition to the integration of novel input modalities, task 4 involved moving an image across two screens that were connected to different machines (Figure 3.13).

All participants were asked to familiarize themselves with glueTK by looking at the overview paper [vdCS13c], the documentation as well as a few tutorial programs that are

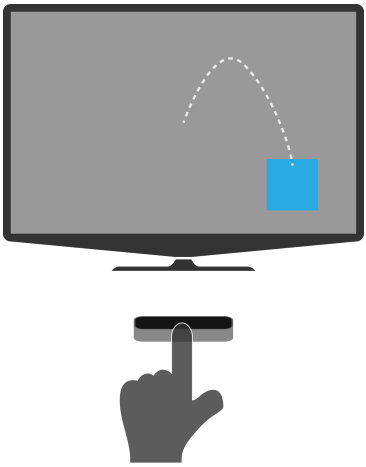


Figure 3.12: Task 3: Integration of the Leap Motion.

part of glueTK to provide examples for basic functionality. Subsequent to this introductory task, participants were asked to complete all four programming tasks and fill out a questionnaire (See Appendix A.1). In addition, all developers had to complete a standardized questionnaire, the so-called "User Experience Questionnaire (UEQ)" [LHS08]. The User Experience Questionnaire is designed to make an assessment of the user experience of interactive products. In addition to these questionnaires, the time required for each task was measured.
To reduce the required time for participants, templates for every task were provided. These templates did not contain any code relevant to the tasks, but allowed for all tasks to

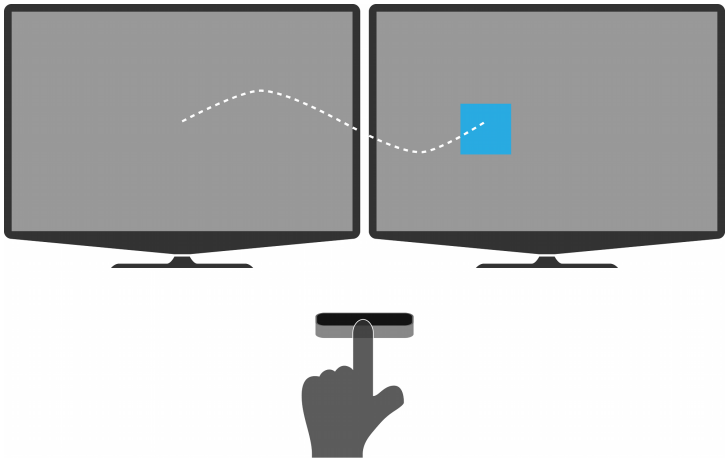


Figure 3.13: Task 4: Multi-display interaction with the Leap Motion.

be configured within the CMake[6] build system beforehand. This eliminated the risk of time being wasted on configuring the build system, which can be a time consuming task, depending on the developer's familiarity with it, and is not relevant to the use and understanding of glueTK.

Results and Discussion

The average time to complete all four tasks was 65.3 minutes ($\sigma = 16.96$). This means that all participants managed to successfully complete all tasks in a reasonable amount of

[6] http://www.cmake.org

time. The longest duration for all tasks was 106 minutes, which, considering developers had no prior knowledge of glueTK, is not uncommon for learning a new framework. Users were asked to give estimates for the distribution of time over the tasks. Figure 4.10 compares these estimates with the actual, measured time.

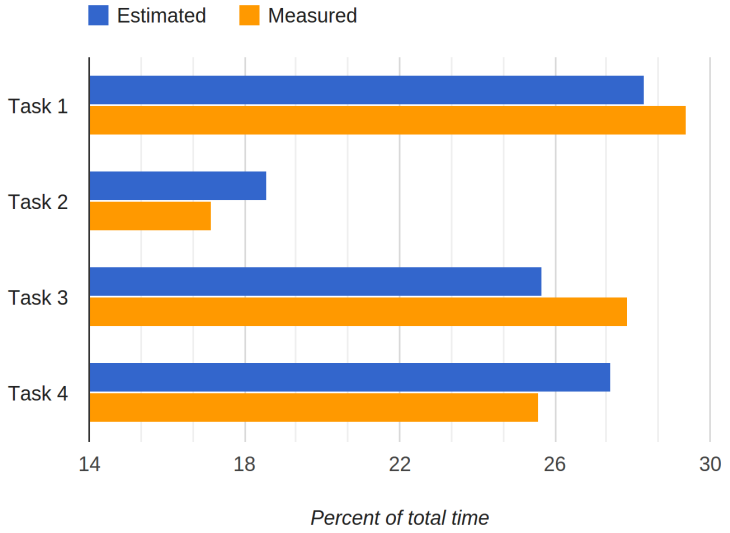


Figure 3.14: Comparison of estimated and measured time to complete each task.

Users rated the available sources of information on a five point scale (1 = not used at all, 5 = main source of information). Table 3.2 shows the average results along with the respective standard deviations of each information source used by the developers. Both, the overview paper as well as

Source	Mean	Standard deviation
Overview Paper	2	1.155
Documentation	3.3	0.67
Source code	1.6	1.075
Tutorials	4.7	0.48

Table 3.2: Ratings of how valuable different information sources were to developers.

the source code, were used less than the documentation and tutorials. While all four are valid sources of information, the distribution shows that no deep understanding of the framework from either the overview paper or the source code was necessary to complete the tasks and utilize the functionality of glueTK.

Figure 3.15 shows how the developers spent their time in average. The strong focus on reading and testing is not surprising since the framework was completely new to all participants. Besides the three dominant categories, only understanding the Leap Motion and the data it provides was mentioned.
The User Experience Questionnaire asks users to decide on tendencies between two opposing pairs on a seven point scale, for example fast and slow or good and bad. All of the 26 pairs can be assigned to one of the six categories: attractiveness, perspicuity, efficiency, dependability, stimulation and novelty. The ratings are transformed to a scale from -3 (horribly bad) to 3 (extremely good). On this scale the average results for each category are interpreted as follows.

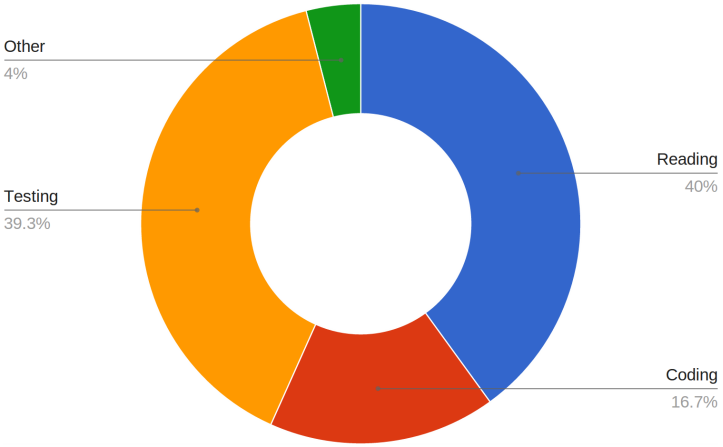


Figure 3.15: Distribution of time with respect to typical programming tasks.

A value between -0.8 to 0.8 is considered neutral. Values larger than 0.8 represent a positive evaluation and values lower than -0.8 are considered a negative evaluation. According to the authors of the UEQ, there exists a tendency to avoid extreme answers which results in values above 2 or below -2 to be unlikely. Table 3.3 shows the results of the User Experience Questionnaire as mean values for each category along with the standard deviation and the 95% confidence level.

The results show, that the participants of the user study perceived glueTK positively with respect to all six categories. Considering the typical upper bound of 2.0 of the UEQ, the results of more than half of the categories are close to upper boundary of the scale. While this study can

Scale	Mean	Standard deviation	Confidence
Attractiveness	1.733	0.639	0.396
Perspicuity	1.150	0.766	0.475
Efficiency	1.725	0.640	0.396
Dependability	1.550	0.685	0.425
Stimulation	1.850	0.428	0.265
Novelty	1.750	0.500	0.310

Table 3.3: Results for the six scales of the UEQ questionnaire.

only give a small insight into the use of glueTK, the results show that glueTK can be used by developers to accomplish typical tasks in multi-display environments with a feasible learning curve. The UEQ results show that participants were highly satisfied with all aspects of the user experience of glueTK.

4

Functionality

The previous chapter described the architecture of glueTK and the different aspects of design that are specifically tailored towards multi-modal and multi-display interaction. In this chapter, several contributions will be presented that build upon the functionality offered by glueTK. As pointing gesture recognition is one of the more prominent novel input modalities, there is a special focus on improving and integrating it.

The first section makes use of the availability of layout information of the user interface. It describes a novel type of force fields that do not only significantly improve the

accuracy of target acquisition with a pointing gesture, but enable the use of small elements that are not usable at all without force fields.
The second section exploits the fact that glueTK is split into separate input and output layers by introducing a novel, computer vision based method for target localization for any existing application. This allows not only the use of force fields with existing applications but also the use of the whole input layer of glueTK.
While these two advancements allow for reliable and accurate target acquisition for any application, the problem of device-free target selection is still an unsolved problem. Section 4.3 presents a taxonomy of gestures as a systematic approach to finding the optimal way of triggering a target selection with the pointing arm. GlueTK is here used for rapid prototyping by creating a cross-device user interface for a Wizard-of-Oz user study.
The animation manager described in the previous chapter is not only intended for visual effects and eye-candy. Animations often help users understand the interface better. Section 4.4 evaluates the effect animations have on perceived latencies when transferring interface elements across displays and machines.
The output layer and especially the concept of building blocks, which can be used to create more complex widgets, is extremely flexible. When multiple displays with different sizes, resolutions and different input devices with varying accuracies are involved, it is hard for developers to scale blocks to work well for all systems and to understand the implications of the design choices they make. To assist

developers in designing usable interfaces for multiple interactive systems with different input and output properties, Section 4.5 explains how to choose element sizes for novel interactive systems.

4.1 Target Acquisition

Many novel input modalities, like pointing gesture recognition, are less accurate than a mouse. With more accurate sensors and improved algorithms the accuracy will improve. The focus on a natural interaction, however, often directly involves the human body as part of the input, which limits the achievable accuracy from the start. Movements of the human body are inherently inaccurate to a certain degree; it is, for example, impossible to hold a hand perfectly still. So no matter how accurate the registration of the human body will get, there is an inherent inaccuracy every system has to deal with.
The interaction between a user and the interface, however, consists of more than just the input modality. It also includes the opposing side of interpreting the data and using it for meaningful actions. Improving the overall interaction is also possible on the side of the interface. A simple example is the use of larger elements. Those are easier to hit and can counteract a certain inaccuracy of the input device. In real world applications, this is of course limited by the available surface area to still accommodate all interface elements and is therefore no feasible option in most cases. There are, however, many techniques for enhancing the accuracy of pointing data on the side of the interface, for example area cursors, gravity wells, force fields, sticky icons, semantic pointing, bubble cursors and object pointing [KB95, WWBH97, FJS10, GB05, GBBL04, Bro05]. These techniques, however, rely on context information; at the very least, they require the position of in-

terface elements and in most cases also their size. One of the design goals of glueTK, as described above, is to make this information available throughout the framework. It is therefore possible to immediately utilize these pointing enhancements in any glueTK based application. Force fields are especially interesting as they show great improvements on the pointing accuracy [AHL06]. Their configuration and placement, however, is tedious. The next section describes dynamic Gaussian force fields [vdCS13b], which address these downsides to show how force fields can easily be used to significantly improve pointing gesture interaction.

4.1.1 Dynamic Gaussian Force Fields

A force field is an invisible area around the center of an interface element that manipulates the cursor position in this area to move towards the center of the element. The field has a defined strength, which is usually a percentage of the total offset between the elements center and the current cursor position. This means, a strength of 1.0 would move the cursor directly to the center, a strength of 0.5 would move it half way and a strength of 0.0 would not affect the cursor at all.
It is often difficult to find the right strength because more correction means more accuracy but also a higher risk of unwanted cursor manipulation. Another problem is the placement of force fields. They can not overlap, which makes placement in real world scenarios problematic and always requires manual placement for every single layout. As much as the force fields can help, cursor manipulation

can be irritating if a user did not intend to click. Section 4.1.1 describes the modeling of force field strengths as a Gaussian distribution to overcome the problem of finding an optimal strength and also solve the problem of overlapping fields. The next subsection shows how force fields can be integrated into a Kalman filter, eliminating an additional data manipulation step. In the following subsection, the problem of undesired force field effects is addressed by predicting possible targets. And finally, Section 4.1.2 presents a user study in which dynamic Gaussian force fields were evaluated.

Gaussian Force Fields

The main reason for the difficulties involved with finding a good strength for force fields is that "good" depends on the situation. If the cursor is close to the target, it is likely that the user actually wants to trigger a click and much force should be applied. If the cursor is further away, even within the force field, little force should be applied, as the user might just want to pass by a target. So instead of an area, which immediately switches from no force at all to a fixed strength, the approach taken here increases the force gradually as the cursor gets closer to the center. To achieve this gradual increase, the force fields are modeled as a Gaussian distribution around the center of a target. Figure 4.1 illustrates the gradually increasing force around a button. This causes a maximum strength to be used when close to the center, providing maximum assistance, without causing the cursor to suddenly jump when entering the

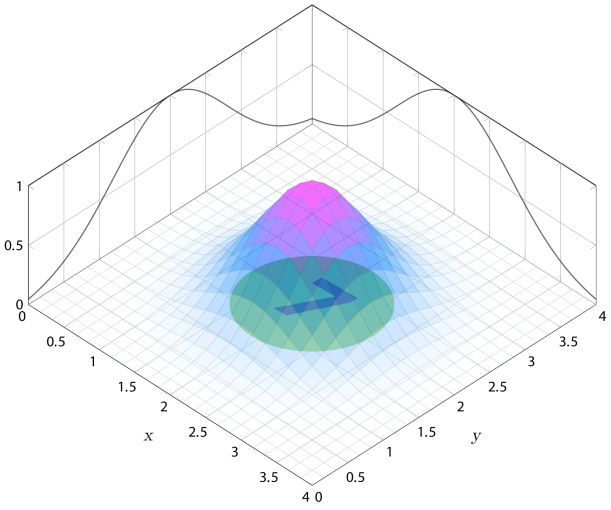


Figure 4.1: Gaussian model of the attraction force of a button.

field. As described above, the effect of a force field is an offset, with a magnitude depending on the field's strength, added to the current cursor position. Modeling the strength as a Gaussian distribution, the offset (x_w, y_w) can be calculated as in Equation 4.1 using the cursor position (c_x, c_y), the button's location (b_x, b_y) as well as the value of the bivariate normal distribution f with a standard deviation of (s_x, s_y) at the current cursor position.

$$x_w = (c_x - b_x) f(c_x, c_y, s_x, s_y)$$
$$y_w = (c_y - b_y) f(c_x, c_y, s_x, s_y) \tag{4.1}$$

An optimal force field depends on the context of the user interface, its element sizes and layout. Because of the lack of overlap handling, force fields needed to be placed manually for every new interface layout. The Gaussian modeling allows a graceful, automatic handling of overlapping fields. As the effects of each field are directional offsets of a certain distance, adding these offsets will eliminate opposing forces. Figure 4.2 shows an example of two overlapping force fields. For a cursor exactly between the fields, the

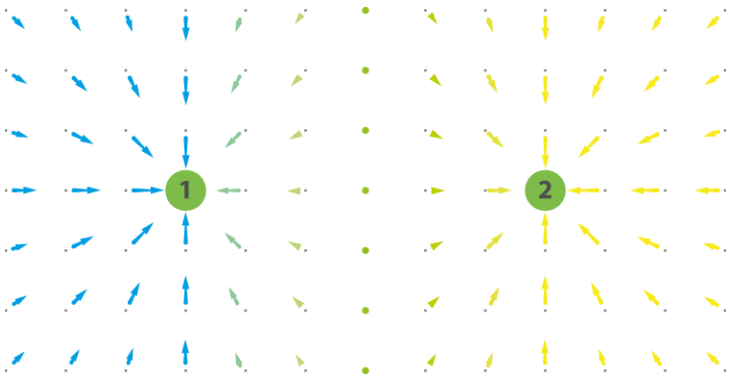


Figure 4.2: Illustration of how the attraction force modeled by multiple Gaussians for buttons with overlapping force fields is compensated.

sum of offsets would eliminate any effect of either field. This is the best possible behavior in this case, as there is no information about which button is a more likely target. As the cursor moves closer to either button, a force is applied, however, weakened by the other nearby field as it is still possible that the user actually intended to click the other button. The offset (x_w^n, y_w^n) applied by the n-th force field is then calculated as in Equation 4.2 using the cursor position (c_x, c_y) with f being the normalized value at the current cursor position of the bivariate normal distribution around the buttons center (b_x, b_y). Normalization here means a maximum of 1.0, which would cause a maximum correction and move the cursor to the exact center of the button, while 0.0 does not manipulate the cursor position at all.

$$\begin{aligned} x_w^n &= (c_x - b_x^n) f^n(c_x, c_y, s_x, s_y) \\ y_w^n &= (c_y - b_y^n) f^n(c_x, c_y, s_x, s_y) \end{aligned} \tag{4.2}$$

The final offset (x_w, y_w) applied to the cursor by the N force fields is calculated using Equation 4.3.

$$x_w = \sum_{n=0}^{N} x_w^n \qquad y_w = \sum_{n=0}^{N} y_w^n \tag{4.3}$$

Kalman Filter Integration

Input data is often noisy, especially when input devices are based on computer vision. To improve the user experience, this data can be filtered to smooth the trajectories. One option to do so is the Kalman filter [Kal60]. It models the system by using previous observations to make predictions about the future state of the system. The design of the interface, especially the layout, heavily affects the user's movements. Typically, however, for filtering input device data with a Kalman filter, this information is not incorporated. Usually, the filter will only improve the noisy data by compensating for measurement errors and bridge short gaps of missing data. To show how the layout information of the interface can be incorporated into a Kalman filter as force fields, initially the setup of the Kalman filter for pointing data is described. The measurements are the coordinates on the screen (x, y), received from the input device. The system state is modeled using this position and the velocity (v_x, v_y). To account for the position and velocity, the transition model M is set up as shown in Equation 4.4.

$$M = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.4}$$

With an estimate for the measurement error w, the current state of the system s_t can be derived from the previous state s_{t-1} using the Kalman filter as in Equation 4.5.

$$s_t = Ms_{t-1} + w \tag{4.5}$$

A Kalman filter set up like this will smooth the cursor movement noticeably. To incorporate the above mentioned context information, the control input of the Kalman filter is used as it is well suited to introduce additional parameters to the system model. By calculating the total offset at the current cursor position as described above, the control vector b_t can be set up to model the force fields as shown in Equation 4.6.

$$b_t = \begin{pmatrix} x_w^t \\ y_w^t \\ -v_x^{t-1} \\ -v_y^{t-1} \end{pmatrix} \tag{4.6}$$

Since getting close to a button makes it more likely that the user will try to stop the cursor, the velocity is eliminated by setting the velocities in the control vector to the inverse of the last known velocity. This allows the use of a control matrix C identical to the transition matrix M. The additional control input results in an overall Kalman filter step as in Equation 4.7.

$$s_t = Ms_{t-1} + Cb_t + w \tag{4.7}$$

Dynamic Force Fields

Force fields, like most other target enhancement techniques, are only helpful when needed. That is, if the user actually intends to click on a target. Otherwise, they can interfere with the interaction by causing irritations or even errors. To achieve an optimal solution in all cases, it is possible to predict the user's behavior and dynamically activate and deactivate force fields accordingly. Human motion has been studied in detail [MU02] and results in cursor movements that always include acceleration and deceleration phases. Deceleration phases are typically observed before a click is triggered, as the user slows down the cursor to accurately hit a target. This indicator can be more or less distinct, but an immediate stop from full speed is neither natural nor common. A deceleration phase D can be detected as shown in Equation 4.8 by comparing the current (v_x^t, v_y^t) and the previously observed (v_x^{t-1}, v_y^{t-1}) velocity.

$$D = \begin{cases} 1 & (v_x^t - v_x^{t-1} < 0) \vee (v_y^t - v_y^{t-1} < 0) \\ 0 & else \end{cases} \tag{4.8}$$

Using this detection mechanism, the input data is continually analyzed and force fields are only activated if a deceleration phase is detected. They stay active until a click occurs or an acceleration is detected.

4.1.2 Evaluation

A user study was conducted to evaluate dynamic Gaussian force fields for pointing enhancement. Participants were asked to use three different variants of a pointing system. All relied on a computer vision based pointing recognition system described in [SvdCIS09]. For the first variant, *PLAIN*, the input data is smoothed by a Kalman filter but no force fields are used at all. For the second variant, *SFF*, the input data is smoothed using a Kalman filter as well and, in addition, static force fields are applied in a further step. These force fields are always active and have a fixed strength. The third variant, *DGFF*, are the dynamic Gaussian force fields described above, using a Gaussian distribution to model the strength and dynamic activation. In addition to measuring cursor positions, speed, accuracy, duration and errors for a quantitative evaluation, users were also asked to fill out a short questionnaire (see Appendix A.2) to get feedback on their impressions.
As a display, a $4m \times 1.5m$ back projection video wall with a resolution of $4096px \times 1536px$ was used as depicted in Figure 4.3.

The pointing gesture system extracts the pointing direction by calculating a 3D reconstruction of the area in front of the video wall from two color cameras with an overlapping field of view. Using the location of the display and the coordinate system of the calibrated cameras, the 2D intersection point of the extension of a pointing arm with the video wall's surface can be calculated and converted to pixel coordinates. This allowed users to control the cursor

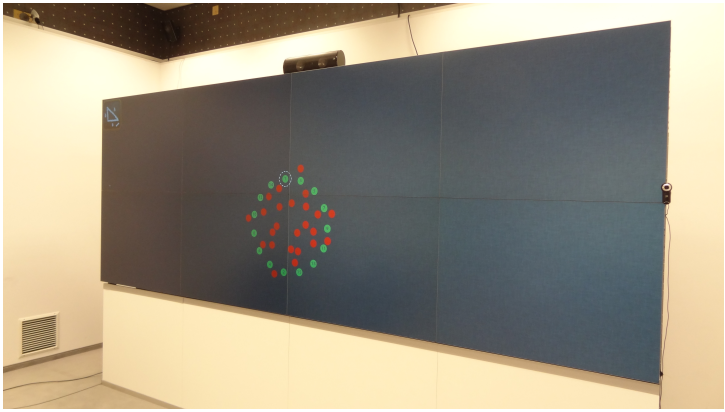


Figure 4.3: Video wall used for the user study with initial button layout in practice mode.

by simply pointing towards the desired position on the wall as shown in Figure 4.4.

The pointing position is updated at $30Hz$ and a click is triggered using a dwell timer as described in Chapter 3. The configuration of the dwell timer used for this evaluation was an allowed deviation of $5px$ to consider the cursor to be stationary and keeping it this way for $0.5sec$ to trigger a click. Users got visual feedback about the click progress as shown in Figure 4.5.

Of the eleven participants, aged from 21 to 32, there were ten male users. Nine users were right handed and all of them used their respective primary arm over the course of the entire experiment. Seven users had used a pointing gesture recognition system before, but not necessarily the one used in this experiment. All users had normal or

Figure 4.4: User interacting by means of a pointing gesture.

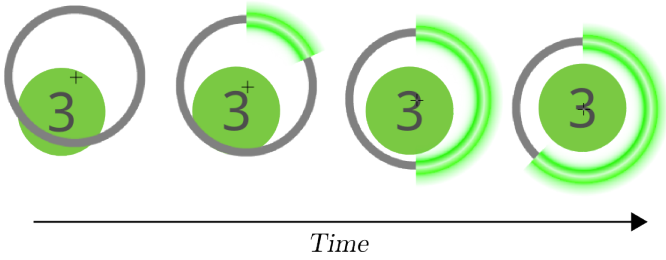


Figure 4.5: Visual feedback for a dwell used for all three techniques.

corrected to normal sight. The average height of participants was 180.3cm and buttons were placed at heights from 99cm to 196cm from the ground. Following the ISO 9241-9 standard [ISO00], participants had to perform a multi-

directional pointing task (compare Figure 4.6).

As suggested in the ISO standard, the circle size was var-

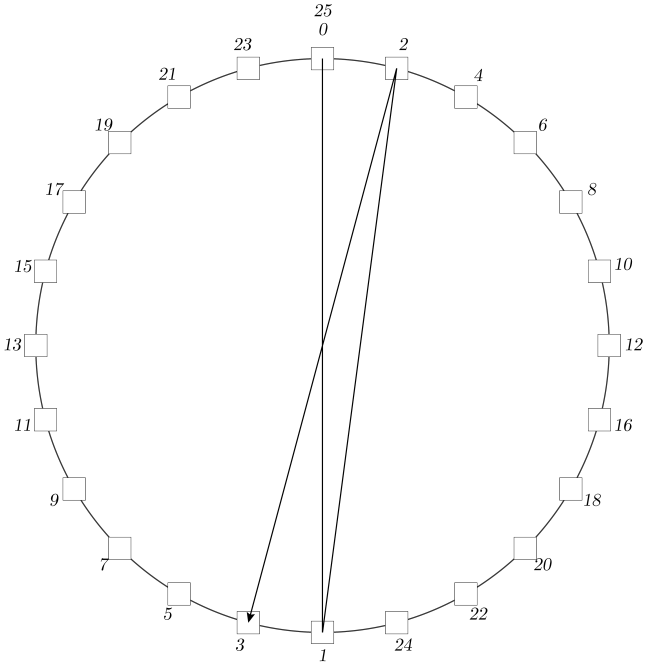


Figure 4.6: The multi-directional pointing task as described in ISO 9241-9. Participants have to click the targets in the indicated order.

ied. Two sizes, a $1000px$ ($\approx 1024mm$) and a $400px$ ($\approx 409mm$) diameter circle, were used. For the targets, but-

tons of three different sizes were used ($12px$ ($\approx 11.7mm$), $25px$ ($\approx 24.4mm$) and $50px$ ($\approx 48.8mm$) in diameter). The ISO standard only requires the display of the actual targets that will be used. This, however, does not reflect the layout of real world applications accurately. Especially in the case of force fields, which can negatively influence the cursor if they are not needed, it does not account for all aspects of a real world application. Therefore, in addition to the 15 numbered green buttons, 24 additional red buttons were added to the interface (see Figure 4.7). The placement of these non target buttons was randomized once for every combination of circle diameters and button sizes, and then remained the same for all users participating. The red buttons behaved exactly the same as the green buttons, but users were instructed to never target them on purpose. The green buttons were labeled according to the order suggested in ISO 9241-9. To avoid any confusion about the next target, a white, dashed indicator circle was displayed around the current target. Figure 4.7 shows the button layout for the $400px$ circle and $50px$ buttons in its initial state with the indicator circle around button 1.

Each participant had to click all 15 buttons in the predefined order for both circle sizes and all three button sizes, resulting in a total of 6 runs per technique. Users were asked to perform each task as quickly as possible, but to balance speed and accuracy. To avoid any effects of fatigue, users were able to take breaks between tasks at will. The order in which the participants used the three different techniques was randomized and users were allowed to practice each new technique before the start of the exper-

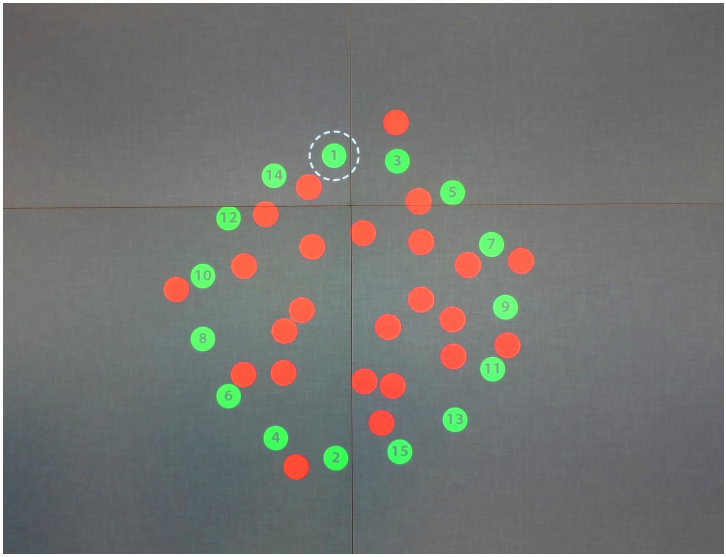


Figure 4.7: One of the two button layouts with the largest of the three button sizes used in the evaluation.

iment until they felt comfortable. To maximize the reproducibility of the accuracy of the pointing system, a small mark on the floor, 74*cm* from the video wall, was used for users to stand on to avoid any influence the distance might have on the pointing recognition. Unlike dynamic Gaussian force fields, which vary their strength depending on the cursor's distance to a target's center, a fixed strength for static force fields needs to be defined. In [AHL06], a strength of 0.8 is recommended and was used in this experiment for the static force fields. The size of the static force fields depended on the available space due to the lack

of overlap handling, which resulted in force field sizes from the size of the target buttons to a maximum of three times the size of the target buttons. For the dynamic Gaussian force fields, a standard deviation of $30px$ was used throughout the experiment.
During the experiment, all clicks with time-stamp and position, along with the ID of the clicked button and the current target were recorded. Independent of clicks, all cursor positions over the course of the whole experiment were recorded as well. From this data, a number of measurements were calculated: the number of erroneous clicks, the time it took to perform a click and the offset to the target's center at the time of the click. Especially with the additional buttons, which were not intended as targets, it was interesting to see if this would lead to an increase of erroneous clicks due to the force fields. Figure 4.8 shows the total number of erroneous clicks by technique and button size.

While the number of erroneous clicks is higher for static force fields (*SFF*) than for the dynamic Gaussian force fields (*DGFF*) or the variant without any force fields (*PLAIN*), considering the total number of more than 900 clicks per technique, this still seems marginal and without serious impact on the usability of either variant.
The recorded offsets to the targets' centers give an insight into the accuracy of each technique, as a large offset means it is more likely that a click will miss a button. Figure 4.9 shows the average offsets for each variant and button size respectively.

The offsets for the *SFF* show that the fields improve the

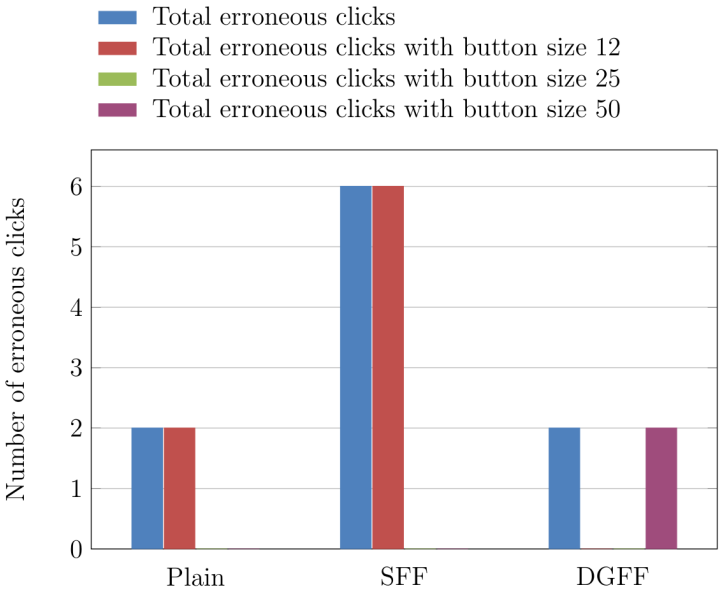


Figure 4.8: Total number of erroneous clicks.

accuracy compared to the *PLAIN* technique. The *DGFF* achieve results for all button sizes that are close to eliminating any offset at all. The reason for this is the high strength of the force fields at the very center, which causes the cursor to be pulled towards it with enough force that it reaches the center before the click is triggered. Another important property is the interaction speed. The time it took to click a button is defined for this experiment as the duration from leaving the previous target to the click on the current target. Since two different circle sizes were used, times were normalized to a distance of $1000px$ to allow for

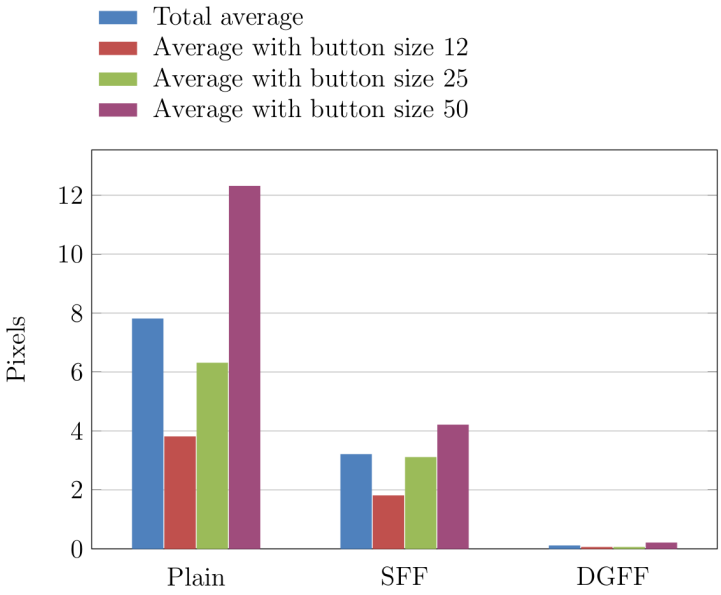


Figure 4.9: Average offset of the actual click position from the targets center.

comparison of all results. A comparison of the circle sizes in separate did not show any noticeable differences. Figure 4.10 shows the overall average results as well as the average results for each button size by variant.

Standing out especially are the long times it took to click smaller buttons with the *PLAIN* technique. Considering an average time of more than six seconds renders this technique unusable for real world applications. Here, the pointing enhancements make the input modality usable in the first place. Both types of force fields show significant im-

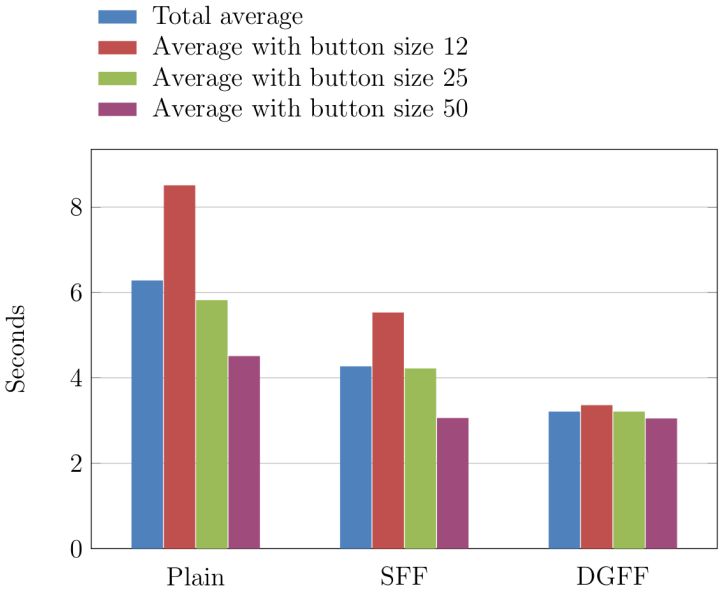


Figure 4.10: Average number of seconds to click a target at a distance of 1000px.

provements over the *PLAIN* technique (*SFF* are 32% faster and *DGFF* are 48% faster). The advantage of the automatic overlap handling of the *DGFF* becomes especially apparent for the smaller button sizes, where larger force fields can be used, which leads to significant improvements over the *SFF*. In Table 4.1, the average durations in seconds (standard deviation in brackets) are given for each variant and button size. The fifth and sixth column show the t-test p-value for *DGFF* over *PLAIN* (*PLAIN) and *DGFF* over *SFF* (*SFF) respectively. Statistically signifi-

cant values are printed in bold font.

Size	PLAIN	SFF	DGFF	*PLAIN	*SFF
Avg.	6.27(2.50)	4.26(0.83)	3.20(1.42)	**0.002**	**0.045**
12px	8.50(2.67)	5.52(0.76)	3.35(1.17)	**0.000**	**0.000**
25px	5.81(1.47)	4.21(0.96)	3.20(1.03)	**0.000**	**0.027**
50px	4.50(1.03)	3.05(0.72)	3.04(0.79)	**0.001**	0.975

Table 4.1: Average time in seconds for all variants and button sizes as well as overall average (Avg.). The standard deviation is given in brackets, the last two columns show the respective p-values of a t-test.

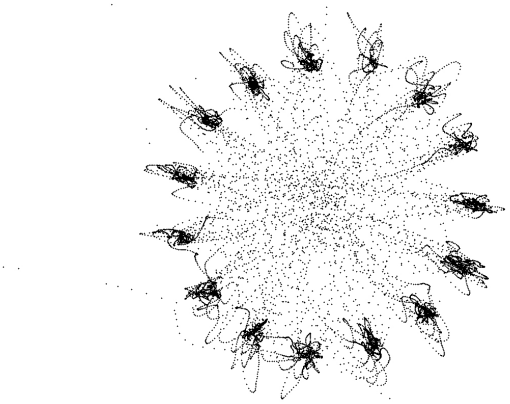


Figure 4.11: Cursor movements with no force fields.

The results from the questionnaire given to participants to complete during the experiment coincided with the conclusions drawn from the measurements. Users perceived the *DGFF* as the fastest technique ("Which technique seemed

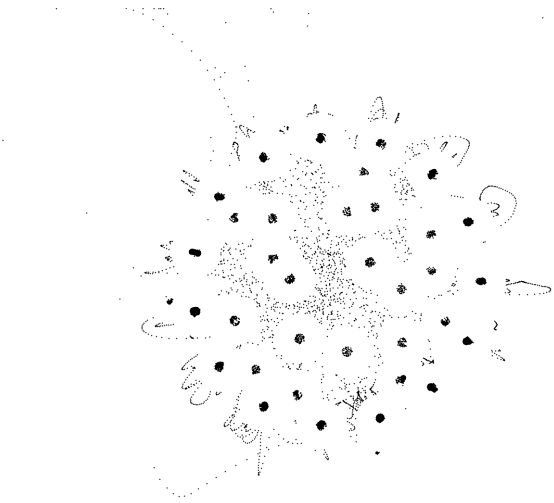


Figure 4.12: Cursor movements with static force fields.

to be the fastest?" *PLAIN*:0, *SFF*:1, *DGFF*:10) and also favored the *DGFF* overall ("Which technique did you like best?" *PLAIN*:0, *SFF*:2, *DGFF*:9). The recorded cursor positions over time from all experiments were used to plot heat maps to get an insight into the movement properties of each technique. To avoid clutter, only the cursor positions for experiments with the $1000px$ circle are plotted in Figure 4.11, Figure 4.12 and Figure 4.13.
The heat maps show, that the movement with the *PLAIN* technique is very smooth (Figure 4.11), but also that there is much scattering around the targets, which is the result of users trying to keep the cursor on the buttons. The heat map for the *SFF* shows much more convergence towards

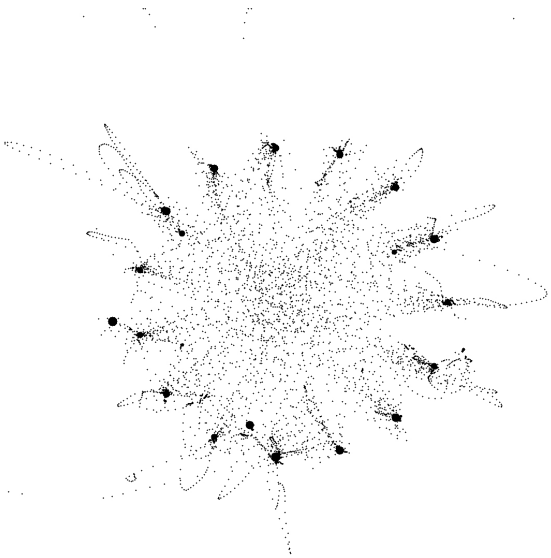


Figure 4.13: Cursor movements with dynamic Gaussian force fields.

the centers of targets (Figure 4.12) but also shows how non target buttons affected the cursor movement and led to a "jumpy" behavior, which was mentioned in the comment section of the questionnaire by several participants as well. The *DGFF* produced a heat map that results from a cursor movement that combines the advantages of the other variants without their downsides (Figure 4.13). The cursor movement is smooth, without much irritation from the non target buttons just like the *PLAIN* variant, while achieving a strong focus around the targets' centers similar to the *SFF*.

With the automatic handling of overlapping fields and the intuitive field strengths of dynamic Gaussian force fields, a pointing gesture becomes accurate and fast enough to be used with complex interfaces. The natural interaction of the modality is preserved due to the automatic activation of force fields, which allows for natural movement. The only condition is that the layout of the user interface is known, so that force fields can be placed at those locations.

4.2 Target Localization

Software development is always an expensive and time-consuming endeavor. For this reason, it is not realistic to expect existing software to be reimplemented using glueTK to integrate new input modalities or make use of its multi-display support. This is why glueTK is designed to offer benefits for existing applications as well. Chapter 3 described how glueTK is split into an input layer and an output layer, which allows independent use of the input layer in existing applications to easily integrate new input modalities. In cases were the source code of an application is accessible, glueTK can easily be integrated as an additional library. There are, however, cases were the existing application can not be modified. This is true for closed source applications with no intention by its original creators to integrate new input modalities, but also in many high security environments. For both military and civil control rooms it is common that application software runs on an isolated machine. It is not allowed to run any additional software on this machine and the only way to interact with it are mouse and keyboard signals which are streamed via a network just like the application's visual output. The following sections present how computer vision techniques can be used to utilize the pointing enhancement of glueTK even in such extreme cases.

4.2.1 Black-Box GUIs

To integrate, for example, a pointing gesture system into an existing application with no access to its source - hence black-box GUI - while making use of a pointing enhancement like force fields, two aspects are crucial. First of all, as existing applications usually understand nothing but mouse input, it is necessary to be able to manipulate the mouse cursor programmatically. This is possible on all major operating systems and therefore does not pose a challenge. Second of all, a crucial condition for the use of pointing enhancement techniques is the knowledge of the interface layout, e.g. the location and size of interface elements. As these properties can change when windows are moved, resized or the context of an application changes, it is not feasible or sufficient to provide the initial locations of those elements manually. The key idea of the black-box GUI approach [vdCS13a] is to exploit the fact that user interface elements are designed to visually stand out and are easily recognizable. This makes it possible to reliably and automatically detect them using computer vision. Instead of building a library of known elements as in the Prefab system [DF10], the goal is to observe the user's behavior and automatically learn the appearance of elements on the fly, which allows the system to work with any user interface without prior annotation or knowledge.

Screen Scraping

To analyze the screen content, it needs to be captured without a large overhead. For high security systems, where the screen content is streamed via a network, this is a given, as the screen content is immediately available as video data. But also in the case of closed source applications it is possible to efficiently capture the screen content on all major operating systems as demonstrated by the many established applications with screen sharing capabilities such as Skype [Sky]. This ensures access to the screen content for further analysis. As mentioned before, the programmatic control of the mouse cursor is possible on all major operating systems. For new input modalities, however, visual feedback is often important. Just like controlling the position of a mouse cursor programmatically, it is often possible to manipulate the pointer icon as well, which allows to give basic visual feedback even without modification of the operating system or the application. Figure 4.14 shows how cursor icon manipulation is used to allow for basic visual feedback of a dwell timer for clicking.

Figure 4.14: Cursor icons to indicate click progress.

Template Matching

Virtually all desktop applications are built using a GUI toolkit, which provides a set of reusable widgets. These widgets look and behave the same way, no matter in which application they are used. Besides assisting the developer, this has the advantage of quickly familiarizing users with new applications as they already know parts of it. The reuse of interface elements and the fact that these are designed to be easily recognizable allows the use of template matching to automatically locate these elements on the screen. In computer vision in general, the use of template matching is often futile as sensor noise, camera movements and angle as well as changing illumination result in a high failure rate. All these factors, however, do not play a role when analyzing screen content. A widget will, pixel by pixel, look identical from one frame to another. This is true for identical widgets but often also for widgets of the same type, e.g. multiple checkboxes. Because of the limited number of widgets provided by GUI toolkits, templates for all widgets could be created beforehand. In a real world scenario this would, however, fail quickly. Most operating systems, not to mention the web, allow for custom skins which results in a virtually unlimited number of possible variations in appearance. Therefore, building a database of templates is tedious and error prone. The approach taken here takes advantage of the fact that it aims at integrating an input device. This provides information about the user's input and, assuming that locations where a user triggers a click are likely candidates for the location of interface ele-

ments, it is possible to automatically extract the element to be used as a template. This template can immediately be used to find further, identical widgets but also serves as a starting point to built a more generic widget model as described in the following section.

Automatic Template Learning

The automatic template learning module constantly monitors the input data. Once a click occurs, the location of the click is used as a starting point to search for a possible interface element at that position on the screen. Since widgets are made to visually stand out, performing canny edge detection allows for a reliable extraction of potential widget borders. To find larger contours within the extracted edges, the Teh-Chin chain approximation algorithm [TC89] is used. Since a successful click would be within the boundaries of the widget, a point-in-polygon test, using the position of the registered click and the found contours, allows the extraction of boundary candidates. From these candidates, the smallest enclosing, convex contour is selected. Figure 4.15 shows three examples of located boundaries of three different widget types starting from the click position indicated by the mouse cursor.

Using the extracted area directly as a template will only allow to locate identical or very similar widgets. To improve this, a widget model is created by expanding the extracted template by 5 pixels in all directions and splitting it horizontally and vertically into four equal parts. The original corners of the contour used for extraction are saved as

Figure 4.15: The mouse cursor indicates the click position, the red boundaries show the extracted contour.

anchor points along with the four image patches (see Figure 4.16). Instead of performing template matching on the

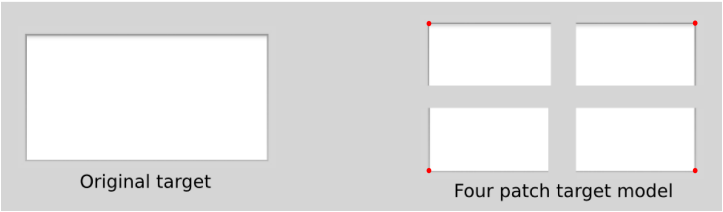


Figure 4.16: Creation of an initial target model (Anchor points are indicated by the red dots).

screen data using a single template for the whole widget, template matching is performed for each of the four patches of the model individually. Matches are then analyzed for their relative position towards each other. A top-left patch must always be to the top-left of a bottom-right patch, left to a top-right patch and above a bottom-left patch. The respective constraints apply to the other patches as well. In addition, there can not be any matches between, for example, a top-left and a top-right patch. These geometric constraints of the widget model ensure that only four matches in the correct configuration will be interpreted as a

successfully located widget. While this model is the starting point for further model refinement, the splitting alone already causes some generalization for widgets of the same type that only vary in size. A simple example of this auto-generalization can be seen for a text-box in Figure 4.17. Generalization means, that after the first click on a text-

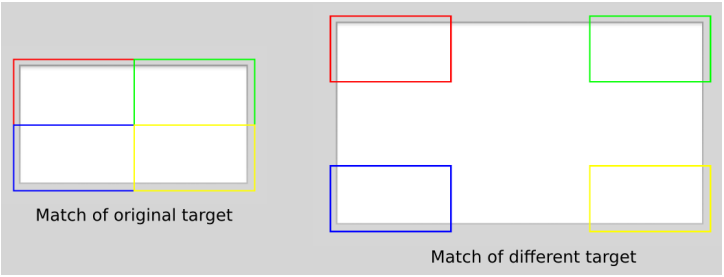


Figure 4.17: Generalisation of an initial target model.

box, every text-box in the interface will be found and force fields can be applied to them. This, however, is usually only the case for very simple widgets like empty text boxes or line edits. For widgets with a higher level of detail and unique areas, further generalization is required to build a model that can locate all widgets of one type. The more examples of the same widget type are available, the easier it becomes to identify which parts of the image patches stem from the general widget type and which are specific to an individual widget e.g. text or labels. The first challenge is to detect if extracted templates are of the same widget type. For this reason, whenever a new template is extracted, its patches are compared to those of already

existing widget models by aligning them at their anchor points. If there are at least 20 consecutive, matching pixels in each of the four patches, the models are assumed to stem from the same widget type and the original model is refined.
The refinement is performed individually for each of the four patches. The first step is aligning the corresponding patches at their anchor points and reducing the patches to the overlapping area as shown in Figure 4.18 for the bottom-right patch. Corresponding patches are then compared, pixel by pixel, and all pixels that differ are set to be transparent. Transparent pixels are simply ignored when comparing them during template matching. In Figure 4.20,

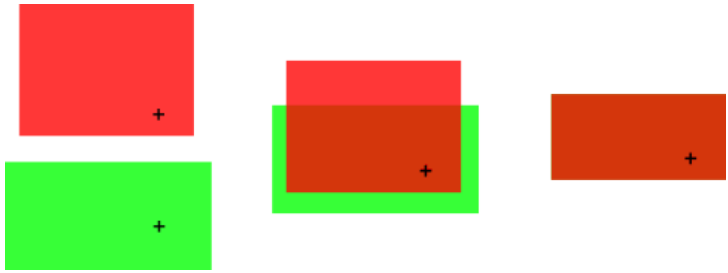


Figure 4.18: Patch alignment for refinement.

the refinement of an existing model of a button (bottom) by a newly extracted template of a button (top) is illustrated. The two buttons differ in size and also have different labels. The refinement process reduces the patches to the area both buttons have in common and eliminates pixels that differ because of the different labels. In this step, another condition for successful refinement is introduced: the edges

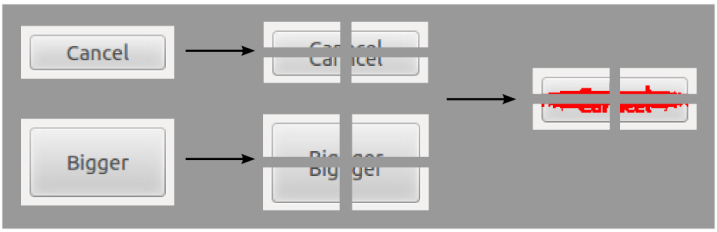


Figure 4.19: Model refinement for generalization (transparent pixels are drawn red for visualisation).

have to remain unchanged in the refined patches. Otherwise it is likely that the refinement would produce over generalizing models that simply match anything. If, however, this condition is met, the newly extracted template is discarded as its information is now contained within the refined model. Otherwise, the newly extracted template is kept as an additional model, which slightly increases the runtime but guarantees that the new target will be found and enhancements can be applied immediately.

Runtime Considerations

Because template matching works by comparing each pixel of the template with each pixel of the screen, both the number and size of templates as well as the screen resolution heavily influence the runtime of the widget localization. The generalizing models presented above significantly reduce the search time as they eliminate the need for many individual templates. Still, considering that many novel in-

put modalities are targeted at large screens, scanning the whole screen can take too long and result in, for example, incorrect force field placement if the layout information is outdated. To avoid runtime related problems, the search area is limited to an area of $500px \times 500px$. This is possible because the current pointing direction is always known from the input data and this "spotlight" can be shifted to the current pointing position. The current pointing position is, at the same time, the only area that needs enhancement as the user is pointing towards this area. Limiting the search area allows for an update rate of $1.4fps$ using a set of 10 widget models which is sufficient considering most interface elements rarely move. Limiting the search area in this way guarantees a constant performance independent of the display size, which makes this approach usable in real world applications.

4.2.2 Evaluation

To evaluate the system described above, two factors are important. First, the template learning needs to be analyzed to evaluate how well interface elements can be extracted based on the presented computer vision based approach. Second, it is crucial to assess if the locations provided by user interaction actually allow for the placement of force fields and that it leads to an improvement of the pointing interaction. The interface element localization depends on successful target extraction and reliable model generalization for target matching. Both aspects are evaluated separately on a variety of interface elements below.

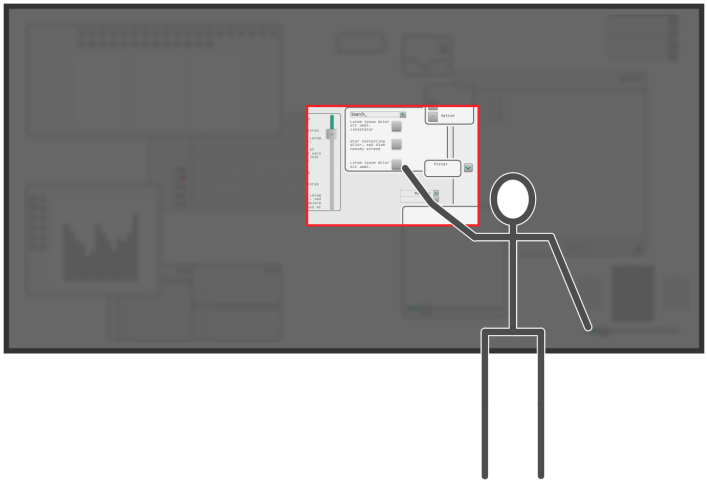


Figure 4.20: The search area is automatically limited to a spot around the current pointing position.

The first condition for creating a target model that can be used for localization is successful extraction. The extraction process described above was evaluated on the default widget sets of the three major operating systems (Microsoft Windows 7, OSX Lion and Ubuntu 12.04), each with their respective default theme, and did not produce any errors. This is due to the visually articulate design of these widget sets. To show the limits of the extraction algorithm, 209 buttons from the web (Figure 4.21) with non-standard designs were used. Four users were asked to click all 209 buttons in random order. This was repeated 10 times. In average, 198 of the 209 buttons were successfully extracted

after a single click of a button.
After extracting an initial template from a clicked target, it is important that the model refinement is fast and reliable. The sooner a model abstracts from an individual widget to all widgets of the same type, the sooner all widgets can be enhanced. The refinement is especially challenging if the widgets vary in size and appearance, due to text or labels. A test interface consisting of 38 buttons, 15 text fields and 15 line edits of varying sizes and with different content and labels was created in style of the themes of the three major operating systems. An examplary interface is shown in Figure 4.22. Ten users were asked to randomly click on buttons, text fields and line edits. In the background, without the users knowledge, the application built models for each widget type and kept track of the number of widgets the current model generalized to. In average, it took 3.4 clicks for buttons, 4.4 clicks for text fields and 3.9 clicks on line edits for the respective model to generalize to all widgets of the same type.

In order to evaluate the interaction of the system with the user, a user study was conducted. A simple application with buttons as targets was displayed on a video wall and users were asked to click the buttons using a pointing gesture recognition system. To compare the automatic force field placement ($ATFF$) by the system above, two additional systems were used. One without any force fields as a baseline (NFF) and one with force fields placed by locating targets by means of a manually created template ($MTFF$). The manually created template was able to locate all buttons successfully, which made a system with

Figure 4.21: Sample of web buttons for automatic widget extraction.

manually placed force fields for comparison redundant.
For the user study, a pointing gesture recognition system based on 3D pose estimation with multiple cameras, described in [SvdCIS09], was used. As a display, a $4m \times 1.5m$ video wall with a resolution of $4096px \times 1536px$ was used. The pointing system allowed users to control the mouse cursor with an update rate of $30Hz$ and to click using a dwell timer with a trigger time of one second along with simple visual feedback as shown in Figure 4.14. An application, created using the Qt framework [Qt2], was used as a user interface with buttons arranged according to the ISO 9241-9 [ISO00] multi-directional pointing test requirements (See

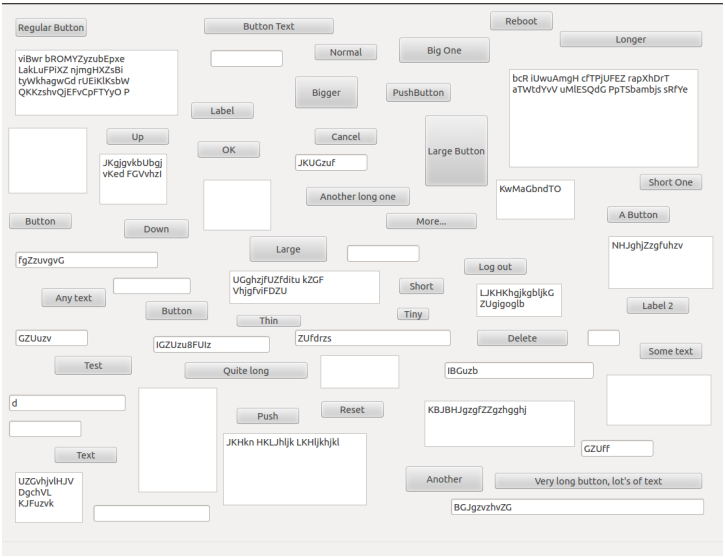


Figure 4.22: Variety of widgets to test model refinement.

Figure 4.23). The application was in no way aware of the pointing system or any pointing enhancements, it would just react to movements of the operating system's mouse cursor. Ten users (9 male), aged from 22 to 32, participated in the user study. Eight users were right handed, all users used their primary arm during the entire experiment. All users had normal or corrected to normal sight. Eight of the users had previous experience with pointing interaction. The average height of participants was 176.8cm and buttons of the interface were at heights from 108cm to 205cm above the ground. In accordance with ISO 9241-9, the size of the circle in which buttons were arranged was

Figure 4.23: User interacting with pointing gesture.

varied using a circle with a $1000px$ diameter and a smaller circle with a $600px$ diameter. Each circle consisted of 15 buttons, sized $80px \times 30px$. The order in which buttons had to be clicked was indicated by numbered labels on the buttons and in addition, the current target was labeled in green while all other buttons had red labels. Users had to click all buttons of both circles in the compulsory order using each of the three different systems *NFF*, *MTFF* and *AFF* respectively. Users were given as much time as they wished to practice before the start of each run. The order in which participants used the three systems was randomized for each user. To avoid fatigue, users were free to take breaks between runs at will. Users were asked to stand on a small mark at $92cm$ from the video wall to eliminate

any effect the distance might have on the accuracy of the pointing gesture recognition. Users were encouraged to select targets as fast as possible but to balance speed and accuracy.
As suggested in [AHL06], a force field strength of 0.8 was used throughout the user study along with a circular field with a diameter of $60px$. All cursor positions during the user study were recorded along with timestamps. In addition, all clicks along with the id of the clicked button, as well as the current target were recorded. This data was used to calculate the time it took users to click buttons as well as the accuracy of each click. The time to click a button is here defined as the time from leaving the area of the previous button to triggering a click on the next button. Accuracy of a click is defined as the offset of the click from the center of a button. Since two different circle sizes were used, the time values were normalized to a distance of $500px$ to allow for all data to be compared. In addition, heat maps from the cursor positions were plotted to gain additional insight into the effects of the force fields on the cursor movement. In a short questionnaire (see Appendix A.3), users were able to report their impressions of the different systems.
The fastest system was *MTFF* (3.5 seconds) which is a significant improvement over the *NFF* system without any enhancement (4.66 seconds). The *ATFF* system achieves comparable performance at 3.66 seconds. The enhanced systems compare similarly in accuracy as the average offsets show: *NFF*:17.12px, *MTFF*:5.25px and *ATFF*:8.59px. The above results are also reflected in the answers given by

participants in the questionnaire. Asked about which system was perceived as the fastest, the enhanced systems got all votes (*NFF*:0, *MTFF*:4, *ATFF*:5, *MTFF+ATFF*:1). Only two participants saw *NFF* as a viable option to control existing desktop applications, however, all participants perceived the enhanced techniques as viable options to do so (*NFF*:2, *MTFF*:9, *ATFF*:8) (All questions allowed for multiple choices).

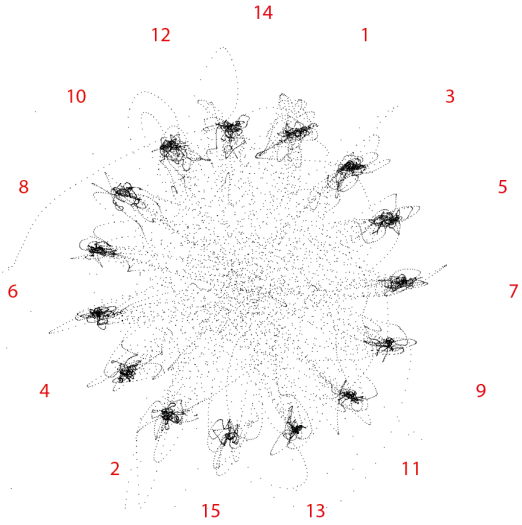


Figure 4.24: No force fields (NFF).

The heat maps plotted from the cursor positions clearly show the effect of the force fields. While there is much scattering around the targets in Figure 4.24 (NFF), the cursor

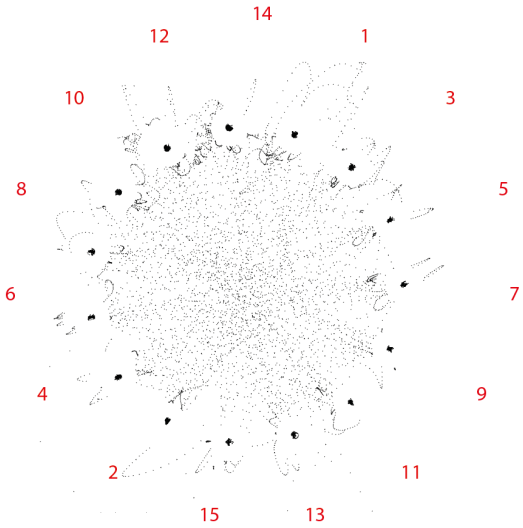


Figure 4.25: Force fields from manually created templates (MTFF).

positions are much more focused in Figure 4.25 (MTFF) and also in Figure 4.26 (ATFF). Especially insightful is Figure 4.26. It shows the cursor positions for the *ATFF* system, which automatically extracts widgets and builds a model, which the system tries to refine with every click. The red numbers around the plot indicate the order in which buttons had to be clicked. It shows that, especially around the first button, the scattering is similar to that around all buttons in Figure 4.24. This is consequent as there is no way for the system to place a force field before the first click. After the first click, the scattering is imme-

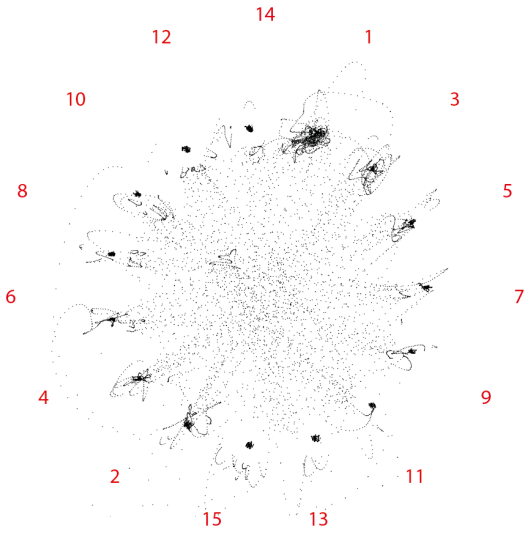


Figure 4.26: Force fields from automatically created templates (ATFF).

diately reduced noticeably around button 2. As the model is refined with each further click, more and more buttons are located due to the generalization of the model and can be enhanced by placing force fields at their locations. This leads to a cursor pattern around the later buttons, similar to the *MTFF* system where all buttons are enhanced from the start.
The automatic localization of interface elements allows to apply pointing enhancements to any existing interface as the interface layout is the information many enhancement technologies rely on. With the dynamic Gaussian force

fields, described in the previous section, and the automatic target localization for existing applications, pointing gestures can be used for fast and accurate target acquisition in any application.

4.3 Target Selection

The previous sections of this chapter have shown that the target acquisition of a pointing gesture recognition system can be significantly improved by utilizing knowledge about the interface and its layout. They have also shown that it is possible to extract the required layout information from existing applications without access to the source code and that pointing gestures can be used for accurate target acquisition in all applications utilizing the presented technologies.
Target acquisition is, however, only the first part of a typical pointing interaction. In addition, the target has to be selected, or, as many years of the mouse as a dominant input device have coined it: the target has to be clicked. For a mouse, where the user is required to hold a device to point, it seems natural to simply add a button for target selection. Pointing gestures, however, do not require the user to hold a device, so introducing a device just for target selection would take from the natural use of the input modality. While gestures have been an important topic in human-computer interaction for a long time, e.g. [Bol80, Ken04, PSH97, VB05], much of the work is very broad and includes triggering a click with two arms or additional hardware devices. Using anything but the pointing arm puts additional strain on the user. While gestures in general can be used for more complex interactions than target selection [BRB09, NWP+11] it is one of the most fundamental interactions, which makes it very powerful. While several clicking gestures have been pro-

posed, there are two fundamental drawbacks in previous work. First, the number of gestures is small and does not allow for a comprehensive overview [VB05]. Second, the evaluations are usually based on actual, computer vision based systems [VB05, BRB09, SvdCIS09]. Such systems are not perfect and the quality of a specific recognition system will influence a user's assessment of the quality of a gesture.
Existing taxonomies [GB05, McN92, Que94, Que95] for gesture interaction are not fine grained enough to differentiate different clicking gestures of the pointing arm. Therefore, a systematic analysis of gestures for target selection using the pointing arm will be presented in the following sections. First of all, a taxonomy of clicking gestures will be introduced that covers all possible options to trigger a click with one arm. Secondly, the gestures of the taxonomy are evaluated in a Wizard-of-Oz [1] user study, followed by a thorough discussion of the results and their implications. The Wizard-of-Oz setup requires complex control of the interface, as users need to feel like they are triggering actions while the hidden experimenter needs to observe the participants and trigger events remotely. GlueTK is therefore used as a rapid prototyping tool that allows to quickly

[1] The term "Wizard-of-Oz" is used for a type of experiment in which a technology is simulated by a hidden (usually in a different room) experimenter. This is especially useful for technologies which involve tasks that can be perfectly executed by a human but not a machine or might not even exist yet. As this is usually carried out without the knowledge of participants, they perceive a system as if a perfect technology would exist and allows for independent judgment, not influenced by errors an actual implementation might (still) have.

create a mock-up interface for users and enables remote control from a different machine in a different room for the experimenter.

4.3.1 Taxonomy

Figure 4.27: The interface for the user study.

Pointing gestures are typically used with large displays such as video walls (Figure 4.27). The target acquisition using a pointing gesture can be split into three distinctive phases: preparation, stroke and recovery [Ken04]. The stroke phase is when the target is actually acquired and the selection is made. All gestures discussed here are therefore meant to be executed in the stroke phase of the pointing gesture. Pre-

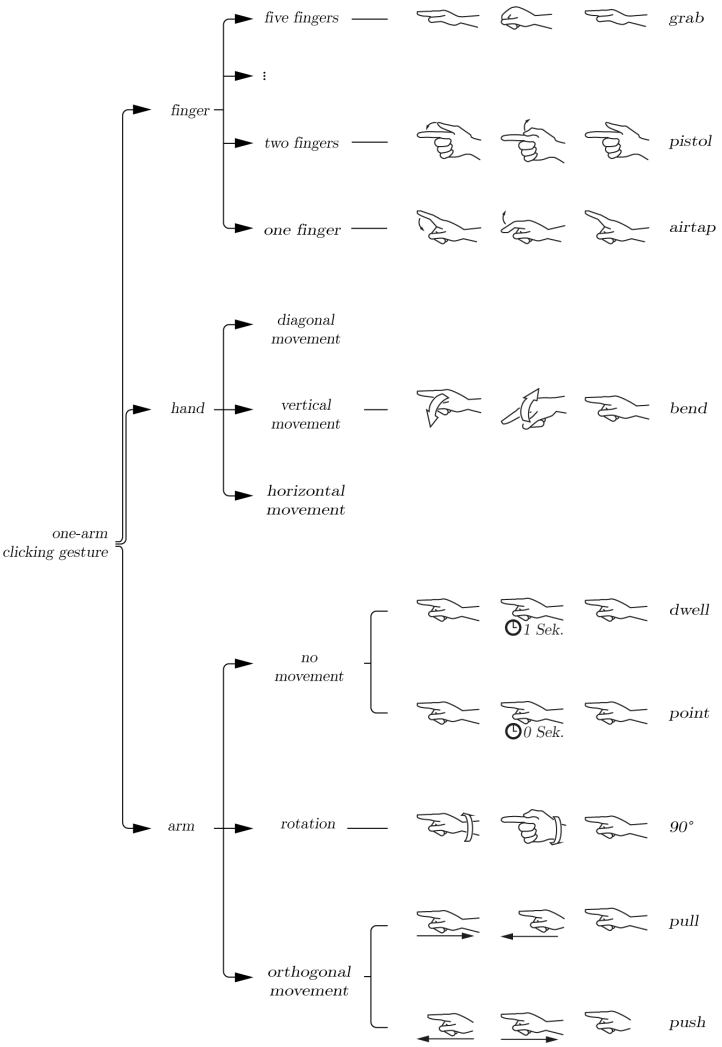


Figure 4.28: Taxonomy for distant one-arm clicking.

vious categorizations of one-arm gestures [Que94, Que95] divide possible gestures into very broad categories of deictic or manipulative gestures and do not capture the range of possible variations. Considering possible movements of a single arm, the first level of differentiation is the part of the arm that is involved in the gesture. This can be the whole arm, just the hand or multiple or individual fingers. Taking different numbers of fingers and the range of anatomically possible movements of hand and arm into account leads to a categorization depicted in Figure 4.28. For each leaf node of the taxonomy an example is given on the right of each leaf node. Each gesture is split into three consecutive phases that are shown from top to bottom. The click event is triggered in the last phase. If the whole arm is used for target selection, the possible movements are limited because it must not affect the pointing location. This reduces possible movements to orthogonal ones (towards and away from the display) and rotation. Not moving the arm is the third option.

The hand alone does not offer much variation and gestures for triggering a selection are limited to bending the hand either vertically, horizontally or diagonally.

The fingers offer many more options and to characterize them, they are divided by the number of fingers involved in a gesture. Since preliminary experiments showed that the use of three or four fingers for a click gesture was perceived as unnatural and awkward, no examples for these two options are given in the taxonomy in Figure 4.28.

The main criteria used for differentiation in the taxonomy is the body part primarily involved in executing the gesture

and the type or direction of movement. The first criterion heavily affects the difficulty of implementing a recognition system, as larger body parts are easier to detect, and also affect the physical strain on the user's body, as finger movements usually cause less fatigue than arm movements. The second criterion mostly affects the physical strain, as some movements are more common and therefore more natural and familiar to the user. While each leaf node of the taxonomy proposes an individual gesture, it would be possible to combine multiple gestures for a more complex input. This would, however, require more coordination and result in gestures too complex for such a common operation as clicking.

4.3.2 Evaluation

While the taxonomy introduced above gives an overview of the possible options for device free clicking, it is not immediately apparent which gestures work best. To evaluate the gesture types proposed by the taxonomy, one example was selected for each leaf node (as depicted beside the nodes in Figure 4.28). The gestures used, that involve the whole arm, were push and pull, 90° rotation, point, and dwell. The delay chosen for the dwell timer was one second, as this is the delay used by the most prominent device using a dwell timer, the Microsoft Kinect, and has proven to be a good trade-off between preventing accidental clicks and tolerability. The pointing gesture is just like a dwell timer with no delay, triggering a click as soon as the arm movement stops. In preliminary experiments, both horizontal

and vertical hand movements were not well perceived. For the hand based gestures, a vertical bending of the hand was used instead. As mentioned before, three and four finger gestures were often perceived as awkward, which is why the finger based gestures used for the evaluation were one, two, and five finger movements: airtap, pistol and grab. As the pistol gesture involves all five fingers, but only two fingers are actively used to define the selection, it is categorized as a two finger gesture. Not moving a finger is similar to the point and dwell gestures and was omitted.
The problem with evaluating gestures that rely on a technical system for recognition is that there will always be a bias, introduced by the accuracy of the detection for each gesture. Gestures might be perceived as bad by users, only because the recognition rate for the gesture is worse than for others. The only way to achieve a fair comparison is if all gestures work equally well from a technical standpoint. This problem is overcome by using a Wizard-of-Oz setup [Kel83], where participants are interacting with a pretense system that is controlled by a hidden human experimenter. This way, users can experience every gesture as if a perfect recognition would exist, which allows to draw conclusions about the preferred gestures independent of an actual implementation.
All techniques were evaluated in front of a large video wall, where a user interface was displayed on the right half of the wall (Figure 4.27). The video wall has a size of $4m \times 1.5m$ and a height of $2.37m$ with a resolution of $4096px \times 1536px$. The eighteen participants (13 males) were aged from 20 to 64, two of which were left-handed. Each participant was

presented with all nine gestures, the order in which the gestures were used was randomized for each user. The task for users was to click buttons that appeared in succession on the video wall at random locations within the user interface. The actual clicks that caused the current button to disappear, were triggered by an experimenter in another room observing the participant via a live video feed. By carefully observing the scene, it was possible to only trigger click events when the participants were actually pointing at the button (which some did not to test the system limits). The perceived system reaction time was minimal and only affected by the latency of the cameras and the reaction time of the hidden experimenter. The exact pointing direction, however, did not influence the accuracy and allowed the use of a single button size of $13.6cm$ for all experiments. For each new gesture, users were allowed to try the gesture for as long as they wished. When users were ready, they had to click 25 buttons. Whenever a button was clicked, the next one automatically appeared. While the order of appearance of the buttons was randomized, they were equally distributed across the screen. In addition to the experimenter in the other room, another experimenter was with the participants at all times and guided them through the whole experiment. After each gesture, participants were asked to rate the gesture using a NASA TLX based questionnaire (see Appendix A.4). The NASA TLX questionnaire aims at evaluating the mental, physical and temporal demand, overall performance, frustration level and effort. Participants were asked to give their ratings for each of these categories on a 7-point Likert scale. The questionnaires were

presented after the use of each gesture to make sure that the users' impressions were still detailed and present. After the use of all gestures, users were asked for additional, general comments and to select which gestures they liked best from the perspective of having used all gestures. To get a ranking of all gestures, users were asked to order the gestures from 1 to 9, from their most to their least favored method.

The ranking allows to get an overview of the results, as it summarizes the overall perception of the participants. The results of the NASA TLX questionnaires focused on the physical and temporal demands and added more detail to this overview, while the other factors evaluated show less influence on the overall preference but are in line with the overall ranking. This is most likely due to the fact that the TLX questions are tailored towards more complex tasks. While physical and temporal demands can be directly related and are meaningful for the simple task of clicking, the other categories like mental demand do not fit as well. Table 4.2 shows the mean values as well as standard deviation for each gesture and the pair-wise significance comparisons. Tables 4.3 and 4.4 show results for physical and temporal demands, respectively. The Wilcoxon rank sum test was used, as the data did not follow a normal distribution in all cases. A t-test did, however, show comparable results with a significance level of 0.05.

	airtap	point	pistol	90°	bend
μ	2.78	3.00	4.61	4.89	5.11
σ	1.93	2.13	2.16	2.26	2.38
airtap	-	0.95	**< 0.01**	**< 0.01**	**≪ 0.01**
point	0.95	-	**0.02**	**0.02**	**0.01**
pistol	**< 0.01**	**0.02**	-	0.62	0.54
90°	**< 0.01**	**0.02**	0.62	-	0.78
bend	**≪ 0.01**	**0.01**	0.54	0.78	-
grab	**≪ 0.01**	**< 0.01**	0.41	0.75	0.94
dwell	**≪ 0.01**	**≪ 0.01**	0.30	0.54	0.72
push	**≪ 0.01**	**≪ 0.01**	**0.03**	0.10	0.18
pull	**≪ 0.01**	**≪ 0.01**	**≪ 0.01**	**≪ 0.01**	**≪ 0.01**

	grab	dwell	push	pull
μ	5.28	5.39	6.22	7.72
σ	2.28	2.31	2.15	1.19
airtap	**≪ 0.01**	**≪ 0.01**	**≪ 0.01**	**≪ 0.01**
point	**< 0.01**	**≪ 0.01**	**≪ 0.01**	**≪ 0.01**
pistol	0.41	0.30	**0.03**	**≪ 0.01**
90°	0.75	0.54	0.10	**≪ 0.01**
bend	0.94	0.72	0.18	**≪ 0.01**
grab	-	0.87	0.28	**< 0.01**
dwell	0.87	-	0.28	**≪ 0.01**
push	0.28	0.28	-	**0.04**
pull	**< 0.01**	**≪ 0.01**	**0.04**	-

Table 4.2: All gestures were ranked by participants on a Likert scale from 1 (liked best) to 9 (liked worst). The table shows the results from left (best) to right (worst) with mean and standard deviation in the top rows and the significance analysis results (p-values) for the pair-wise comparisons below (using the Wilcoxon rank sum test).

	point	airtap	90°	pistol	bend
μ	-2.83	-2.28	-1.39	-1.11	-1.11
σ	0.37	0.93	1.70	1.63	1.49
point	-	**0.04**	**$\ll$ 0.01**	**$\ll$ 0.01**	**$\ll$ 0.01**
airtap	**0.04**	-	0.15	**0.03**	**0.02**
90°	**$\ll$ 0.01**	0.15	-	0.55	0.47
pistol	**$\ll$ 0.01**	**0.03**	0.55	-	0.95
bend	**$\ll$ 0.01**	**0.02**	0.47	0.95	-
push	**$\ll$ 0.01**	**$<$ 0.01**	0.31	0.59	0.68
dwell	**$\ll$ 0.01**	**0.04**	0.48	0.85	0.85
grab	**$\ll$ 0.01**	**$<$ 0.01**	0.22	0.47	0.47
pull	**$\ll$ 0.01**	**$\ll$ 0.01**	**0.02**	**0.05**	**0.05**

	push	dwell	grab	pull
μ	-0.89	-0.89	-0.67	0.00
σ	1.45	1.94	1.73	1.49
point	**$\ll$ 0.01**	**$\ll$ 0.01**	**$\ll$ 0.01**	**$\ll$ 0.01**
airtap	**$<$ 0.01**	**0.04**	**$<$ 0.01**	**$\ll$ 0.01**
90°	0.31	0.48	0.22	**0.02**
pistol	0.59	0.85	0.47	**0.05**
bend	0.68	0.85	0.47	**0.05**
push	-	0.82	0.68	0.09
dwell	0.82	-	0.68	0.13
grab	0.68	0.68	-	0.28
pull	0.09	0.13	0.28	-

Table 4.3: Participants were asked how physically exhaustive each interaction with the given technique was. They rated the techniques on a Likert scale from -3 (not exhaustive at all) to +3 (very exhaustive). The table shows the results from left (best) to right (worst) with mean and standard deviation in the top rows and the significance analysis results (p-values) for the pair-wise comparisons below (using the Wilcoxon rank sum test).

	point	airtap	bend	pistol	90°
μ	-2.94	-2.06	-2.00	-1.83	-1.78
σ	0.23	1.22	1.00	1.30	1.23
point	-	$<$ **0.01**	$\ll$ **0.01**	$<$ **0.01**	$\ll$ **0.01**
airtap	$<$ **0.01**	-	0.67	0.61	0.47
bend	$\ll$ **0.01**	0.67	-	0.89	0.69
pistol	$<$ **0.01**	0.61	0.89	-	0.82
90°	$\ll$ **0.01**	0.47	0.69	0.82	-
grab	$\ll$ **0.01**	0.14	0.23	0.33	0.41
push	$\ll$ **0.01**	0.20	0.25	0.33	0.40
pull	$\ll$ **0.01**	$\ll$ **0.01**	$\ll$ **0.01**	$<$ **0.01**	$<$ **0.01**
dwell	$\ll$ **0.01**	$\ll$ **0.01**	$\ll$ **0.01**	$<$ **0.01**	$<$ **0.01**

	grab	push	pull	dwell
μ	-1.28	-1.22	-0.28	0.06
σ	1.59	1.69	1.56	2.12
point	$\ll$ **0.01**	$\ll$ **0.01**	$\ll$ **0.01**	$\ll$ **0.01**
airtap	0.14	0.20	$\ll$ **0.01**	$\ll$ **0.01**
bend	0.23	0.25	$\ll$ **0.01**	$\ll$ **0.01**
pistol	0.33	0.33	$<$ **0.01**	$<$ **0.01**
90°	0.41	0.40	$<$ **0.01**	$<$ **0.01**
grab	-	0.97	0.07	0.06
push	0.97	-	0.10	0.06
pull	0.07	0.10	-	0.68
dwell	0.06	0.06	0.68	-

Table 4.4: The temporal demand of each technique was rated on a Likert scale from -3 (very short) to +3 (very long). The table shows the results from left (best) to right (worst) with mean and standard deviation in the top rows and the significance analysis results (p-values) for the pair-wise comparisons below (using the Wilcoxon rank sum test).

4.3.3 Discussion

The smooth descent from airtap as the best rated gesture to pull as the worst, immediately reveals the favorite among the evaluated gestures. An analysis of the pair-wise significance (Table 4.2) reveals additional insights. The gestures can be divided into three groups of similar ratings. This means, the ratings are comparable within the group but differ significantly from those of gestures in the other groups. The results of the physical and temporal demand ratings are in line with the ranking of the gestures and support this grouping. The highest ranked gestures, airtap and point, constitute the first group. The resemblance to a mouse click and the minimal effort required makes airtap a very intuitive gesture. It is no surprise that the pointing gesture is highly favored as well, as it requires even less effort. It has to be noted, however, that an actual implementation of a dwell timer with zero delay is very unlikely. As it immediately triggers a click when over a target, it is far too likely that clicks would be triggered accidentally. In addition, several participants pointed out that they missed the active control of triggering the click. It does show, however, that techniques that reduce the required dwell time, like the target prediction discussed in Section 4.1, are desirable. Considering these caveats, airtap should be used as the primary gesture for triggering target selections.
The second group consists of pistol, bend, 90°, grab, dwell, and push. All of these gestures were not as highly rated as those in the first group, but were often among those that users considered useful in everyday life. Many interfaces

are too complex to display all information available, which is why context information often has to be revealed with a secondary type of click. For the mouse this is typically the right mouse button. The gestures of the second group are candidates to fulfill this functionality for a pointing gesture based system. The gestures are rated well enough to be used, but not well enough for the most common task. As there are multiple gestures in this group, it would even be possible to assign to them several secondary functions or shortcuts to fully utilize their potential. The final and third group consists of only one gesture: pull. Its ratings were peculiarly bad and almost no user mentioned this gesture when asked which would be usable for a real application. The reason for this seems to be the counterintuitive motion of pulling the arm away from the display. Due to this finding, the gesture should not be used at all in an actual application.
It is striking, that the ranking follows the general observation that a gesture with less required effort results in a better overall rating. The first group of gestures add little to no additional strain to the always required pointing gesture, the pull gesture of the third group requires movement of the complete arm which, depending on the execution of the gesture, can even include the upper body. In between are the gestures of the second group that mostly require movement of multiple fingers or the whole hand. While no movement is necessary for the dwell gesture, the long delay during which the arm has to remain extended is tiring. Comparing the dwell gesture to the pointing gesture, the delay seems to play the key role in how the gesture is

perceived. While the gestures are identical except for the delay, the dwell gesture was rated much worse than the pointing gesture. Since the dwell gesture is used in several existing systems despite these results, its major advantage has to be pointed out: It is very easy to detect, which makes it very robust for systems that actually recognize the gesture and is comparably easy to implement. If recognition works well enough, however, most other gestures should be preferred. Almost all of the gestures have three distinctive phases that can be compared to a mouse click, which consists of mouse down, hold and mouse up. When clicking with the mouse, the hold phase is usually extremely short. A prolonged hold phase is actually used for an additional functionality: drag-and-drop. The similarity of the phases would allow to implement this functionality for most of the gestures as well. While the gestures in this evaluation were based on the Wizard-of-Oz concept for the aforementioned reasons, note that all gestures can be implemented in a real-world system; in fact, most of them are already available to different degrees of robustness at the Fraunhofer IOSB digital situation table [PBERG07]. As pointed out, the ranking of the gestures shows a smooth descent, which indicates that several gestures can be considered useful. This leads to the conclusion that gestures of the first group could be used for common operations, like clicking, because they were generally perceived as being faster and more efficient. Gestures of the second group would then be a good choice for less frequent but still common operations such as drag-and-drop or opening a context menu.

4.4 Inter-display Interaction

One of the goals of glueTK is to enable the creation of immersive multi-display applications. An important aspect of these multi-display applications is that interaction is not bound to a single screen and users may use the same input device to interact with different interfaces and different screens and also transfer data between them. Allowing for cross display interaction and data transfer allows to perceive a whole room, with all displays in it, as the interface and ignore technical details as well as the fact that there are different machines and applications involved to drive all displays.
However, enabling the abstraction from multiple machines is a technical challenge. Interfaces have to communicate, synchronize and exchange data over a network to create the illusion of a single coherent interface. No matter how fast network connections are, latencies can not always be avoided. Ever since computers became powerful enough, animations of various kinds have been used to enrich the user interface. It has been shown that animations can help to alter the users' perception of lags or latencies [Gon96, HS93]. Work by Miller et al. [Mil68] and later Card et al. [CRM91] provides a rough guideline for system response times:

- $100ms$ is about the limit for the perception of instantaneous reaction
- $1000ms$ is about the limit to keep the user's flow of thought uninterrupted

- $10000ms$ is about the limit to keep the user's attention focused on the current activity

The following sections present two user studies that investigate what the limits of acceptable latencies are, and also what can be done to bridge those latencies for the best possible user experience in the case of multi-display environments [Rot12]. A final conclusion will compare the results to the prior work.

4.4.1 Acceptable Data Transfer Latencies

As latencies can cause irritation and frustration, it is obvious that the elimination or reduction is an important goal. When it comes to reducing latencies, especially when complete elimination is not possible, it is important to know what kind of reduction actually has the desired effect, e.g. what latency is acceptable to users. The goal of the user study described below is therefore to find such a limit of latency for transferring interface elements across screens.
The study is set up by presenting participants with a task of sorting geometric shapes. Starting out with all shapes on a screen right in front of the user, rectangles have to be moved to the screen immediately to the left of the user and circles have to be moved to a screen to the right of the user in a distance of about $2.5m$ (see Figure 4.29 for an illustration of the setup). Using a mouse, the geometric shapes can be transferred to either adjacent screen via drag-and-drop by placing a shape in the corresponding transfer area, which is indicated by a vertical, dashed line on both sides. The

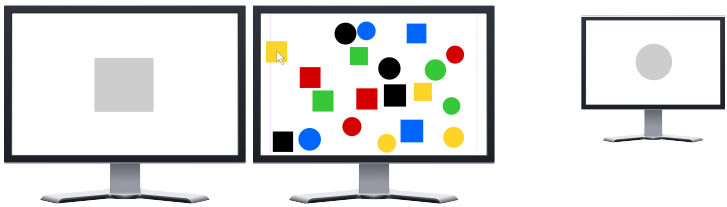


Figure 4.29: Initial user study set up for finding a limit of acceptable latency for inter-display data transfer.

shapes have different colors, which are associated with different artificially induced transfer latencies. Table 4.5 gives an overview of all colors and latencies. The latencies are independent of the shape, however. While each color corresponds to an artificial latency, the actual latency is longer as the actual system latency for transferring the shape has to be incorporated. This additional overhead equates to $18ms$ and is the same for any shape and target display. In each run, a user was presented with ten rectangles and ten circles at random positions on the center screen. Every participant had to complete four runs of sorting these shapes. Each set of shapes contained every color twice. Whenever a user transferred a shape to either screen, the interface was locked for the corresponding time. This required users to wait for the shape to appear on the target screen before the next shape could be transferred, and forced them to experience the latency instead of simply dragging the next shape while the previous one was still being transferred.
Ten male users, aged 22 to 32 (average 28), participated in the study and in addition to performing the task described

above, answered a questionnaire (see Appendix A.5) to get an insight into their subjective impressions.

Color	Blue	Green	Red	Yellow	Black
Latency	0ms	500ms	900ms	1200ms	1800ms

Table 4.5: Overview of how latencies correspond to colors.

Table 4.6 shows the average rating of how acceptable the experienced latency was to users for the respective colors on a Likert scale form 1 (very fast) to 7 (very slow). This shows, that the limit of acceptance lies between $518ms$ (green) and $918ms$ (red). In addition to rating each color individually, users were also asked which was the slowest color acceptable to them, which results in an average acceptable limit of $640ms$ ($337.31ms$ standard deviation) for the latency when transferring data across screens.
Interestingly, there was no significant difference between the rectangles (nearby target screen) and circles (distant target screen). This is important in the context of applications in a multi-display environment, as it can not be expected that users are more forgiving when it comes to latency for displays that are further away.

4.4.2 Bridging Latencies

The user study described above gives an insight into what latency needs to be achieved to be acceptable for users.

Color	Blue	Green	Red	Yellow	Black
Average Rating	1.4	2.1	4.5	5.7	7.0

Table 4.6: Average user ratings of the perceived speed on a seven point Likert scale.

Since this is not always possible, it is important to investigate what options exist to bridge the difference between acceptable and actual latency. Animations are used in many interfaces today to improve the user experience, as they often help the user understand what is going on. When it comes to transferring objects across screens, a sudden disappearance on one screen and a reappearance on a different screen can be irritating. In addition, the animation itself gives the user something to do, which can alter the user's perception of time in a subtle manner. Because the animation takes time, it is possible that the animation will allow for a higher tolerance towards latency, as it is not perceived by users the same way. To test this hypothesis, seven different animations for cross display transfer were created and evaluated in a user study.

The first animation is *Shrink*, as depicted in Figure 4.30. In this animation, the object is moved to the edge of the origin screen and scaled down. As it reaches the edge, it disappears on the origin screen and appears on the target screen, where it expands to its original size. The *Slide* animation moves the object along a horizontal line to the edge of the origin screen, where it disappears. It then reap-

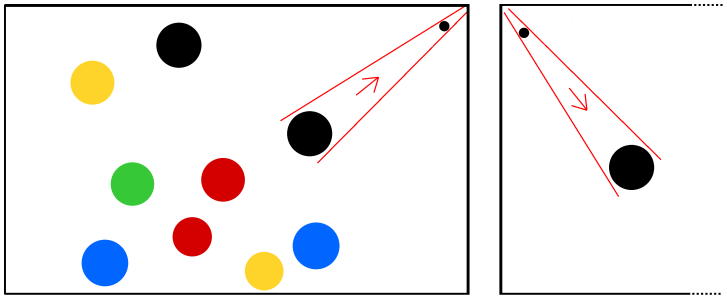


Figure 4.30: Shrinking animation for object transfer.

pears at the adjacent border of the target screen and slides along the horizontal line to a target position as show in Figure 4.31. By fading out the object on the origin display,

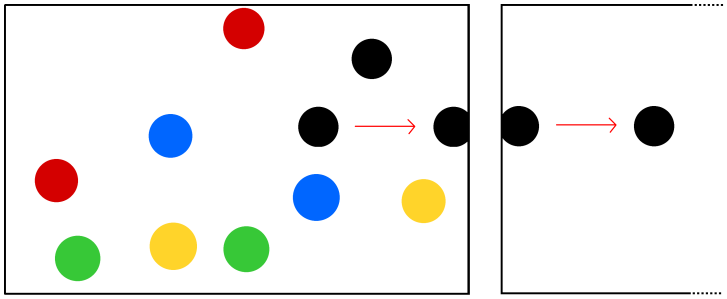


Figure 4.31: Sliding animation for object transfer.

the start of the transfer is indicated for the *Fade* animation (Figure 4.32). Right before the object becomes invisible, the object is faded in at a faster speed on the target display until fully opaque. Figure 4.33 illustrates the *Notification* animation, which shows a notification banner to let

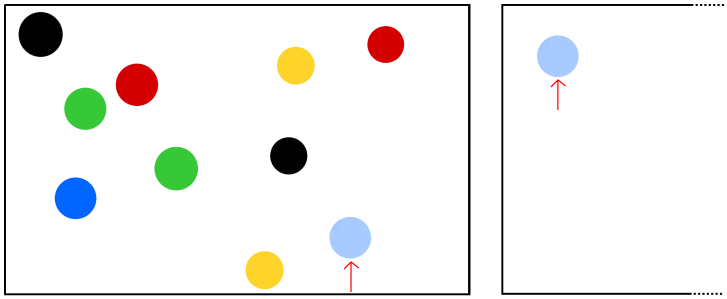


Figure 4.32: Fade in and out animation to indicate object transfer.

the user know about the ongoing transfer. The banner is hidden once the transfer is complete. Similar to the *Slide*

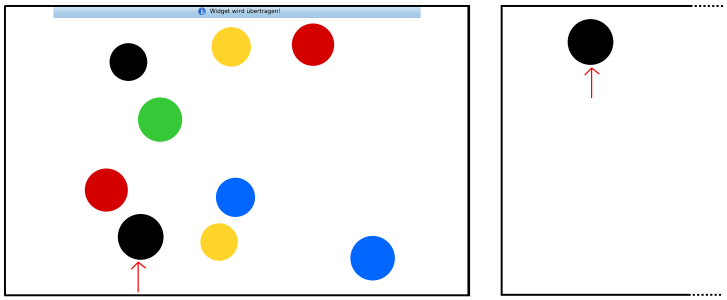


Figure 4.33: A notification is shown to let the user know the transfer is in progress.

animation is the *Slingshot* animation. Instead of directly moving an object across screens along a horizontal line, the object is first pulled in the opposite direction and upon "release" moved along the horizontal line to the target screen.

The movement is shown in Figure 4.34. The *Placeholder*

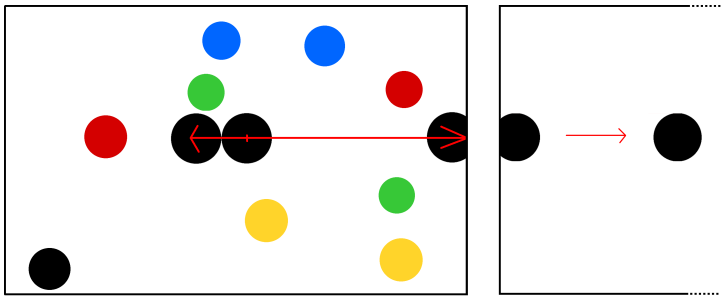


Figure 4.34: The slingshot animation slightly moves the object in the opposite direction before moving it quickly along a horizontal line.

animation is an atypical animation, as no movement is involved. Once the transfer is triggered, a placeholder image (as shown in Figure 4.35) is immediately displayed on the target display. Once the transfer is completed, the placeholder object is simply replaced by the actual object. The *Loader* animation shows a spinning circle as to indicate that the transfer is in progress. After the transfer, the object is immediately displayed on the target display and removed from the origin display along with the loader animation. The spinning circle directly displayed on the object to be transfered is shown in Figure 4.36. These animations can be divided into two groups. The first group of animations, *Shrink*, *Slide*, *Fade*, *Slingshot* and *Placeholder*, animate the object on the origin display as well as on the target display, which requires the object to be completely transferred before the animation can be shown on the target display. Such

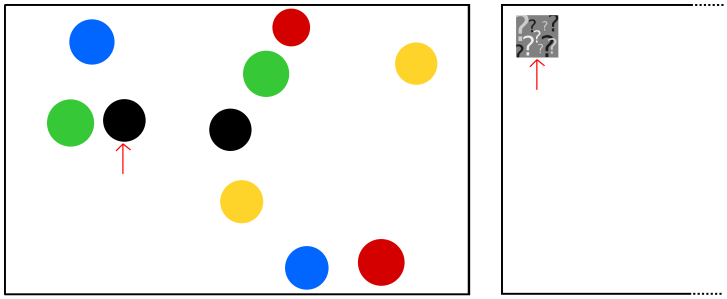


Figure 4.35: A placeholder image is used to indicate where an object will show up before it is transferred.

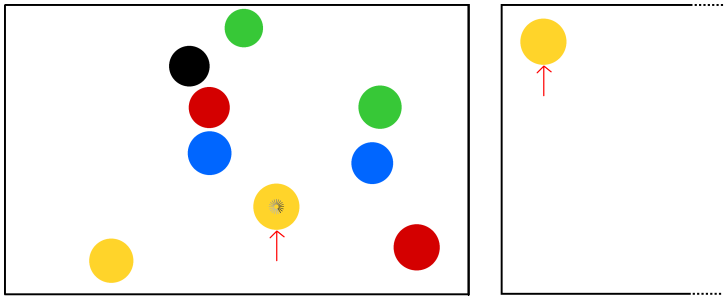


Figure 4.36: A loader animation is shown on the object while its data is being transfered.

an animation after the transfer comes at the cost of a higher latency but can be beneficial for the comprehensibility. All animations from this first group took 2.5*sec* to complete. The second group, consisting of *Notification* and *Loading*, can be stopped immediately when the transfer is complete and therefore require less time in comparison. Both were

configured to require $2sec$.
To evaluate if animations like the ones described above can assist in reducing the perceived latency, the user study was set up as follows: Two screens were set up next to each other with the task for the ten male participants to move all circles on the left screen to the screen on the right by simply clicking on them. Ten objects had to be transferred with each animation. While an animation was running, the user was not able to trigger further transfers. Users were asked to rate the different animations with respect to speed and intuitivity on a scale from one (very good) to five (very bad). The results are shown in Table 4.7. While it is not surprising that the ratings of intuitivity differ significantly (the p-value of the t-test between Shrink and Placeholder is 0.002), there are even significantly different ratings of the perceived speed within the first group of animations that all took the exact same time (t-test p-value: 0.036). The differences between the best and worst rated animations across both groups are significant for both factors as well: The t-test p-value between best (*Loading*) and worst (*Fade*) animation with respect to the rating of perceived speed is 0.013 and the t-test p-value between best (*Shrink*) and worst (*Placeholder*) with respect to the rating of intuitivity is 0.002.

In addition to rating the different animations on a Likert scale, participants had to order them from most to least preferred, which resulted in the following order: *Shrink*, *Loading*, *Slide*, *Notification*, *Placeholder*, *Fade*, *Slingshot*.
Two interesting conclusions can be drawn from these results. First of all, several animations have been rated well

Animation	Speed	Intuitivity
Shrink	2.5	1.6
Slide	2.7	1.9
Fade	3.2	2.5
Notification	2.6	2.4
Slingshot	3.1	2.3
Placeholder	2.8	3.0
Loading	2.4	2.2

Table 4.7: Animations rated with respect to speed and intuitivity on a five point Likert scale.

with respect to the perceived speed even though the transfer delay was well above the maximum acceptable latency discovered in the previous user study for all animations. This is in line with observations made in many everyday software products as well as websites, where animations are used to distract from delays. Second of all, even though the *Shrink* animation took 0.5 seconds longer than the *Notification* animation, there is no significant difference in their speed rating (t-test p-value: 0.377), the intuitivity of the *Shrink* animation, however, is rated significantly higher (t-test p-value: 0.024).
Besides these general observations, the feedback sections of the user questionnaire provided detailed insight into the participants' ratings. It was pointed out several times that the fact that the *Shrink* animation guides the users gaze made the operation easy to understand. The immediate feedback of the *Placeholder* animation was well perceived,

but the fact that there is no feedback on the origin display caused some irritations. The second group of animations, without visualization on the target display, was criticized for the missing feedback about where the object would show up after the transfer.

4.4.3 Conclusion

The limit of an acceptable latency for inter display transfer lies between $518ms$ and $918ms$, as the first user study has shown. This is in line with the guidelines for single screen system response times [Mil68, CRM91]. Transferring an object across screens is a more complex operation than a button click for example, which explains the acceptable limit to be above $100ms$. The upper limit of $918ms$ is just below the limit according to the guidelines to keep a user's thought uninterrupted.
For many interactions in the user interface, animations can improve the user experience as they can bridge occurring latencies [Gon96, HS93]. The second user study has shown that this is also the case for inter-display interaction. Animations are capable of altering the way users perceive delays in the interface and therefore provide important functionality above pure eye-candy. The importance of the design of static interface elements, especially their size, is the focus of the next section of this chapter.

4.5 Choosing Element Sizes

The approach of glueTK for creating interfaces and widgets is to offer basic blocks that allow for the easy creation of custom interfaces tailored towards a specific system. While this approach offers great flexibility, it does not dictate any layouts or element sizes. Specifically the element size is an important property when designing interfaces for novel interactive systems. Developers need to know what element sizes will be usable for a target system.
For a long time, there was little variation in the properties of computer systems. Developers of applications, just like users, used desktop computers with a mouse and keyboard for input and a screen within a small range of physical sizes and resolutions. The first widespread deviation from this homogeneity were smartphones and tablet computers. Not only do these systems use touch input, which is less accurate than a mouse, but the display sizes also differ greatly from those of desktop computers which are still used by developers to create applications for these new systems.
The two most dominant representatives, iOS and Android, of this new class of systems have rudimentary solutions to the problem of different display sizes and resolutions, not only between a developer's system and the target system, but also between multiple target systems like smartphone and tablet. The approach of iOS is to design a specific interface for each device class. It is then possible to bundle multiple, manually designed interfaces in a single application and depending on the system that runs the application, the corresponding interface is automatically used [App].

The device landscape for Android is a lot more diverse, which is why creating custom interfaces for each supported screen size and resolution is not an option. Instead, Android allows developers to define the size of elements as density-independent pixels (dp). These are a virtual pixel unit that abstracts from one of the parameters that differ between devices: the display density, which is dependent on the physical size and resolution. This way, interface elements, can be scaled on devices with different display densities automatically [And]. The solutions, however, only work in the class of mobile devices and only account for different displays, not different input modalities.
GlueTK on the other hand supports a much larger range of display devices, especially with very different input modalities. The goal is to find a solution for any interactive system, that means both output and input, display and corresponding input device. Also in the case of glueTK, a developer will usually use a different display and input device for development than the application will later run on. Because of this, a developer can not develop a feeling for the right sizes of elements but needs a way to know up front what element sizes work for a target system.
Different element sizes will affect the time it takes a user to select an element. To accurately describe this influence, the following terms will be used throughout this chapter. We define the *interaction performance* as the total time it takes a user to select an element, from the initial intention to locating the element and successfully selecting it. This overall time is split into two distinctive phases. We define the time it takes for a user to visually perceive and

locate an element as the *perceptual performance*. After locating an element the user has to perform a selection or activation using an input device. We define this share of the time of the overall interaction performance as the *input performance*. The following sections describe how to find usable element sizes for any interactive system consisting of a screen and an input device. The first section will deal with the perceptual performance, addressing different display properties, and the second section will deal with the input performance to show how the performance of different input devices can be matched. The third section analyzes the combination of perceptual and input performance to control the overall interaction performance by adjusting the sizes of interface elements.

4.5.1 Perceptual Performance

The perceptual performance describes how well visual information can be perceived. This includes both the search speed when localizing elements as well as the speed of recognition when visually selecting the desired target. For information displayed on a computer screen, this does not only depend on the visual acuity of the user and the distance of the display to the user, but also on the physical size of elements and the display's resolution. For text, these properties can be used to calculate font sizes which will yield good legibility. Equation 4.9 from ISO 9241-303 [ISO11] shows how the optimal font size h_r is affected by the distance to the screen d_r and the visual acuity δ, which is 0.3° for normal sighted persons.

$$h_r = 2 \cdot tan(\frac{\delta}{2}) \cdot d_r \tag{4.9}$$

The physical size can be converted to the corresponding pixel size using the display's size and resolution. Given a fixed distance of use, this allows to calculate optimal text sizes for any display. Interfaces are, however, not only made up of text elements but also of graphical elements, e.g. icons. Both have their respective advantages [BT93, Wie99], which is why most interfaces make use of both types of elements. For text, the main factor of legibility is the effective visual acuity, e.g. how well two parallel lines can be distinguished at the given distance. Graphical elements, however, are often much more complex than text, and many additional factors play a role in the perceptual performance. Some of these factors are: contrast, level of detail, context, and the user's familiarity with the elements. As these factors are not only hard to measure but also person specific, it is infeasible to find a generic equation to calculate an optimal size for graphical elements factoring in all of these parameters. As a solution to this problem, the concept of a reference system is introduced in the next section.

Reference System

The idea of a reference system [vdCSS13b] originates from the fact that despite the multitude of new devices extending the range of available displays and input modalities, developers still tend to use traditional desktop computers with

keyboard and mouse input for development. The problem that arises from the fact that developers and users do not use the same systems has already been described above. To close the gap between these different systems with their specific properties, the two need to be put in relation. If developers know how element sizes on their system relate to a target system, it is much easier to design interfaces that will work on that target system and also to spot potential problems before testing on the actual target system. The great advantage of using the concept of a reference system is not only the intuitivity for developers, but first and foremost the fact that only the parameters that change between systems need to be compensated for. The soft factors of graphical items described above, like context and familiarity, stay the same and do not have to be taken into account. If these factors can be ignored, it is a valid hypothesis that Equation 4.9 can be used to calculate the required visual acuity to perceive any element on the reference system by solving it for the visual acuity δ as shown in Equation 4.10a. This way, the visual acuity required to perceive the element as well as the developer intended is known. By inserting it in Equation 4.9 the element size required on the target system h_t with its associated distance of use d_t can be calculated as shown in Equation 4.10b. This can be further simplified to Equation 4.10c because of the periodicity of the tangent. The goal is essentially to preserve the visual properties of each element by adjusting its scale to match those of the reference system, while all parameters not accounted for stay the same.

$$\delta = 2(\tan^{-1}(\frac{h_r}{2d_r}) + \pi * n),\ n \in \mathbb{Z} \tag{4.10a}$$

$$h_t = 2 \cdot \tan(\tan^{-1}(\frac{h_r}{2d_r}) + \pi * n) \cdot d_t,\ n \in \mathbb{Z} \tag{4.10b}$$

$$h_t = \frac{h_r d_t}{d_r} \tag{4.10c}$$

Experimental Evaluation

To evaluate the validity of the hypothesis that the reference system allows to adjust element sizes to retain perceptual performance on a target system not only for text but also for graphical items, a user study was conducted. A 24 inch LCD monitor was used as a reference system and a Nexus 7 tablet computer and a Galaxy Nexus smartphone were used as target systems. After selecting a text size as well as a size for graphical elements on the reference system, both were scaled using Equation 4.10c to achieve the same perceptual performance on the respective target system.
To measure the perceptual performance, a test setup suggested in ISO 9241-304 [ISO08] was used. The task described, involves the display of random characters following several constraints with respect to the layout and character frequency. In this random text, all occurrences of a target character have to be located and counted. The number of target characters is always the same but a different target character is selected for each participant. The average time it takes users to count all occurrences is the achieved

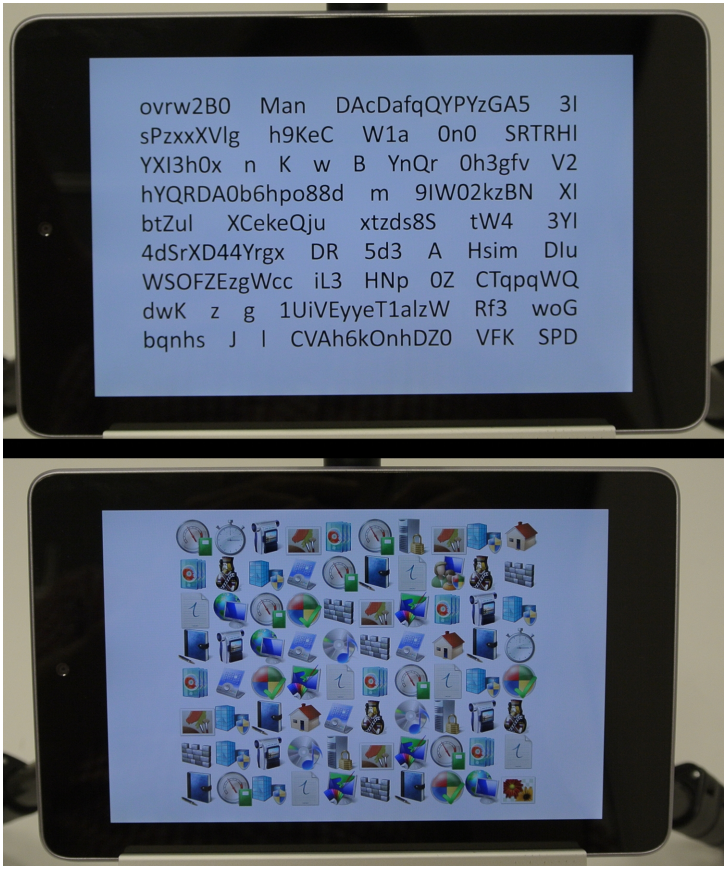


Figure 4.37: Example displays from the textual and graphical experiments on a Nexus 7 tablet.

perceptual performance. In addition to a task with text consisting of 306 random characters, a comparable test for

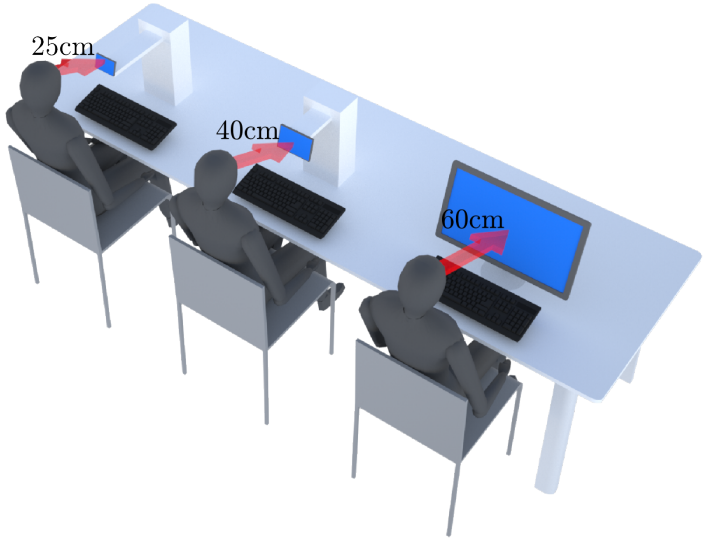


Figure 4.38: Set up of the user study with the three displays from left to right: Samsung Galaxy Nexus, ASUS Nexus 7 and Samsung 24" LCD display.

icons was designed. In this test, 80 random icons were displayed and users had to count the occurrences of a target icon. Again, the number of target icons was always the same but the target icon was different for every user. Figure 4.37 shows an example layout for both experiments. The reference system had a display diagonal of $60.96cm$ and a resolution of $1920px \times 1080px$ and a distance of use of $60cm$. The first target system was a Samsung Galaxy Nexus smartphone with a display diagonal of $11.81cm$ and a resolution of $1280px \times 720px$, used at a distance of $25cm$.

	LCD	Galaxy Nexus	Nexus 7
Characters	23,83	23,17	23,08
Icons	7,42	7,5	7,33

Table 4.8: Average durations for both experiments for each display.

The second target system was an ASUS Nexus 7 tablet computer with a display diagonal of $17.78cm$ and a resolution of $1280px \times 800px$ at a distance of use of $40cm$.
The twelve (ten male, two female) participants, aged from 14 to 52, all had normal or corrected to normal sight. Each participant was assigned a random target character and icon and was asked to count the occurrences on every display. Figure 4.38 illustrates the set-up of the experiment. The order of displays was randomized for every user. Table 4.8 shows the average time it took to locate all targets in seconds for both text and icons for all systems. The results show that the performance on all three systems is almost identical. The aberration between the average time it took users to locate and count all occurrences of a target is less than one second for text as well as icons. This difference is not significant as the one-way ANOVA [Win70] results presented in Table 4.9 show. These results confirm that the concept of the reference system, along with the adapted equation to calculate element sizes for target systems, allows for retaining the perceptual performance of the reference system on a target system.

	Source	SS	df	MS	F	p
Text	Between	4.05	2	2.02	0.05	0.95
	Within	1436.25	33	43.52		
Icons	Between	0.16	2	0.0833	0.03	0.97
	Within	102.58	33	3.10		

Table 4.9: ANOVA results for the perceptual performance tests.

4.5.2 Input Performance

As described above, the complete interaction between a user and an interactive system consists of two distinctive parts. The first part, the perception, has been described above and with the introduction of the concept of a reference system it has been shown that the perceptual performance can be adapted for every display by choosing the right element sizes. The second part deals with the actual user input, utilizing the input device available for a given interactive system. The input performance has been studied in great detail. The most prominent result of this research is Fitts's law [Fit54], which allows for predictions of the input performance. However, Fitts's law and the many extensions to it focus on the prediction of the input performance and the take away for most developers is the simplified conclusion "bigger is better" when it comes to interface elements. Of course, it is true that bigger targets are easier to hit and that this is an implication of Fitts's

law. For real world applications, however, there is a limit to the maximum size of interface elements as there is only limited screen estate. The more complex applications become, the more interface elements need to be incorporated into the interface. Therefore, the goal has to be to find element sizes that allow for a certain input performance but not more to still allow for meaningful user interfaces. Solving Fitts's law (Equation 4.11) for the size of the interface element w results in Equation 4.12. This defines the element size depending on the parameters a,b,d and t. The parameters a and b are empirically determined for an input device and are therefore easy to obtain.[2] The distance to the element d as well as the movement time t are variable. Especially the distance depends on the location of the element and the current position of a cursor. To find a generic solution to this problem, we define the distance d as the diagonal of the display for the following rationale. Movements are always thought of as direct movements in the context of Fitts's law. This means, that the diagonal of the display is the maximum possible distance for a given interactive system and therefore the worst possible case. If the diagonal is used as the distance in Equation 4.12, the time t can intuitively be selected as the upper boundary for the time a single input action should take. By selecting this upper boundary, an element size can be calculated that

[2]To determine the parameters a and b, the time it takes to click several targets of varying size and distance is measured. In a coordinate system with time on the y-axis and the index of difficulty ($\log_2(\frac{2d}{w})$) on the x-axis, a regression line is calculated from all measurements. The intercept of that line defines the parameter a and the slope defines the parameter b.

satisfies this requirement for the given interactive system.

$$t = a + b \cdot \log_2(\frac{2d}{w}) \tag{4.11}$$

$$w = d \cdot 2^{\frac{a+b-t}{b}} \tag{4.12}$$

The concept of the reference system is not only used to be able to transfer soft factors that influence the perceptual performance of graphical elements. It can also be used as a model of the way novel interactive systems are usually developed, that is on traditional desktop computers instead of the actual target system. To incorporate the calculation of element sizes with respect to the properties of the input device into the concept of the reference system, the following shows how input performance can be retained from a reference system to a novel interactive system using the adaptation of Fitts's law using the display diagonal as a fixed value for the distance. Equation 4.13 shows how how Fitts's law can be used to calculate the maximum input time t_r for an element on the reference system using the display diagonal d_r as the distance, the size of the element as defined by the interface designer w_r and the empirically found parameters of the input device of the reference system, a_r and b_r. The goal for the target system is to find the element size that guarantees the same maximum input time t_r on the target system. Equation 4.14 shows how this can be accomplished by inserting the time t_r into the Fitts's law formula solved for the element size on the target system w_t. The use of the input device specific parameters

of the target system a_t and b_t as well as the diagonal of the targets system display d_t result in an element size with the same upper boundary for the input time on the target system as the original element on the reference system with its input device.

$$t_r = a_r + b_r \cdot \log_2(\frac{2d_r}{w_r}) \tag{4.13}$$

$$w_t = d_t \cdot 2^{\frac{a_t + b_t - t_r}{b_t}} \tag{4.14}$$

To evaluate this adaptation of element sizes based on the concept of the reference system, a user study was conducted in which participants had to perform the same task on three different interactive systems. The task involved selecting 300 randomly placed squares with the respective input device. The first system, which served as the reference system, was a $60.96cm$ LCD screen with a resolution of $1920px \times 1080px$ in conjunction with an optical mouse as an input device. The second system was a $132cm$ TV screen with a resolution of $1920px \times 1080px$ connected to a gyro mouse. A gyro mouse is an input device similar to a mouse but instead of moving it on a flat surface it is freely moved in the air to control the application's cursor. The third system was a SMART 685i3 board with a touch sensitive surface. The diagonal of the board is $221.3cm$ and the image projected onto it has a resolution of $1280px \times 800px$. The size for the squares was chosen to work well on the reference system with an edge length of $100px$. The respective edge lengths for the TV system and the SMART

	LCD + Mouse	TV + Gyromouse	SMART Board + Touch
Element size	100px	197px	27px
Avg. time	797.2	857.3	855.7

Table 4.10: Average input performance for each system in seconds.

board were calculated as described above. This resulted in $197px$ for the TV + gyro mouse system and $27px$ for the touch sensitive SMART board.
Ten users (8 male), aged 21 to 42, participated in the user study. Each user was asked to click all targets on each of the systems using the respective input device. Users were given time to familiarize themselves with each system and the order in which they used the systems was randomized. Users were asked to click all targets as fast as possible, but to balance speed and accuracy. For each system and user the time from the click on the first target to the click on the last target was measured.
Table 4.10 shows the average time in seconds for each system. The differences between the three systems are not significant as the results of the one-way ANOVA test in Table 4.11 show. The results allow the conclusion, that the incorporation of the adapted Fitts's law, based on the display diagonal, into the reference system allows to retain the input performance from one system to another by making the proper adjustments to the size of the interface elements.

Source	SS	df	MS	F	p
Between	23456.06	2	11728.03	1.55	0.23
Within	203897.8	27	7551.77		

Table 4.11: ANOVA results for the input performance test.

4.5.3 Interaction Performance

As shown in the previous sections, by taking the relevant parameters into account it is possible to adapt user interface elements to yield the same perceptual performance on target systems as on a reference system. It was also shown how Fitts's law can be adapted to calculate element sizes to retain input performance from the reference system's input device to a target system's input device. However, to make predictions about the overall interaction performance, it is important to know how input performance and perceptual performance can be combined to calculate the overall interaction performance. While Fitts's law assumes direct and straight motions, a low perceptual performance could irritate the user and influence the movement and therefore the input performance. If the perception, which precedes the input, influences the input performance in any way, the two parts of the overall interaction are not independent and the interface adaptation becomes much more complex.
An experiment was designed to study if and how the input performance is influenced by a degrading perceptual performance. In the style of the previous experiments, a grid layout of 112 buttons was used, each with a single random

character in the center as a label (see Figure 4.40). Again, the task for users was to count the occurrences of a target character. The experiment was conducted in three variants. The first variant involved only counting the occurrences. The second variant required the participants to also click the target buttons with a mouse and the third variant required the users to touch the target buttons on the screen. The experiment was conducted eight times for each variant, each time with a different font size for the labels. The size of the buttons stayed the same at all times. The intention behind decreasing the font size was to provoke a degradation of perceptual performance. Adding an input task to the otherwise identical experiment in the other two variants had the goal of putting the perceptual performance in contrast to the input and overall interaction performance. To make this comparison possible, both the time for the whole experiment and the time used for input alone (mouse movement time / hand movement time) was measured[3]. A Samsung Nexus 10 tablet computer was used as a display (Figure 4.41), which has a $255.4mm$ screen with a resolution of $2560px \times 1600px$, to avoid any effects a low pixel density might have on the legibility of small fonts. By asking users to sit on a chair with their head leaned against a wall, a distance of $60cm$ from the eyes to the screen was ensured (Figure 4.39 shows the experiment's setup). The order in which participants were presented with the three variants was randomized. For each variant,

[3]Participants had to put their hand as well as the mouse back to a fixed position. The moment they left that position to the occurrence of the click was measured to determine the input time alone.

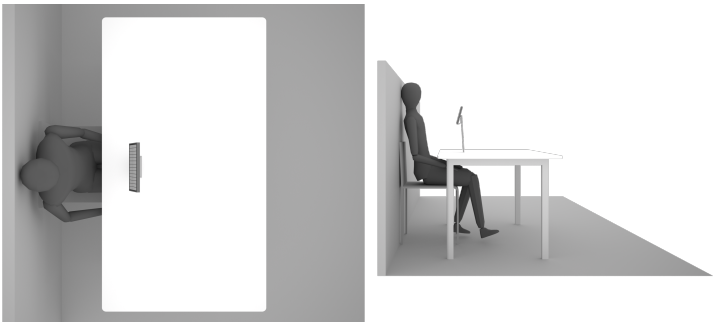


Figure 4.39: Set up for the user study on interaction performance. By asking users to lean their head on the wall a fixed distance to the display was ensured.

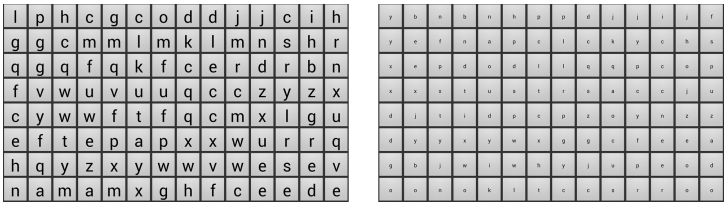


Figure 4.40: Example screen-shots from the user study on the Nexus 10 tablet using the largest font and the second to smallest font size respectively.

the order of font sizes (10.1mm, 8.5mm, 5.1mm, 3.4mm, 2.0mm, 1.4mm, 1.0mm,0.5mm) was randomized as well. Fifteen users took part in the experiment (12 male) aged from 21 to 42, all with normal or corrected to normal sight. All users were asked to take an eyesight test before the experiment to ensure normal vision. Figure 4.42 shows the

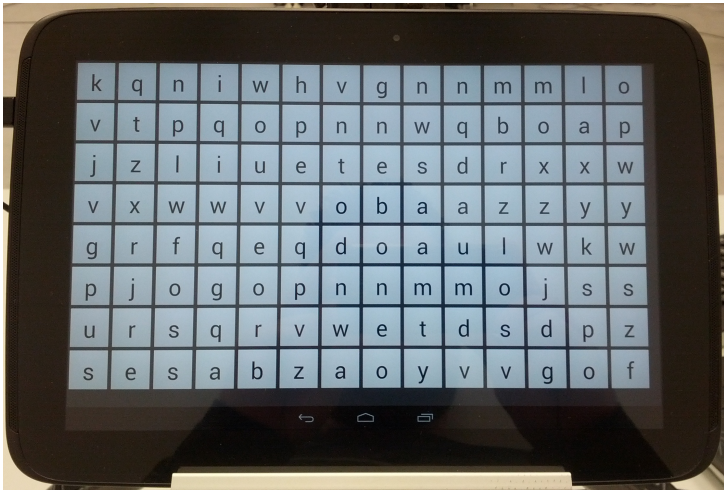


Figure 4.41: The Nexus 10 tablet used in the user study, displaying buttons of the same size with random letters as labels.

average durations of the experiments for all three variants and all font sizes. In addition, it shows the mouse and touch variants with the input time subtracted for each experiment. The graph shows that down to $3.4mm$, the performance does not degrade considerably. The optimal font size, calculated using Equation 4.9, is $3.1mm$. This is in accordance with the results, as it is around this point that the performance degradation begins. From that point on, the performance constantly degrades. While the smallest font size used had a height of $0.5mm$, none of the participants were able to correctly read the characters at this size. It is therefore safe to assume that between $1.0mm$ and $0.5mm$

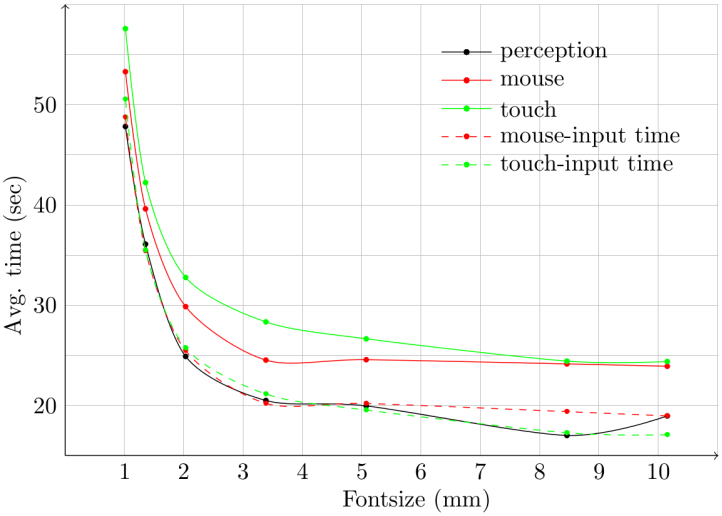


Figure 4.42: Average task durations for all font sizes.

lies the limit of human perception at the used distance of $60cm$. The fact that the input time subtracted mouse and touch curves are almost congruent to the perception curve, suggests that the input performance was consistent for all font sizes, which means that the input performance is independent of the perceptual performance. This insight allows to make predictions about the interaction performance by simple summation of input performance and perceptual performance.

The experiments described in the sections above have shown that the concept of a reference system enables developers to easily calculate element sizes for target systems. It is important to note that since perceptual performance and input performance are independent of each other, in most cases one of the two puts a larger constraint on the element size. Usually the bigger constraint originates from the input performance due to the low accuracy of many novel input devices compared to a mouse. And while the perceptual performance could have been matched with a smaller element, due the input performance, the final element will be bigger. The fact that most developers use a mouse as a pointing device, which is extremely accurate, results in a problem when it comes to retaining interaction performance. While it is possible to retain the performance for a single element from a mouse based reference system to most target systems by scaling the element up, this is not practical for real world applications as there is not enough screen estate for enough large elements to allow for sufficient application complexity. Retaining the performance of a mouse, however, is an overly ambitious goal for most novel devices. Their strengths do not lie in speed and accuracy but in new use cases and natural interaction, without the need to wear or hold any device. The reference system concept can aid the developer nevertheless, because it puts the developer's system in relation to the target system. Equation 4.13 is used to find the time to use as an upper boundary if the performance of the reference system has to be matched. Adjusting this upper boundary before inserting it into Equation 4.14 allows for an intuitive ad-

justment of how fast the interaction with the target system will be in comparison to the developer's system. Using the same equations, it is also possible to calculate the upper boundary of the input performance for an element size on the target system and therefore getting an intuitive feeling for the usability of the element on the target system.

5 Applications

This chapter will present two applications that have been created using glueTK. While many of the characteristics and features of glueTK have been described above, the way everything works together to enable new kinds of interactive systems is best demonstrated in an application context. The first application is a next generation control room, that utilizes multiple input modalities, from pointing interaction to person tracking, and several displays, from a smartphone to a video wall, to provide users with intuitive assistance in such a high pressure environment. The main focus of the technical perspective of these applications is

the integration of many different modalities as well as the multi-display interaction.
The second application is an interactive terminal for maritime situation analysis and consists of only three input technologies and two displays. The main focus of this application is to make use of the insights into element adaptation described in Section 4.5. Instead of adapting the interface to multiple interactive systems, however, the principle of the reference display is used to modify the screen content depending on the users' position, creating a distance dependent display.

5.1 Smart Control Room

Control rooms are centralized spaces from which missions are monitored and controlled. The military sector is a typical example for users of control rooms, but they are used in many branches of civil security forces and large industrial companies as well. Common tasks for the users in a control room, besides monitoring incoming information, is the planning and dispatching of emergency vehicles and relief units. Across the board, it is very common to make use of the continental staff system [Bmi09] in these rooms, which assigns each user with a specific role. This results in users requiring different information and often also different or specialized interfaces with functionality that supports their assignment.
Today's control rooms often incorporate large displays like video walls which are, however, mostly used as passive dis-

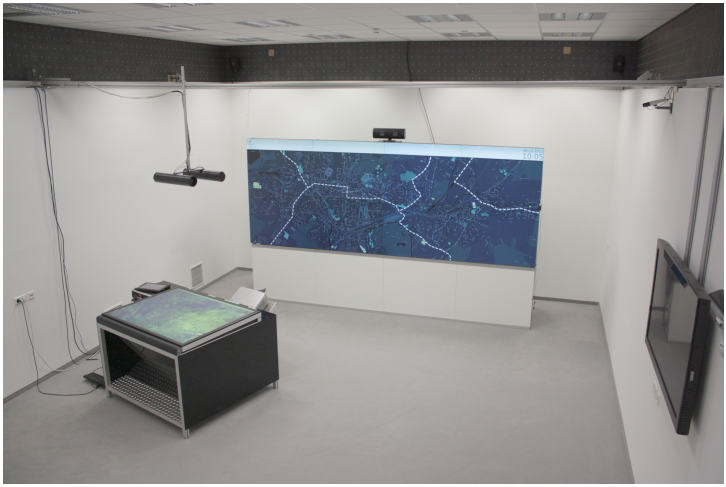


Figure 5.1: The Smart Control Room, including digital situation table and video wall.

plays and serve as an overview. Users usually only interact with personal desktop computers. Any manipulation of the content on the video wall has to be controlled from a desktop computer, which makes direct interaction with the video wall tedious. A controlled environment with strictly regulated activities and clearly assigned roles, such as a control room, is well suited to be enhanced by new technology as it is of manageable complexity. It has been shown early on [CJM+97] that technology can be of great assistance in a command and control environment, therefore the goal of this application is the creation of a next generation control room - a smart control room.
The idea of the smart control room is to allow intuitive in-

teraction with large displays by utilizing novel input technology. This interaction can be explicit or implicit, as some of the input modalities are not directly suitable for interaction but provide information that can be utilized to improve the user experience within the room, as will be shown in the following description. The room is depicted in Figure 5.1, where all stationary displays can be seen. These are a $4m \times 1.5m$ video wall with a resolution of $4096px \times 1536px$, which is intended to give an overview and to be used by multiple users at the same time, and a table display, which has a resolution of $1400px \times 1050px$ and measures $0.9m \times 1.2m$. In addition to these stationary displays, two mobile display devices are used: A motion computing j3500 tablet with a diagonal of $30.73cm$ and a resolution of $1280px \times 800px$ as well as a NOKIA N900 smartphone with a display diagonal of $8.9cm$ and a display resolution of $800px \times 480px$. Besides these two mobile devices, which constitute two separate machines, both the video wall and the digital table are driven by one computer respectively. Most of the input devices used are computer vision based and utilize nine Axis 211a cameras and one Logitech Quickcam Pro placed at different positions throughout the room. The first input technology is a commercial face recognition system [vid] that is able to identify persons who have previously been added to its database via the webcam. The webcam is attached to the right side of the video wall and allows persons to actively identify themselves by looking into it. This identification allows for personalized functionality and is especially useful because of the clear continental staff role assignment described above as roles can

be assigned automatically using the identity. To keep the identity of a person for a longer period of time and also to obtain the locations of all persons throughout the whole room at all times, a person tracker is used that relies on a single camera with a fisheye lens installed in the ceiling. Whenever the face recognition system recognizes a person, the identity is linked to the corresponding track of the person tracker closest to the webcam. A 3D reconstruction of the area in front of the video wall is created using two cameras on each side of the wall and a voxel carving approach, described in [SvdCIS09], uses this information to derive the pointing gestures of persons in front of the video wall. The digital table is equipped with a stereo camera setup above the display that allows for accurate hand gesture recognition in the 3D space above the display's surface as described in [PBGB09] and is from here on called digital situation table. The digital situation table is a back-projection display, which allows the installation of a camera looking along the axis of the projector, allowing to see the underside of the projection surface and locate MCMXT markers [GER+07]. By attaching MCMXT markers to the backside of the mobile devices, these can be be located and identified on the table when placed on its surface. Another camera, oriented towards the table, is used by a head pose estimation system [VNS07] to determine the line of sight of a person standing in front of the table.
With all the input devices and displays available, the following will describe an examplary course of events and how glueTK is used to utilize the available input as well as output to aid users in interacting with the smart control room.

As the video wall is intended for multi-person use and serves as an overview display for all users, it displays a complete map of a city. A person, S1, is standing at the digital situation table which displays a subsection of the map in more detail. A second person, S2, enters the room. While the application is aware of the new person due to the person tracker data, S2 is not able to interact with any display yet as all functionality is assigned to specific roles and S2 is not identified yet, and therefore not assigned a role. When S2 walks up to the webcam attached to the video wall, the face identification component detects that a person is present and compares the person to its database of known users. The component is handled by an event handler within glueTK, which triggers several signals during this process. The first signal that is sent out is: *faceid_identifying*. This signal is connected to a widget that is intended to give the user visual feedback about the ongoing identification process by first displaying a banner as shown in Figure 5.2. Depending on whether the person is known to the system, corresponding signals are emitted and result in according visual feedback of the same banner widget. In this example it is assumed that the person is known. This does not only inform the user about the successful identification (see Figure 5.3) but the corresponding signal *faceid_identified_S2* is also connected to a widget that manages personal workspaces and triggers the appearance of the workspace that belongs to S2 and can be seen in Figure 5.4. The workspace appears in front of S2 and follows S2 along the video wall. To enable this, the person tracker information is received by a correspond-

Figure 5.2: The user gets visual feedback about the progress of face identification.

ing event handler which passes the 3D room coordinates of persons along as signals. A context handler is connected to the trackers' signals and those of the face identification event handler. It uses this information to correlate identities with tracks and also converts the 3D room coordinates from the tracker to display coordinates of the display closest to each track. The person specific signal, containing pixel coordinates of the display closest to the person, is connected to the person's workspace. This causes the workspace to not only appear right in front of S2, it also causes the workspace to follow S2 along the wall so it is always within reach without explicit interaction.
To get an overview of various activities in the city, S2 has

Figure 5.3: The user gets visual feedback about the successful identification.

several tools at his disposal that can be selected from the personal workspace using a pointing gesture. The pointing gesture uses a dwell timer for triggering a click as described in Chapter 3. This requires the user to keep the cursor still above the targeted menu item. Since the position of the workspace depends on the user's position, movement can cause problems with the dwell timer as an item could move relative to the cursor. To avoid this, the context handler mentioned above is also connected to signals from the pointing gesture event handler. This way, it can lock the position of the workspace in place whenever the user points towards it. To correlate the pointing direction with the workspace position, the 3D pointing direction of the

Figure 5.4: The user selects an item from his personal menu using a pointing gesture, while the position of the menu is locked in place.

pointing gesture recognition system is converted to display coordinates in another context handler by intersecting the pointing direction with the surfaces of all known displays in the room. An example tool is a map overlay which displays a rectangular overlay of a different map layer around the current pointing position as shown in Figure 5.5. This allows S2 to get an overview, after which he walks over to the digital situation table to join S1. At some point between the two displays, S2 will be closer to the table than to the wall. The signal that controls the position of the workspace again just sets its position and it appears on the table and disappears from the wall. The fact that

Figure 5.5: A map overlay can be freely moved across the video wall by pointing to the desired location. A click discards the tool.

video wall and table are controlled by different machines, each running their own glueFrame, is not a problem as the element transfer happens automatically within the room-embracing glueTK application as described in Section 3.1.3. So while the workspace widget is actually serialized, sent over the network and recreated on the table while being deleted from the wall, it appears to the user as if the widget simply followed from one display to another. The same kind of transfer happens in the background when an alarm message is shown and both users walk up to the video wall to assess the situation (see Figure 5.6). They can both use the tools present in their respective workspaces to analyze

the current situation. One of the users attaches the alarm message to the personal workspace and takes responsibility for handling the task this way. When the users return to the table, the map is automatically moved and zoomed to the area the alarm message originated from. The table can then be used to plan the deployment of relief squads and the table content can be manipulated by using different hand gestures (see Figure 5.7). Which hand gesture triggers which functionality is a matter of configuration, e.g. appropriate connections between signals and slots.

Figure 5.6: Multiple users interacting with the video wall.

While S2 is interacting with the table, where the tablet computer can be used to make annotations, another alarm message is displayed. To make sure that S2 will see the message but at the same time not disturb other users, the message is only displayed on the table, as S2 is focused on it. When S2 looks up at the video wall, the alarm mes-

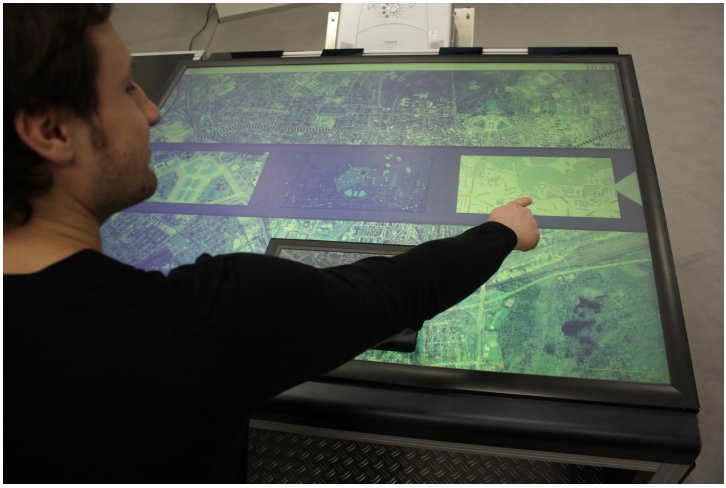


Figure 5.7: A hand gesture consisting of a single, streched out finger allows to trigger clicks and choose between different map layers, for example.

sage is displayed there. This way, the message is always within the users view. This is accomplished by utilizing the head pose estimation system. An event handler provides signals that indicate the focus of the user at the table and can be used to move the alarm message to the correct display. The user has to options to acknowledge the alarm message: Either select the message on the table or point towards it on the video wall. This is possible because the hand gesture recognition system is available throughout the glueApplication. It is physically connected to the machine that drives the table display due to spatial proximity, but the intersection points with all displays that are calculated

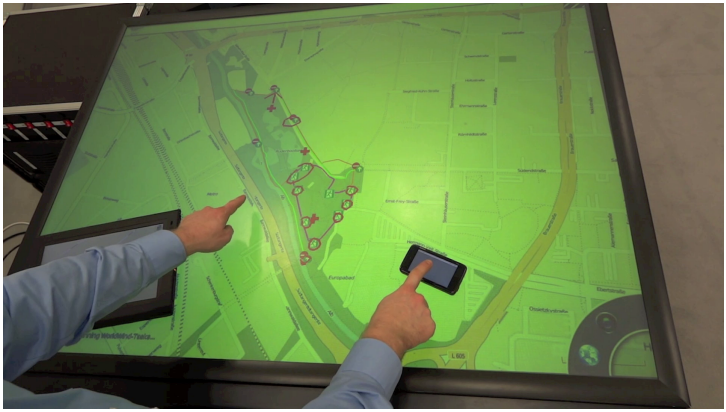


Figure 5.8: By pointing to the map on the digital situation table and the smartphone as well, the map data from the table is automatically transferred to the smartphone.

within a context handler are passed on by the proxy event handler of glueTK, which makes the signals available to all glueFrames.
Another example of the use of a remotely connected input device is the use of the smartphone. A third user, S3, enters the room with the smartphone described above. Using the internal gyroscope, which is made available as signals within the glueTK application, it is possible to control a cursor on the video wall. By simply touching selected images on the smartphone, they are transferred from the smartphone to the video wall and placed at the location of the cursor. This transfer is, unlike the following of workspaces, explicitly triggered by the user and animations are used to bridge the small latencies that occur. S3 then

places the smartphone on the digital situation table where it is localized and identified by the MCMXT marker on its back. This information, along with a hand gesture pointing with one hand to the map on the table and with the other to the smartphone, initiates a transfer of the map data to the smartphone.
The application was created with the intent to be a demonstration, not a real world application. All input, interface control and cross display and device interaction, however, are usable and work in real time. The application has been showcased many times, including exhibitions like CeBIT 2011, and has proven to work under these challenging conditions.

5.2 Distance Dependent Display

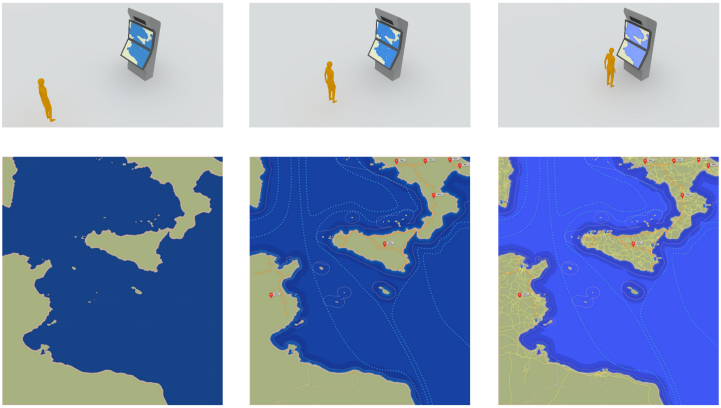


Figure 5.9: An illustration of the concept of a distance dependent display: The distance of the user (top) influences the level of detail used in the user interface (bottom).

The application described in this section is an interactive terminal for maritime situation analysis. One of the tasks in maritime situation analysis is the tracking and analysis of vessels in order to detect anomalies early on, e.g. pirate ships or refugee ships that do not identify themselves or try to actively spoof the identity of other vessels. While much of the analysis is done automatically using machine learning techniques, final analysis and decision by an operator is required. Thus he is presented with preprocessed, high-level data. An operator is usually assigned a certain geographical area to keep track of by reviewing different information about vessels in this area. The application consists of two

screens, in front of which a user can interact with the user interface using pointing gestures from further away as well as close to the display, and is tracked in this area up to a distance of about three meters.
To create a useful digital map interface, it is helpful to analyze people interacting with a printed map for comparison. They tend to move further away to get an overview and move closer to the map to see details.
For the digital variant, this behavior is exploited by modifying the interface on the display depending on the users distance to the screen. Section 4.5 has shown that distance is one of the key factors influencing the perceptual performance. The obvious idea of scaling elements depending on the distance is problematic as scaling would result in the need for rearrangement of interface elements from a certain scale factor on. There have been approaches to solve the rearrangement problem [Sea93, MP02], but due to the required context information this can not be accomplished automatically. The maritime situation analysis involves different tasks at different levels of details, so even if rearrangement of scaled interface elements was possible, it would not be the appropriate solution for this application. To exploit the printed map metaphor, the interface offers different functionalities and options at different distances, from an overview to detailed information. Unlike a real map, however, not all details are displayed all the time. The concept of the reference display described in Section 4.5 allows the accurate prediction of element sizes that are legible at a given distance and yield consistent perceptual performance. At a great distance to the displays

the user is presented with overview information as shown in Figure 5.11. As the user moves closer to the displays, more details are revealed as illustrated in Figure 5.9.
This adaptation matches the behavior metaphor of the interaction with a printed map and at the same time keeps the interface free of clutter, always assuring that what is displayed allows for optimal perceptual performance at the current distance.
At the same time, the distance dependent display helps to deal with limited screen estate, because instead of having to select information or layers explicitly using menus or similar methods, the subset of displayed information is automatically selected by the users' distance to the displays.
When modifying the interface automatically, it is not only important that it does not irritate users but also that users are able to discover this functionality. Because users are not explicitly triggering the display and hiding of layers, it needs to be communicated to them. To accomplish this, all information is spread over different layers that are associated with a different level of detail. Layers have a fade-in area, a visible area and a fade-out area as illustrated in Figure 5.10. This does not only prevent sudden changes or flicker that could irritate users when the distance and therefore the layers change, but is also useful for enticement.

When moving towards or away from the display, users get subtle hints about the adaptation and the ability to control the level of detail by their position. From about three meters away, which is the furthest distance that still allows

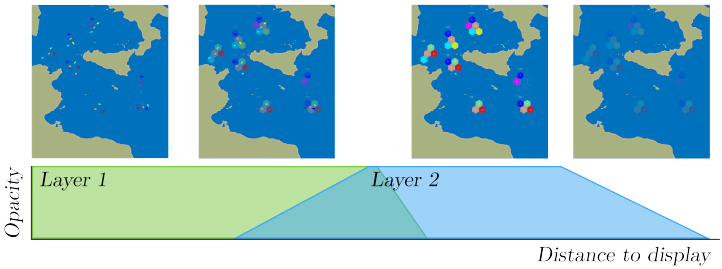


Figure 5.10: Layers are assigned a certain area in front of the display in which they are visible and fade-in/-out before and after this visible section.

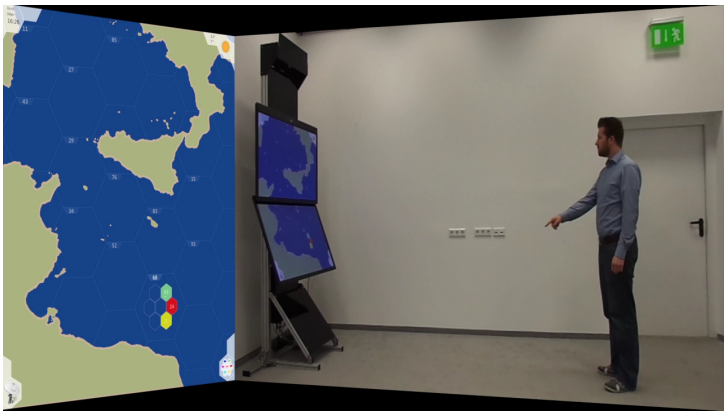


Figure 5.11: From further away, pointing interaction allows to get a rough overview. The inaccuracy of pointing from a distance and the large elements visible at this distance complement each other.

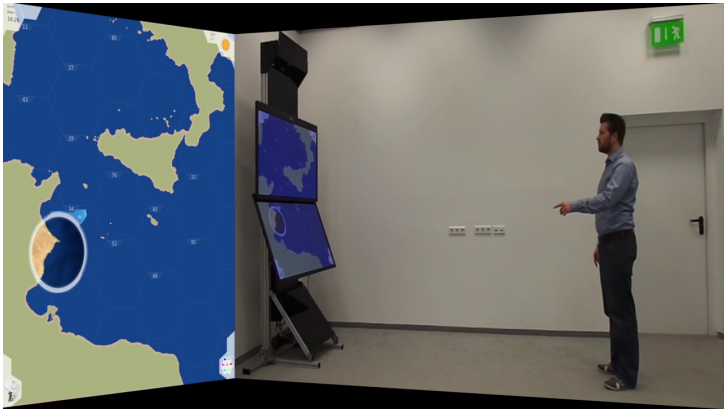


Figure 5.12: A magnifying map overlay allows to obtain addional information from a distance.

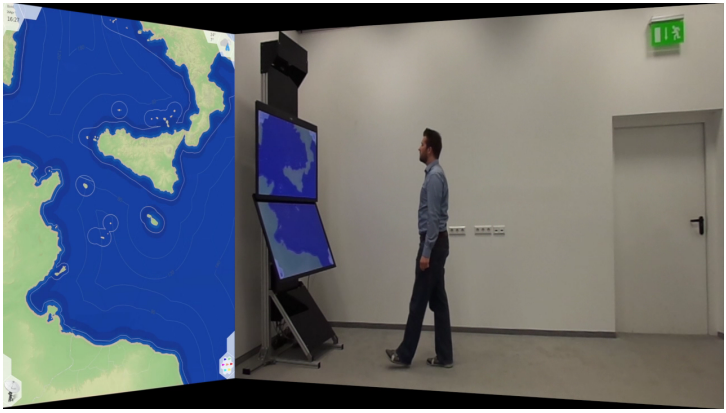


Figure 5.13: As the user approaches the display, more details are revealed.

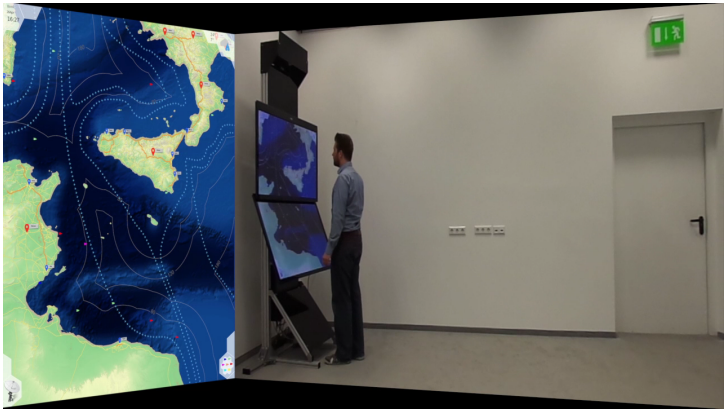


Figure 5.14: Right in front of the display a high level of details is shown, including individual vessels.

for accurate registration of position and pointing gestures, a user can get a rough overview of the displayed map region. The water surface is divided into hexagonal sections that only indicate how many vessels are in a particular section. When pointing to individual sections, more details are displayed as shown in Figure 5.11. This distance is best suited for getting an overview and therefore details are not yet visible. The tools available also reflect this, such as combined map overlay and magnifier shown in Figure 5.12. As the user moves closer to the display, some of the overview related information layers disappear and other layers with a higher level of detail appear, such as borders of territory or elevation data as depicted in Figure 5.13. Right in front of the displays, the level of detail is increased to show individual vessels (Figure 5.14). At this distance, even small

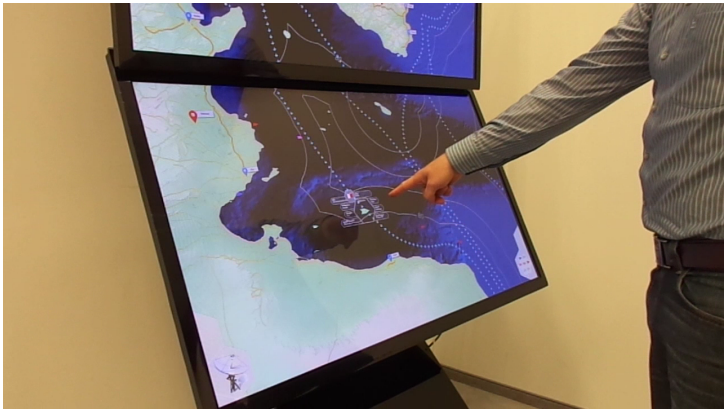


Figure 5.15: A fingertip detector allows pointing interaction with a single finger, accurate enough to select small items such as individual boats.

elements allow for a high perceptual performance, which is why the user can interact with individual vessels by pointing (Figure 5.15) towards or touching (Figure 5.16) them and get detailed information about the vessel's origin, destination, speed and identification data.

The technical setup includes two $121.92cm$ TV screens with a resolution of $1920px \times 1080px$, used as a single surface across both displays, ignoring the display borders that run across the interface. Two Microsoft Kinects [Kin] are installed above the screens as depicted in Figure 5.18. The Kinect on the bottom is oriented to look straight down, while the other is rotated into the room to observe the

Figure 5.16: The fingertip detector works up to the displays surface and allows pointing as well as touch interaction.

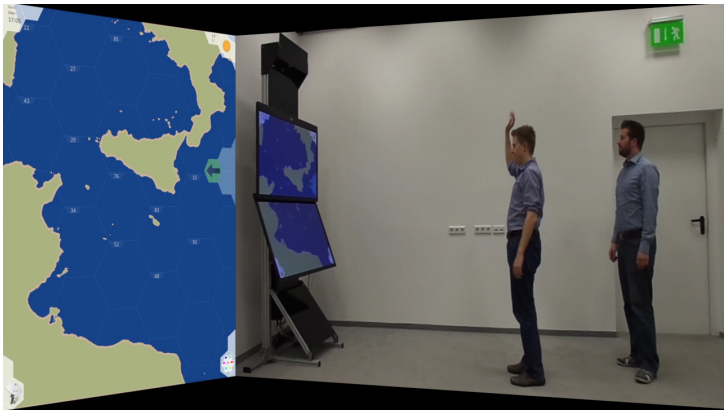


Figure 5.17: By waving, another user can take control of the distance dependent display.

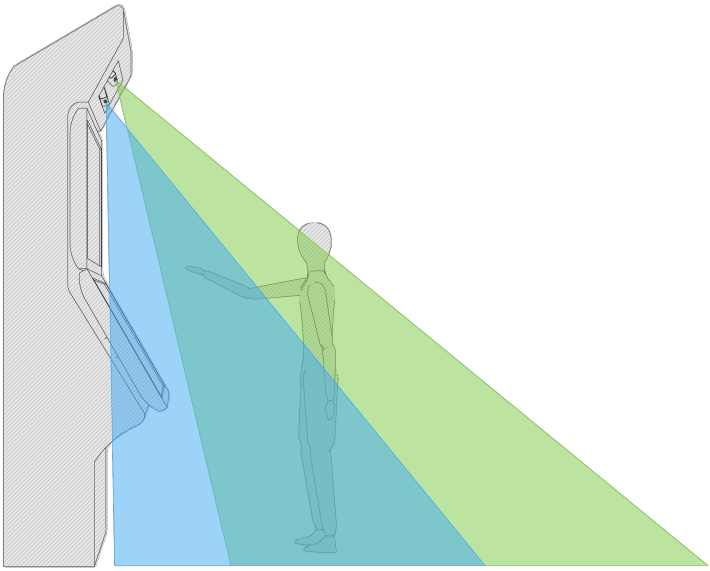


Figure 5.18: The technical setup of the distance dependent display along with the obverlapping fields of view of the two Kinect cameras.

area in front of the displays. In the field of view of the upper Kinect (indicated as a green area in Figure 5.18) the person is tracked, meaning the location and therefore the distance is known. In addition, the user can control a cursor using a pointing gesture. The system responsible for the pointing gesture recognition is based on OpenNIs' skeleton tracking [ope]. In the field of view of the bottom Kinect (indicated as a blue area in Figure 5.18), a fingertip detector [Hen11] is used to detect the pointing direction

more accurately. While the pointing gesture recognition, lacking the resolution to detect individual fingers from further away, only detects the pointing direction of the arm as a whole, the fingertip detector enables the detection of the pointing direction of an individual finger. This gives the user much more fine-grained control right in front of the displays. This is important, because the level of detail on the interface is much higher at this distance due to the distance dependent display. As shown in Figure 5.18, the fields of view of the two sensors overlap. In the situation where the pointing arm is registered by both systems, two results for the pointing position are available and the system needs to deal with this conflict of input. The pointing gesture system works best if the entire person is in the field of view as it is based on a skeleton tracking approach and therefore benefits from a more comprehensive view on the person to deliver robust input data. The fingertip detector works best right in front of the screen, as there are few sources of irritation for the sensor because of the flat and uniform background. As the user is approaching the displays, in the overlapping area of the two sensors, the pointing gesture gets continuously less accurate while the accuracy of the fingertip detector increases. Accordingly, the two results are fused by averaging the respective positions, weighting them depending on the user's position in the overlapping interval I as shown in Equation 5.1.

$$
\begin{aligned}
w &= \frac{I}{\|I\|} \\
x &= \frac{(w * x_p + (1 - w) * x_f)}{2} \\
y &= \frac{(w * y_p + (1 - w) * y_f)}{2}
\end{aligned}
\tag{5.1}
$$

The weight w is calculated by normalizing the interval. This way, the influence of the position originating from the pointing gesture (x_p, y_p) is maximized when the user just entered the interval and minimized right in front of the displays. The influence of the position produced by the fingertip detector (x_f, y_f) is scaled to have the reverse effect, resulting in the final position (x, y).
In addition to the gradual transition between the two pointing systems, the position of the user in the overlapping interval is used to control the opacity of the cursor. Since the fingertip detector is not only much more accurate than the pointing gesture recognition, but can also be used to directly touch the screen, a cursor is unnecessary. The pointing gesture benefits from a cursor as visual feedback of the current pointing position. By controlling the opacity in dependence of the users distance, the cursor is automatically available when further away, but hidden when right in front of the displays. The fusion of input data is implemented as a context handler in glueTK, which also generates signals that are connected to the cursor's `setOpacity()` slot. This way, the interface developer does not have to deal with multiple, changing modalities and their different requirements,

as the fusion is done at this early stage in the context handler and after that, both inputs are available as a single coherent input.
Because the distance of a user to the displays is used as input data, only one user can have control, which is a problem if multiple users are in front of the terminal or different users want to interact. Unlike the Smart Control Room application, there is no face identification system integrated here and while the Kinect can distinguish between different users, it can not identify them. So a role-based assignment of control is not an option. Therefore, the decision of control when multiple users are standing in front of the terminal has to be solved in a different manner. It is implemented as a hand waving gesture. This can be easily recognized from the skeleton tracking data and allows users to explicitly decide who has control as shown in Figure 5.17.
While the distance dependent interaction presented in this section is demonstrated by means of a special purpose application, the concepts are easily transferable to other fields as well. Of particular interest are interactive info-terminals in public spaces as the distance dependent display allows for user enticement and different stages of engagement.

6 Conclusion

This thesis presented the glueTK framework, which allows the creation of applications that can be controlled using novel input modalities and can spread across multiple displays and machines. A central property is the bidirectional communication between the input and output layers, which makes interface layout information available to input devices. The network-transparent signal and slot system provides a coherent communication and transfer mechanism within a glueApplication, spanning multiple glueFrames on multiple machines, and makes input data available at all displays. It allows to transfer interface elements across

screens and machines without differentiation between in-application and inter-application communication. No other framework provides all of these aspects required for interaction in multi-display environments in a single package. The equality of input and output enables novel opportunities to improve input devices as well as interactive applications.
Building upon the functionality provided by glueTK, such as access to the interface layout and separation between the input and output layer, a pointing enhancement technique for improving the target acquisition of a pointing gesture recognition system was presented. The technique, called dynamic Gaussian force fields, extends previous force field variants in several aspects. The Gaussian modeling of the field strength solves the problem of defining a single fixed strength, which is always a trade-off between help and irritation. By allocating for multi-directional forces of overlapping fields, a generic solution to the overlapping problem is presented, which allows force field placement even for complex user interfaces. As force fields can always be a source of irritation when not needed, the dynamic Gaussian force fields predict if an interface element is targeted from the user's pointing data and dynamically turn fields on and off. These improvements lead to faster, more accurate and smoother interaction, as has been shown in a user study following the ISO standard evaluation for multi-directional pointing tasks.
While the layout information required by target-aware pointing enhancements is available in glueTK, the same is not true for existing applications. The split of glueTK into input and output layers allows to utilize the input layer

and the abstraction from input device specific interfaces it provides, in combination with existing applications as well. Since this still requires access to and modification of the source code of an existing application, an approach to automatically detect the user interface layout for existing applications without such access has been presented. By acting as an intermediary between the existing user interface and the input data of an input device to be integrated, force fields can be used with existing applications without their knowledge. The approach involves capturing the screen and using template matching to localize targets. Unlike previous approaches, the need for prior training or configuration is avoided, as target templates are created on the fly by observing the user interact. By automatically generalizing target models using multiple acquired templates, the localization works fast enough to enhance the target acquisition with pointing gestures for any existing application, as a user study utilizing an application written using the desktop application framework Qt has revealed.
With the ability to improve the speed and accuracy of pointing input not only for glueTK applications but also for existing applications by utilizing the gained knowledge about the layout of the user interface, target acquisition is sufficiently fast and accurate for pointing gesture recognition. Once the target is acquired, however, it has to be selected as well. Since one of the advantages that make pointing gesture recognition natural and intuitive is the lack of any devices the user is required to wear or hold, target selection should be device-free as well. To give a comprehensive overview of device-free options to trigger a

selection of a target in mid-air, a novel taxonomy was presented that systematically categorized different arm-, hand- and finger gestures into respective classes to cover all possible kinds of target selection. In a Wizard-of-Oz study, all gestures from the taxonomy have been evaluated. The results show the "airtap" gesture as a clear favorite with significantly better results. In addition, the ranking created from all results allows for predictions about which gestures should be used for secondary interaction tasks and which should not be used at all.

GlueTK is tailored towards multi-display environments and pointing gestures are not bound to a single display, which makes them usable across multiple displays. Therefore, the design of user interfaces for different systems, taking into account the display and input device properties, is an important aspect of glueTK applications. When it comes to the effect the user interface has on the interaction performance, two distinctive parts of the interaction have been identified. Before an input device is even used, a target has to be visually acquired by the user. This perceptual performance has been studied for text but not for graphical interface elements. Therefore, the concept of a reference system has been introduced, which allows developers and interface designers to put design decisions they make into perspective by allowing the calculation of required element sizes for a target system to achieve the same perceptual performance as on the specific reference system they use. The accuracy of this relation has been shown to be below a second in a user study, involving multiple mobile devices as target systems with a desktop computer as a reference sys-

tem. The other distinctive part of the overall interaction performance is the input performance, the time it takes a user to select a target with a given input device once the target is visually acquired. An adaptation of the well known Fitts's law has been presented, which allows predictions with respect to a reference system for the input performance with similar accuracy as the predictions for the perceptual performance. An additional user study that analyzed the correlation between perceptual performance and input performance by provoking a decline of perceptual performance has shown that perceptual performance and input performance can be considered independent of each other, which allows predictions about the overall interaction performance by simple summation. The novel concept of the reference system allows developers to make design decisions by calculation instead of a tedious trial and error approach.

One of the guiding themes of this thesis is the coupling of input data and output of interactive systems. The glueTK framework lies the technical groundwork for the information exchange and the sections in Chapter 4 describe several improvements of pointing gesture interaction in a multi-display environment that utilize this tight integration. Besides pointing interaction, more and more modalities are explored and new input devices become available. Some have already been integrated into glueTK, but not to the extent of pointing gestures. The prediction of intended targets used to dynamically activate force fields is a first step to predicting the users intentions. Those are of course much more complex and vary with the application. Pre-

dicting more complex intentions however, could make many input devices even more robust and could also lead to intelligently adapting user interfaces. While the reference system concept introduced in this thesis allows to calculate the size of interface elements to be usable for a given interactive system, the layout of elements is left out. Especially with new display sizes and aspect ratios being constantly introduced, automatic layout adaptation will become a significant problem in the future as it will not be feasible to target every single device manually. The approaches so far require manual annotation, as context knowledge about the relationship of interface elements is required to create meaningful rearrangements. An interesting approach would be to derive this context information from observing the user interact, which would not only eliminate the need for providing context information manually but also create interface adaptations that are tailored towards the specific work flow of a user.

List of Figures

List of Tables

Own Publications

[BvdCS07] Keni Bernardin, Florian van de Camp, and Rainer Stiefelhagen. Automatic Person Detection and Tracking using Fuzzy Controlled Active Cameras. In *The Seventh IEEE International Workshop on Visual Surveillance (VS2007)*, pages 1–8, 2007.

[SvdCIS09] Alexander Schick, Florian van de Camp, Joris Ijsselmuiden, and Rainer Stiefelhagen. Extending Touch: Towards Interaction with Large-Scale Surfaces. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS 2009, pages 117–124, 2009. 126

[vdCBS09] Florian van de Camp, Keni Bernardin, and Rainer Stiefelhagen. Person Tracking in Camera Networks using Graph-Based Bayesian Inference. In *The Third ACM/IEEE International Conference on Distributed Smart Cameras, ICDSC 2009.*, pages 1–8, 2009.

[vdCS13a] Florian van de Camp and Rainer Stiefelhagen.

Applying Force Fields to Black-Box GUIs Using Computer Vision. In *Proceedings of the 1st IEEE Workshop on User-Centred Computer Vision*, UCCV 2013, pages 1–6. ACM, 2013.

[vdCS13b] Florian van de Camp and Rainer Stiefelhagen. Dynamic Gaussian Force Field Controlled Kalman Filtering for Pointing Interaction. In *Mensch & Computer 2013: Interaktive Vielfalt*, pages 261–270, München, 2013. Oldenbourg Verlag.

[vdCS13c] Florian van de Camp and Rainer Stiefelhagen. glueTK: A Framework for Multi-Modal, Multi-Display Human-Machine-Interaction. In *Proceedings of the 18th International Conference on Intelligence user interfaces*, IUI 2013, pages 329–338. ACM, 2013.

[vdCSS13a] Florian van de Camp, Alexander Schick, and Rainer Stiefelhagen. How to Click in Mid-Air. In *Proceedings of the 15th International Conference on Human-Computer Interaction*, HCII 2013, pages 78–86. Springer, 2013.

[vdCSS13b] Florian van de Camp, Patrick Schührer, and Rainer Stiefelhagen. How to Choose Element Sizes for Novel Interactive Systems. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS 2013, pages 385–388. ACM, 2013.

[vdCVS10] Florian van de Camp, Michael Voit, and Rainer Stiefelhagen. Efficient Person Identification using Active Cameras in a Smartroom. In *Proceedings of the 1st ACM international workshop on Multimodal pervasive video analysis*, MPVA 2010, pages 17–22, New York, NY, USA, 2010. ACM.

[VvdCI+13] Michael Voit, Florian van de Camp, Joris Ijsselmuiden, Alexander Schick, and Rainer Stiefelhagen. Visuelle Perzeption für die Multimodale Mensch-Maschine-Interaktion in und mit Aufmerksamen Räumen. *at - Automatisierungstechnik*, 61(11), 2013.

Bibliography

[AC05] Stephen Brewster Andy Cockburn. Multimodal Feedback for the Acquisition of Small Targets. *Ergonomics*, 48(9):1129–1150, 2005. 26, 23

[AHL06] David Ahlström, Martin Hitz, and Gerhard Leitner. An Evaluation of Sticky and Force Enhanced Targets in Multi Target Situations. In *4th Nordic conference on Human-Computer-Interaction (NordiCHI)*, pages 14–18, 2006. 30, 83, 96, 120, 27, 73, 86, 109

[And] Google Android Developer Guide. *http://developer.android.com/guide* (accessed November 2013). 153, 141

[App] Apple iOS Programming Guide. *https://developer.apple.com/library/ios* (accessed November 2013). 152, 140

[AZ97] Johnny Accot and Shumin Zhai. Beyond Fitts' Law: Models for Trajectory-based

HCI Tasks. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, CHI '97, pages 295–302, New York, NY, USA, 1997. ACM. 39, 36

[AZ01] Johnny Accot and Shumin Zhai. Scale Effects in Steering Law Tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '01, pages 1–8, New York, NY, USA, 2001. ACM. 39, 36

[Bad11] Thomas Bader. *Multimodale Interaktion in Multi-Display-Umgebungen.* PhD thesis, Karlsruhe Institute of Technology (KIT), 2011. 19, 18

[BB99] B.B. Bederson and A. Boltman. Does Animation Help Users Build Mental Maps of Spatial Information? In *Information Visualization, 1999. (Info Vis '99) Proceedings. 1999 IEEE Symposium on*, pages 28–35, 1999. 35, 32

[BDB+97] Rodney A. Brooks, Darren Dang, Jeremy De Bonet, Joshua Kramer, Tomas Lozano-perez, John Mellor, and Polly Pook. The Intelligent Room project. In *Proceedings of the 2nd International Conference on Cognitive Technology*, CT '97, pages 271–, Washington, DC, USA, 1997. 17, 18

[BDHM10] Andrew Bragdon, Rob Deline, Ken Hinckley, and Meredith Ringel Morris. Code Space : Touch + Air Gesture Hybrid Interactions for Supporting Developer Meetings. In *ACM International Conference on Interactive Tabletops and Surfaces, ITS 2010*, Kobe, Japan, 2010. 34, 31

[BMG10] Till Ballendat, Nicolai Marquardt, and Saul Greenberg. Proxemic Interaction: Designing for a Proximity and Orientation-aware Environment. In *ACM International Conference on Interactive Tabletops and Surfaces*, ITS '10, pages 121–130, New York, NY, USA, 2010. ACM. 37, 34

[Bmi09] Feuerwehr-Dienstvorschrift 100. Technical report, Bundesministerium des Inneren, 2009. 174

[Bol80] Richard A Bolt. Put-that-there: Voice and Gesture at the Graphics Interface. *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, 14(3):262–270, 1980. 16, 125, 14

[boo] Boost. *http://www.boost.org* (accessed November 2013). 56, 55

[BR03] Harry Brignull and Yvonne Rogers. Enticing People to Interact with Large Public Displays in Public Spaces. In

Matthias Rauterberg, Marino Menozzi, and Janet Wesson, editors, *In Proceedings of the IFIP International Conference on Human-Computer Interaction (INTERACT 2003)*, pages 17–24, 2003. 36, 33

[BRB09] T. Bader, R. Räpple, and J. Beyerer. Fast Invariant Contour-Based Classification of Hand Symbols for HCI. In *Proceedings of Computer Analysis of Images and Patterns*, pages 689–696, 2009. 125, 126

[Bro05] PJ Brock. An Investigation of Target Acquisition with Visually Expanding Targets in Constant Motor-space. Master's thesis, University of Canterbury, New Zealand, 2005. 29, 82, 26, 72

[BS90] R. Baecker and I. Small. *Animation at the Interface*. Addison-Wesley, New York, 1990. 35, 32

[BT93] Izak Benbasat and Peter Todd. An Experimental Investigation of Interface Design Alternatives: Icon vs. Text and Direct Manipulation vs. Menus. *International Journal of Man-Machine Studies*, 38(3):369–402, March 1993. 155, 143

[BYCH05] Darius Burschka, Guangqi Ye, Jason J. Corso, and Gregory D. Hager. A Practical

Approach for Integrating Vision-Based Methods Into Interactive 2D/3D Applications. Technical report, The Johns Hopkins University, 2005. 32, 29

[Byr93] Michael D. Byrne. Using Icons to Find Documents: Simplicity is Critical. In *Proceedings of the ACM SIGCHI conference on Human factors in computing systems*, INTERCHI '93, pages 446–453, Amsterdam, The Netherlands, 1993. 40, 37

[CB06] Andy Cockburn and Philip Brock. Human On-line Response to Visual and Motor Target Expansion. In *Graphics Interface*, pages 81–87. Canadian Human-Computer Communications Society, 2006. 29, 26

[CEB87] S. K. Card, W. K. English, and B. J. Burr. Human-computer Interaction. chapter Evaluation of Mouse, Rate-controlled Isometric Joystick, Step Keys, and Text Keys, for Text Selection on a CRT, pages 386–392. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987. 38, 35

[CJM+97] Philip R. Cohen, Michael Johnston, David McGee, Sharon Oviatt, Jay Pittman, Ira Smith, Liang Chen, and Josh Clow. QuickSet: Multimodal Interaction for Simulation Set-up and Control. In *Proceedings of the fifth conference on*

Applied natural language processing, ANLC '97, pages 20–24, Stroudsburg, PA, USA, 1997. Association for Computational Linguistics. 18, 175, 163

[CRM91] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. The Information Visualizer, an Information Workspace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pages 181–186, New York, NY, USA, 1991. ACM. 140, 151

[CU95] Bay-Wei Chang and David Ungar. Animation: From Cartoons to the User Interface. Technical report, Mountain View, CA, USA, 1995. 35, 32

[CYM10] Tsung-Hsiang Chang, Tom Yeh, and Robert C. Miller. GUI Testing using Computer Vision. In *International conference on Human factors in computing systems*, CHI '10, pages 1535–1544, New York, NY, USA, 2010. 32, 29

[DCE+11] A. Dworak, P. Charrue, F. Ehm, W. Sliwinski, and M. Sobczak. Middleware Trends and Market Leaders 2011. In *13th International Conference on Accelerator and Large Experimental Physics Control Systems, 2011*, pages 1334–1337, 2011. 20, 18

[DF10] Morgan Dixon and James Fogarty. Prefab: Implementing Advanced Behaviors using Pixel-based Reverse Engineering of Interface Structure. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 1525–1534, New York, NY, USA, 2010. ACM. 32, 106, 29, 96

[DFW12] Morgan Dixon, James Fogarty, and Jacob Wobbrock. A General-purpose Target-aware Pointing Enhancement Using Pixel-level Analysis of Graphical Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 3167–3176, New York, NY, USA, 2012. ACM. 32, 29

[DGBG05] Iain Darroch, Joy Goodman, Stephen Brewster, and Phil Gray. The Effect of Age and Font Size on Reading Text on Handheld Computers. In *INTERACT 2005*, volume 3585 of *Lecture Notes in Computer Science*, pages 253–266. Springer, 2005. 40, 37

[DLF11] Morgan Dixon, Daniel Leventhal, and James Fogarty. Content and Hierarchy in Pixel-based Methods for Reverse Engineering Interface Structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI

'11, pages 969–978, New York, NY, USA, 2011. ACM. 32, 29

[DLO09] Bruno Dumas, Denis Lalanne, and Sharon Oviatt. Multimodal Interfaces : A Survey of Principles , Models and Frameworks. In *Human Machine Interaction*, volume 5440 of *Lecture Notes in Computer Science*, pages 3–26. Springer, 2009. 17, 15

[Dru75] Colin Drury. Application of Fitts' Law to Foot-Pedal Design. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 1975. 38, 35

[EK08] Florian Echtler and Gudrun Klinker. A Multitouch Software Architecture. In *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges*, NordiCHI '08, pages 463–466, New York, NY, USA, 2008. ACM. 19, 17

[FGA+07] Vitor Fernandes, Tiago Guerreiro, Bruno Araújo, Joaquim Jorge, and João Pereira. Extensible Middleware Framework for Multimodal Interfaces in Distributed Environments. In *Proceedings of the 9th international conference on Multimodal interfaces*, ICMI '07, pages 216–219, New York, NY, USA, 2007. ACM. 20, 18

[Fit54] Paul M. Fitts. The Information Capacity of the Human Motor System in Controlling

the Amplitude of Movement. *Journal of Experimental Psychology*, 47(6):381–391, 1954. 24, 38, 161, 22, 35

[FJS10] Leah Findlater, Alex Jansen, and Kristen Shinohara. Enhanced Area Cursors: Reducing Fine Pointing Demands for People with Motor Impairments. *UIST '10, October 3-6, 2010, NY, New York.*, 2010. 26, 82, 23, 72

[FKM03] Frans Flippo, Allen Krebs, and Ivan Marsic. A Framework for Rapid Development of Multimodal Interfaces. In *Proceedings of the 5th International Conference on Multimodal Interfaces*, ICMI '03, pages 109–116, New York, NY, USA, 2003. ACM. 16, 52, 15, 50

[Gar] Forecast: Devices by Operating System and User Type, Worldwide, 2010-2017, 1Q13 Update. *https://www.gartner.com/doc/2396815* (accessed November 2013). 2

[GB05] Tovi Grossman and Ravin Balakrishnan. The Bubble Cursor: Enhancing Target Acquisition by Dynamic Resizing of the Cursor's Activation Area. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 281–290, 2005. 26, 82, 126, 23, 72

[GBBL04] Yves Guiard, Renaud Blanch, and Michel Beaudouin-Lafon. Object Pointing: A Complement to Bitmap Pointing in GUIs. In *Proceedings of the 2004 Graphics Interface Conference*, GI '04, pages 9–16, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society. 28, 82, 25, 72

[GC92] David Gelernter and Nicholas Carriero. Coordination Languages and Their Significance. *Communications of the ACM*, 35(2):97–107, February 1992. 21, 19

[GER+07] Jürgen Geisler, Ralf Eck, Nils Rehfeld, Elisabeth Peinsipp-Byma, Christian Schütz, and Sven Geggus. Fovea-Tablett : A New Paradigm for the Interaction with Large Screens. In *Human Interface and the Management of Information.*, volume 4557 of *Lecture Notes in Computer Science*, pages 278–287. Springer, 2007. 177, 165

[GLH10] Adam Gokcezade, Jakob Leitner, and Michael Haller. LightTracker: An Open-Source Multitouch Toolkit. *Computers in Entertainment (CIE)*, 8(3):19:1–19:16, December 2010. 19, 17

[Gon96] Cleotilde Gonzalez. Does Animation in User Interfaces Improve Decision Making ?

In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '96, pages 27–34, New York, NY, USA, 1996. ACM. 140, 151

[GW07] Tovi Grossman and Daniel Wigdor. Going Deeper: a Taxonomy of 3D on the Tabletop. In *Second Annual IEEE International Workshop on Horizontal Interactive Human-Computer Systems, TABLETOP '07*, pages 137–144, 2007. 34, 31

[GWW07] Krzysztof Z. Gajos, Jacob O. Wobbrock, and Daniel S. Weld. Automatically Generating User Interfaces Adapted to Users' Motor and Vision Capabilities. In *ACM Symposium on User Interface Software and Technology (UIST)*, UIST '07, pages 231–240, New York, NY, USA, 2007. ACM. 40, 36

[Hal90] Edward Hall. *The Hidden Dimension*. Peter Smith Publisher Inc., Gloucester, MA, USA, 1990. 37, 34

[Har99] Eric Harlow. *Developing Linux Applications: With Gtk+ and Gdk*. New Riders Publishing, Thousand Oaks, CA, USA, 1999. 22, 15

[HDS11] Lode Hoste, Bruno Dumas, and Beat Signer. Mudra: A Unified Multimodal

Interaction Framework. In *Proceedings of ICMI, 13th International Conference on Multimodal Interaction, 2011*, pages 97–104, 2011. 17, 52, 15, 50

[Hen11] Vitali Henne. Hand Gesture Recognition with a Depth-Sensing Camera. Bachelorthesis, Karlsruhe Institute of Technology (KIT), 2011. 195, 182

[HHV+09] Thomas E. Hansen, Juan Pablo Hourcade, Mathieu Virbel, Sharath Patali, and Tiago Serra. PyMT: A Post-WIMP Multi-touch User Interface Toolkit. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '09, pages 17–24, New York, NY, USA, 2009. ACM. 23, 20

[HS93] Scott E. Hudson and John T. Stasko. Animation Support in a User Interface Toolkit: Flexible, Robust, and Reusable Abstractions. In *Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology*, UIST '93, pages 57–67, New York, NY, USA, 1993. ACM. 140, 151

[IBR+03] Shahram Izadi, Harry Brignull, Tom Rodden, Yvonne Rogers, and Mia Underwood. Dynamo: A Public Interactive Surface Supporting the Cooperative Sharing

and Exchange of Media. In *Proceedings of the 16th annual ACM symposium on User interface software and technology*, UIST '03, pages 159–168, New York, NY, USA, 2003. ACM. 34, 31

[ISO00] ISO. Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) – Part 9: Requirements for Non-keyboard Input Devices. ISO 9241-9 2000, International Organization for Standardization, Geneva, Switzerland, 2000. 93, 117, 83, 107

[ISO08] ISO. Ergonomics of Human-system Interaction – Part 304: User Performance Test Methods for Electronic Visual Displays. ISO 9241-304 2008, International Organization for Standardization, Geneva, Switzerland, 2008. 40, 157, 37, 145

[ISO11] ISO. Ergonomics of Human-system Interaction – Part 303: Requirements for Electronic Visual Displays. ISO 9241-303 2011, International Organization for Standardization, Geneva, Switzerland, 2011. 40, 154, 37, 142

[JF02] Brad Johanson and Armando Fox. The Event Heap: A Coordination Infrastructure for Interactive Workspaces. In *Proceedings of the Fourth IEEE Workshop on Mobile*

Computing Systems and Applications, WMCSA '02, pages 83–, Washington, DC, USA, 2002. IEEE Computer Society. 20, 19

[JFW02] B. Johanson, A. Fox, and T. Winograd. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. *IEEE Pervasive Computing*, 1(2):67–74, April 2002. 20, 19

[JFW11] Alex Jansen, Leah Findlater, and Jacob O. Wobbrock. From the Lab to the World: Lessons from Extending a Pointing Technique for Real-world Use. In *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems*, CHI EA '11, pages 1867–1872, New York, NY, USA, 2011. ACM. 30, 27

[JLK08] Wendy Ju, Brian A. Lee, and Scott R. Klemmer. Range: Exploring Implicit Interaction Through Electronic Whiteboard Design. In *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, CSCW '08, pages 17–26, New York, NY, USA, 2008. ACM. 37, 34

[JM85] Richard Jagacinski and Donald Monk. Fitts' Law in Two Dimensions with Hand and Head Movements. *Journal of Motor Behavior*, 1985. 38, 35

[JPSK04] Bonnie E. John, Konstantine Prevas, Dario D. Salvucci, and Ken Koedinger. Predictive Human Performance Modeling Made Easy. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 455–462, New York, NY, USA, 2004. ACM. 39, 36

[JRV+89] Jeff Johnson, Teresa L. Roberts, William Verplank, David C. Smith, Charles H. Irby, Marian Beard, and Kevin Mackey. The Xerox Star: A Retrospective. *Computer*, 22(9):11–26, 28–29, September 1989. 5

[Kal60] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME Journal of Basic Engineering*, pages 35–45, 1960. 88, 78

[KB95] Paul Kabbash and William A S Buxton. The "Prince" Technique. In *Proceedings of the SIGCHI conference on Human factors in computing systems CHI 95*, pages 273–279, 1995. 25, 82, 22, 72

[KB07] Martin Kaltenbrunner and Ross Bencina. reacTIVision: A Computer-vision Framework for Table-based Tangible Interaction. In *Proceedings of the 1st international conference on Tangible and*

embedded interaction, TEI '07, pages 69–74, New York, NY, USA, 2007. ACM. 18, 17

[KBBC05] M. Kaltenbrunner, T. Bovermann, R. Bencina, and E. Costanza. TUIO: A Protocol for Table-Top Tangible User Interfaces. In *Proceedings of the The 6th International Workshop on Gesture in Human-Computer Interaction and Simulation*, Vannes, France, 2005. 18, 17

[KE88] B. Kantowitz and G. Elvers. Fitts' with an Isometric Controller: Effects of Order of Control and Control-display Gain. *Journal of Motor Behavior*, 1988. 38, 35

[Kel83] J.F. Kelley. An Empirical Methodology for Writing User-friendly Natural Language Computer Applications. In *ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 193–196, 1983. 131, 120

[Ken04] A. Kendon. *Gesture: Visible Action as Utterance*. Cambridge University Press, 2004. 125, 127, 115

[KHM+00] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale, and S. Shafer. Multi-camera Multi-person Tracking for EasyLiving. *Proceedings of the third International Workshop on Visual Surveillance*, pages 3–10, 2000. 21, 19

[Kin] Microsoft Kinect. *http://www.microsoft.com/en-us/kinectforwindows* (accessed November 2013). 7, 53, 193, 6, 51, 181

[KO08] Vassilis Kostakos and Eamonn O'Neill. Size Matters: Performance Declines if Your Pixels are too Big or too Small. *Computing Research Repository*, abs/0804.3103, 2008. 39, 36

[Lea] Leap Motion. *https://www.leapmotion.com* (accessed November 2013). 53, 72, 51

[LHS08] Bettina Laugwitz, Theo Held, and Martin Schrepp. Construction and evaluation of a user experience questionnaire. In *Proceedings of the 4th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society on HCI and Usability for Education and Work*, USAB '08, pages 63–76, Berlin, Heidelberg, 2008. Springer-Verlag. 73

[LNP+09] Denis Lalanne, Laurence Nigay, Philippe Palanque, Peter Robinson, and Jean Vanderdonckt. Fusion Engines for Multimodal Input : A Survey. *Interfaces*, pages 153–160, 2009. 17, 52, 15, 50

[Mac92] I. Scott MacKenzie. Fitts' Law as a Research and Design Tool in

Human-computer Interaction. *Human-Computer Interaction*, 7(1):91–139, 1992. 38, 35

[MB93] I. Scott MacKenzie and William Buxton. A Tool for the Rapid Evaluation of Input Devices using Fitts' Law Models. *ACM SIGCHI Bulletin*, 25(3):58–63, 1993. 38, 35

[MBB+12] N. Marquardt, T. Ballendat, S. Boring, S. Greenberg, and K. Hinckley. Gradual Engagement between Digital Devices as a Function of Proximity: From Awareness to Progressive Reveal to Information Transfer. Technical Report 2012-1025-08, Department of Computer Science, University of Calgary, Calgary, Alberta, Canada, April 2012. 37, 34

[MBN03] Atif Memon, Ishan Banerjee, and Adithya Nagarajan. GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing. In *Working Conference on Reverse Engineering*, WCRE '03, pages 260–, Washington, DC, USA, 2003. IEEE Computer Society. 32, 29

[MCdB99] S.J. McDougall, M.B. Curry, and O. de Bruijn. Measuring Symbol and Icon Characteristics: Norms for Concreteness, Complexity, Meaningfulness, Familiarity, and Semantic Distance for 239 Symbols.

Behavior Research Methods, Instruments & Computers, 31(3):487–519, 1999. 40, 37

[McN92] D. McNeill. *Hand and Mind: What Gestures Reveal about Thought.* University of Chicago Press, 1992. 126

[MDMBG11] Nicolai Marquardt, Robert Diaz-Marino, Sebastian Boring, and Saul Greenberg. The Proximity Toolkit: Prototyping Proxemic Interactions in Ubiquitous Computing Ecologies. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 315–326, New York, NY, USA, 2011. ACM. 37, 34

[MGVVR09] Jérémie Melchior, Donatien Grolaux, Jean Vanderdonckt, and Peter Van Roy. A Toolkit for Peer-to-peer Distributed user Interfaces: Concepts, Implementation, and Applications. In *Proceedings of the ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '09, pages 69–78, New York, NY, USA, 2009. 21, 20

[Mil68] Robert B. Miller. Response Time in Man-computer Conversational Transactions. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), pages 267–277, New York, NY, USA, 1968. ACM. 140, 151

[MP02] Guido Menkhaus and Wolfgang Pree. A Hybrid Approach to Adaptive User Interface Generation. In *Proceedings of the 24th International Conference on Information Technology Interfaces, ITI, 2002*, volume 1, pages 185–190, 2002. 188, 176

[MU02] Takaki Mori and Kuniaki Uehara. Extraction of Primitive Motion and Discovery of Association Rules from Motion Data. In *Proceedings 10th IEEE International Workshop on Robot and Human Interactive Communication*, pages 200–206. IEEE, 2002. 90, 80

[NKG13] Matthias Nielsen, Mikkel Baun Kjærgaard, and Kaj Grønbæk. Exploring Interaction Techniques and Task Types for Direct-Touch as Input Modality. In *Proceedings of the IEEE Visual Analytics Science and Technology, IEEE Information Visualization, and IEEE Scientific Visualization (IEEE VIS), 2013*, October 2013. 4

[NPRI09] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of JSON and XML Data Interchange Formats: A Case Study. In *CAINE'09*, pages 157–162, 2009. 59, 58

[NS07] Kai Nickel and Rainer Stiefelhagen. Visual Recognition of Pointing Gestures for Human-robot Interaction. *Image and Vision Computing*, 25(12):1875–1884, December 2007. 3

[NWP+11] M. Nancel, J. Wagner, E. Pietriga, O. Chapuis, and W. Mackay. Mid-air Pan-and-zoom on Wall-sized Displays. In *Proceedings of SIGCHI Conference on Human Factors in Computing Systems*, pages 177–186, 2011. 33, 125, 30

[ope] OpenNI. *http://www.openni.org* (accessed November 2013). 195, 182

[PB02] Peter R. Pietzuch and Jean Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *Proceedings of the 22Nd International Conference on Distributed Computing Systems*, ICDCSW '02, pages 611–618, Washington, DC, USA, 2002. IEEE Computer Society. 20, 18

[PBERG07] Elisabeth Peinsipp-Byma, Ralf Eck, Nils Rehfeld, and Jürgen Geisler. Situation Analysis at a Digital Situation Table with Fovea-Tablett. volume 6495, page 64950E. SPIE, 2007. 139

[PBGB09] Elisabeth Peinsipp-Byma, Jürgen Geisler, and Thomas Bader. Digital Map and

Situation Surface: a Team-oriented Multidisplay Workspace for Network Enabled Situation Analysis. In John T. Thomas and Daniel D. Desjardins, editors, *Display Technologies and Applications for Defense, Security, and Avionics III*, volume 7327, page 732703. SPIE, 2009. 177, 165

[Pro] Processing libraries. *http://www.processing.org* (accessed November 2013). 22, 16

[PRS+03] Thorsten Prante, Carsten Röcker, Norbert Streitz, Richard Stenzel, Carsten Magerkurth, Daniel van Alphen, and Daniela Plewe. Hello.Wall - Beyond Ambient Displays. In *Proceedings of Ubicomp'03*, pages 277–278, Seattle, WA, USA, 2003. Springer. 37, 34

[PSH97] V.I. Pavlovic, R. Sharma, and T.S. Huang. Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review. In *Pattern Analysis and Machine Intelligence*, pages 677–695, 1997. 125

[PZ97] Michael Philippsen and Matthias Zenger. JavaParty - Transparent Remote Objects in Java. *Concurrency: Practice and Experience*, 9(11):1225–1242, 1997. 20, 18

[Qt2] Qt Project. *http://qt-project.org/doc/qt-*

5.0/qtcore/signalsandslots.html (accessed November 2013). 56, 68, 117

[Que94] F.K.H. Quek. Toward a Vision-Based Hand Gesture Interface. In *Proceedings of Virtual Reality Software and Technology*, pages 17–29, 1994. 34, 126, 129, 30, 115

[Que95] F.K.H. Quek. Eyes in the Interface. In *Image and Vison Computing*, pages 511–525, 1995. 34, 126, 129, 30, 115

[RG98] Dirk Riehle and Thomas Gross. Role Model Based Framework Design and Integration. In *Proceedings of the 13th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '98, pages 117–133, New York, NY, USA, 1998. ACM. 16

[Rot12] Frank Roth. Interactive Data Transfer in Multi Display Environments. Bachelorthesis, Hochschule Karlsruhe University of Applied Sciences, 2012. 141, 129

[SBL93] John Stasko, Albert Badre, and Clayton Lewis. Do Algorithm Animations Assist Learning?: An Empirical Study and Analysis. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93,

pages 61–66, New York, NY, USA, 1993. ACM. 35, 32

[Sea93] Andrew Sears. Layout Appropriateness: A metric for Evaluating User Interface Widget Layout. *IEEE Transactions on Software Engineering*, 19:707–719, 1993. 39, 188, 36, 176

[Sky] Skype. *http://www.skype.com* (accessed November 2013). 107, 97

[SNL+08] Marcos Serrano, Laurence Nigay, Jean-yves L Lawson, Andrew Ramsay, Roderick Murray-smith, Sebastian Denef, and Sankt Augustin. The OpenInterface Framework : A Tool for Multimodal Interaction. In *Design*, pages 3501–3506, 2008. 17, 15

[SSG+10] Marcus Specht, Andrea Söter, Jens Gerken, Hans-christian Jetter, Harald Reiterer, Arbeitsgruppe Mensch-computer Interaktion, Universität Konstanz, and Volkswagen Ag. Dynamic Force Fields zur Präzisionserhöhung von zeigegeräten. *Mensch & Computer 2010*, 2010. 31, 28

[SVFR04] Chia Shen, Frédéric D. Vernier, Clifton Forlines, and Meredith Ringel. DiamondSpin: An Extensible Toolkit for Around-the-table Interaction. In

Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '04, pages 167–174, New York, NY, USA, 2004. ACM. 19, 17

[TC89] C. H. Teh and R. T. Chin. On the Detection of Dominant Points on Digital Curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(8):859–872, August 1989. 109, 99

[TH13] Matthew Turk and Gang Hua. *Vision-Based Interaction*. Synthesis Lectures on Computer Vision. Morgan & Claypool Publishers, 2013. 4

[TL02] Thuan L. Thai and Hoang Lam. *.NET Framework Essentials (2nd Edition)*. O' Reilly & Associates, Inc., 2002. 22, 15

[VB04] Daniel Vogel and Ravin Balakrishnan. Interactive Public Ambient Displays: Transitioning from Implicit to Explicit, Public to Personal, Interaction with Multiple Users. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, UIST '04, pages 137–146, New York, NY, USA, 2004. ACM. 38, 35

[VB05] D. Vogel and R. Balakrishnan. Distant Freehand Pointing and Clicking on Very

Large, High Resolution Displays. In *ACM symposium on User Interface Software and Technology*, pages 33–42, 2005. 33, 125, 126, 30, 114

[vid] Videmo Face SDK. *http://videmo.de/products* (accessed November 2013). 176, 164

[VNS07] Michael Voit, Kai Nickel, and Rainer Stiefelhagen. Neural Network-Based Head Pose Estimation and Multi-view Fusion. In *Multimodal Technologies for Perception of Humans*, volume 4122, pages 291–298. Springer, 2007. 177, 165

[Vvv] vvvv - a multipurpose toolkit. *http://www.vvvv.org* (accessed November 2013). 23, 16

[War00] Patrick Ward. *QT Programming for Linux and Windows*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000. 22, 15

[WFL+09] Jacob O. Wobbrock, James Fogarty, Shih-Yen (Sean) Liu, Shunichi Kimuro, and Susumu Harada. The Angle Mouse: Target-agnostic Dynamic Gain Adjustment based on Angular Deviation. In *Proceedings of the 27th international conference on Human factors in computing systems*, CHI '09, pages 1401–1410, New York, NY, USA, 2009. ACM. 31, 28

[WG00] Torben Weis and Kurt Geihs. Components on the Desktop. In *Proceedings of the Technology of Object-Oriented Languages and Systems*, TOOLS '00, Washington, DC, USA, 2000. 55, 53

[Wie99] Susan Wiedenbeck. The Use of Icons and Labels in an End User Application Program: An Empirical Study of Learning and Retention. *Behaviour & Information Technology*, 18(2):68–82, 1999. 155, 143

[Win70] B. J. Winer. *Statistical Principles in Experimental design*. McGraw-Hill, 1970. 160

[WMJ+12] Andy Wu, Sam Mendenhall, Jayraj Jog, Loring Scotty Hoag, and Ali Mazalek. A Nested API Structure to Simplify Cross-device Communication. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*, TEI '12, pages 225–232, New York, NY, USA, 2012. ACM. 22, 20

[WMW09] J.O. Woobrock, M.R. Morris, and A.D. Wilson. User-Defined Gestures for Surface Computing. In *Proceedings of Human Factors in Computing Systems*, pages 1083–1092, 2009. 33, 30

[WRW96] Ann Wollrath, Roger Riggs, and Jim Waldo. A Distributed Object Model for the Java System. In *Proceedings of the 2nd Conference on USENIX Conference on Object-Oriented Technologies (COOTS) - Volume 2*, COOTS'96, pages 17–17, Berkeley, CA, USA, 1996. USENIX Association. 20, 18

[WWBH97] Aileen Worden, Nef Walker, Krishna Bharat, and Scott Hudson. Making Computers Easier for Older Adults to use: Area Cursors and Sticky Icons. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, volume 97p of *ACM Conference on Human Factors in Computing Systems*, pages 266–271. ACM New York, NY, USA, ACM, 1997. 25, 82, 22, 72

[WZX11] Xibo Wang, Qiao Zhou, and Yizhong Xin. The Construction and Application of Multitouch Interactive Platform Based on Touchlib. In *Proceedings of the 2011 4th International Conference on Intelligent Networks and Intelligent Systems*, ICINIS '11, pages 153–156, Washington, DC, USA, 2011. IEEE Computer Society. 19, 17

[YCM09] Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. Sikuli: Using GUI Screenshots for Search and Automation. In

ACM Symposium on User Interface Software and Technology (UIST), UIST '09, pages 183–192, New York, NY, USA, 2009. ACM. 32, 29

[Yin06] Jibin Yin. The Beam Cursor: A Pen-based Technique for Enhancing Target Acquisition. In *Proceedings of the ACM Conference on Human-Computer-Interaction, HCI 2006*, pages 119–134, 2006. 28, 25

[ZRZ08] Xinyong Zhang, Xiangshi Ren, and Hongbin Zha. Improving Eye Cursor's Stability for Eye Pointing Tasks. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 525–534. ACM, 2008. 31, 28

Appendices

A

Used Questionaires

A.1 glueTK - Developer Study

Developer Questionnaire

Task 0 : *Get to know glueTK*

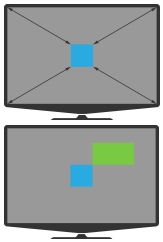
Take a look at the documentation ☐

Find the tutorials, compile & run them and take a look at the code ☐

Do an initial commit (git commit -am "task0") ☐

Task 1: *Display*

The frame id is set to "taskApp" for all task templates, which is already defined in the task.xml file that is loaded by the frame - there is no need to modify this configuration! However, feel free to take a look !

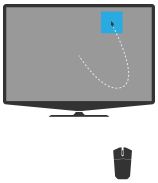


Use the application template "task1" to create a fullscreen application that displays the image "blue.png" in the exact center. ☐

Place a second image "green.png" in such a way that its bottom left corner touches the top right corner of "blue.png" ☐

Commit your work (git commit -am "task1") ☐

Task 2: *Interaction*



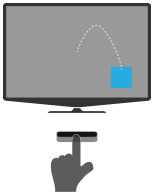
Here you will use the `connect` command for the first time. See tutorials 4+5 to understand how the `connect` command works.

Slots used in the `connect` command are just strings but the method name and variable names passed are parsed and used !

Use the application template "task2" to create a fullscreen application that displays the image "blue.png" and make the image follow the mouse position ☐

Commit your work (git commit -am "task2") ☐

Task 3: *Modalities*



The task3 template comes with an EventHandler for the Leapmotion which provides the pointing data as a signal called "leapPoint" containing the frame (called "`frame`") it intersects as well as the x (called "x") and y (called "y") coordinates of the intersection

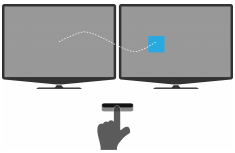
Use the application template "task3" to create a fullscreen application that displays the image "blue.png" and make the image follow the pointing direction of the index finger, using the leap motion. ☐

Commit your work (git commit -am "task3") ☐

Task 4: *Transfer*

The target frames name is “receiverApp”

If you have not yet, take a look at the different variants of setting the position of a block in the documentation.



Use the application template “task4” to modify the code of task3 to allow for the image to be moved across screens. □

Commit your work (git commit -am “task4”) □

Feedback

Demographics

Age	
Gender (m/w)	

Coding

How would you rate your C++ knowledge (0 = No knowledge at all, 5 = Expert)	
How much experience do you have with GUI development ? (0 = No experience at all, 5 = I do GUI development on a regular basis)	
How many hours do spend coding / week ?	
How many of those are coding C++ ?	
Did you have any prior experience with glueTK ?	

Frameworks
How well do you know the following software ?
(1=never heard of it, 2=heard of it, 3=used it, 4=use it regularly, 5=expert)

Software	*Level of experience*
Qt	
Gtk	
PyMT / Kivy	
Cocos2d	
TuIO	
Microsoft Surface SDK	
LeapMotion SDK	
Squid Middleware	

Sources of Information about glueTK
How valuable were the following sources of information to you ?
(1=did not use it at all, 2=hardly used it, 3=used it some, 4=used it regularly, 5=main source of information)

Overview Paper	
Doxygen Documentation	
Source code	
Tutorials	

Time
Order the tasks according to the time you needed to complete them
(1=fastest, 5=slowest)

Task 0	
Task 1	
Task 2	
Task 3	
Task 4	

How would you say you spend your time ?
(Give percent values adding up to 100%)

Reading (Documentation, Code, Tutorials, Overview)	
Coding	
Testing/Debugging	
Other	

If Other > 0%, please elaborate:

Complexity
How would you judge the difficulty of each task ?
(1=unable to complete, 2=help required to complete, 3=it took effort, 4=no problem, 5=very easy)

Task 0	
Task 1	
Task 2	
Task 3	
Task 4	

Comments
Do you have any suggestions, ideas for improvement ?

A.2 User study - Force Fields

User Study : Klicken mit Gesten

Alter (in Jahren) :

Geschlecht :

Größe (in cm) :

Rechtshänder ? (ja/nein) :

Erfahrung mit Gesteninteraktion ? (ja/nein) :

Welche Technik hat Ihnen am besten gefallen ? (1,2 oder 3) :

Welche Technik empfanden Sie als die schnellste ? (1,2 oder 3) :

Welche Technik(en) empfanden Sie als natürlich ? (1,2,3) :

Anmerkungen:

A.3 User study - Black-Box GUIs

User Study : Gestensteuerung von Desktopanwendungen

Alter (in Jahren) :

Geschlecht :

Größe (in cm) :

Rechtshänder ? (ja/nein) :

Erfahrung mit Gesteninteraktion ? (ja/nein) :

Welche Technik hat Ihnen am besten gefallen ? (1,2 oder 3) :

Welche Technik empfanden Sie als die schnellste ? (1,2 oder 3) :

Könnten Sie sich vorstellen mit einer der Techniken eine Desktopanwednung (z.B. Videoplayer) zu bedienen ? :

Wenn ja, mit welcher/welchen ? :

Anmerkungen:

A.4 User study - How to click in mid air

Allgemeine Fragen

Geschlecht (m/w):

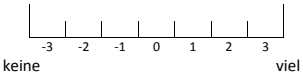
Wie groß sind Sie (in cm)?

Wie alt sind Sie?

Üben Sie einen technischen Beruf aus oder haben Sie eine technische Ausbildung absolviert?

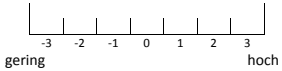
Sind Sie Rechtshänder?

Wie viel Erfahrung haben Sie mit Gesteninteraktion (z.B. Kinect, Wii, Touchscreens)?



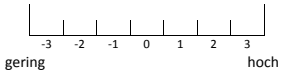
Geistige Anforderungen Technik: Drücken

Wie viel geistige Anstrengung war für die korrekte Ausführung der Geste erforderlich (z.B. Denken, Entscheiden, Rechnen, Erinnern, Hinsehen, Suchen …)? War die Geste leicht oder anspruchsvoll, einfach oder komplex, erfordert sie hohe Genauigkeit oder ist sie fehlertolerant?



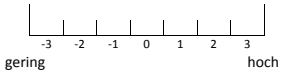
Körperliche Anforderungen

Wie viel körperliche Aktivität war erforderlich? War die Geste körperlich leicht oder schwer, einfach oder anstrengend, erholsam oder mühselig?



Zeitliche Anforderungen

Wie empfanden Sie den Zeitaufwand für die Ausführung der Geste?

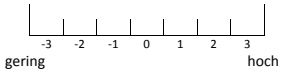


Ausführung der Aufgaben

Wie erfolgreich haben Sie ihrer Meinung nach die Aufgabe erreicht? Wie zufrieden waren Sie mit Ihrer Leistung bei der Verfolgung dieser Ziele?

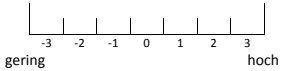
Anstrengung

Wie hart mussten Sie arbeiten, um Ihren Grad an Aufgabenerfüllung zu erreichen?



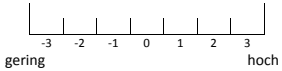
Frustration

Wie unsicher, entmutigt, irritiert, gestresst und verärgert (versus sicher, bestätigt, zufrieden, entspannt und zufrieden mit sich selbst) fühlten Sie sich während der Ausführung der Geste?



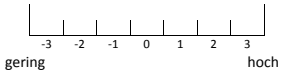
Geistige Anforderungen Technik: Ziehen

Wie viel geistige Anstrengung war für die korrekte Ausführung der Geste erforderlich (z.B. Denken, Entscheiden, Rechnen, Erinnern, Hinsehen, Suchen …)? War die Geste leicht oder anspruchsvoll, einfach oder komplex, erfordert sie hohe Genauigkeit oder ist sie fehlertolerant?



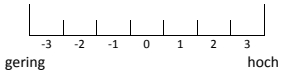
Körperliche Anforderungen

Wie viel körperliche Aktivität war erforderlich? War die Geste körperlich leicht oder schwer, einfach oder anstrengend, erholsam oder mühselig?



Zeitliche Anforderungen

Wie empfanden Sie den Zeitaufwand für die Ausführung der Geste?

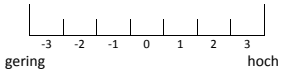


Ausführung der Aufgaben

Wie erfolgreich haben Sie ihrer Meinung nach die Aufgabe erreicht? Wie zufrieden waren Sie mit Ihrer Leistung bei der Verfolgung dieser Ziele?

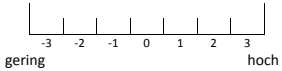
Anstrengung

Wie hart mussten Sie arbeiten, um Ihren Grad an Aufgabenerfüllung zu erreichen?



Frustration

Wie unsicher, entmutigt, irritiert, gestresst und verärgert (versus sicher, bestätigt, zufrieden, entspannt und zufrieden mit sich selbst) fühlten Sie sich während der Ausführung der Geste?



Geistige Anforderungen Technik: Warten

Wie viel geistige Anstrengung war für die korrekte Ausführung der Geste erforderlich (z.B. Denken, Entscheiden, Rechnen, Erinnern, Hinsehen, Suchen …)? War die Geste leicht oder anspruchsvoll, einfach oder komplex, erfordert sie hohe Genauigkeit oder ist sie fehlertolerant?

Körperliche Anforderungen

Wie viel körperliche Aktivität war erforderlich? War die Geste körperlich leicht oder schwer, einfach oder anstrengend, erholsam oder mühselig?

Zeitliche Anforderungen

Wie empfanden Sie den Zeitaufwand für die Ausführung der Geste?

Ausführung der Aufgaben

Wie erfolgreich haben Sie ihrer Meinung nach die Aufgabe erreicht? Wie zufrieden waren Sie mit Ihrer Leistung bei der Verfolgung dieser Ziele?

Anstrengung

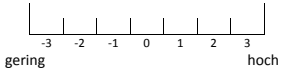
Wie hart mussten Sie arbeiten, um Ihren Grad an Aufgabenerfüllung zu erreichen?

Frustration

Wie unsicher, entmutigt, irritiert, gestresst und verärgert (versus sicher, bestätigt, zufrieden, entspannt und zufrieden mit sich selbst) fühlten Sie sich während der Ausführung der Geste?

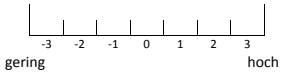
Geistige Anforderungen Technik: Zeigen

Wie viel geistige Anstrengung war für die korrekte Ausführung der Geste erforderlich (z.B. Denken, Entscheiden, Rechnen, Erinnern, Hinsehen, Suchen …)? War die Geste leicht oder anspruchsvoll, einfach oder komplex, erfordert sie hohe Genauigkeit oder ist sie fehlertolerant?



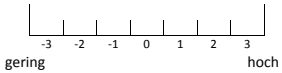
Körperliche Anforderungen

Wie viel körperliche Aktivität war erforderlich? War die Geste körperlich leicht oder schwer, einfach oder anstrengend, erholsam oder mühselig?



Zeitliche Anforderungen

Wie empfanden Sie den Zeitaufwand für die Ausführung der Geste?

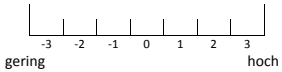


Ausführung der Aufgaben

Wie erfolgreich haben Sie ihrer Meinung nach die Aufgabe erreicht? Wie zufrieden waren Sie mit Ihrer Leistung bei der Verfolgung dieser Ziele?

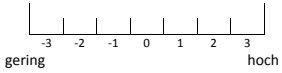
Anstrengung

Wie hart mussten Sie arbeiten, um Ihren Grad an Aufgabenerfüllung zu erreichen?



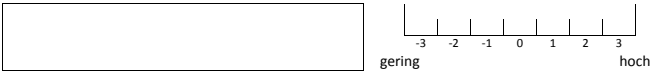
Frustration

Wie unsicher, entmutigt, irritiert, gestresst und verärgert (versus sicher, bestätigt, zufrieden, entspannt und zufrieden mit sich selbst) fühlten Sie sich während der Ausführung der Geste?



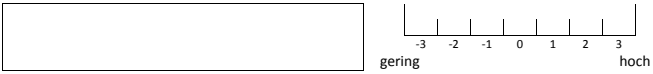
Geistige Anforderungen Technik: Handknicken

Wie viel geistige Anstrengung war für die korrekte Ausführung der Geste erforderlich (z.B. Denken, Entscheiden, Rechnen, Erinnern, Hinsehen, Suchen …)? War die Geste leicht oder anspruchsvoll, einfach oder komplex, erfordert sie hohe Genauigkeit oder ist sie fehlertolerant?



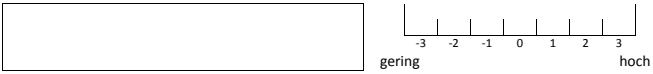
Körperliche Anforderungen

Wie viel körperliche Aktivität war erforderlich? War die Geste körperlich leicht oder schwer, einfach oder anstrengend, erholsam oder mühselig?



Zeitliche Anforderungen

Wie empfanden Sie den Zeitaufwand für die Ausführung der Geste?

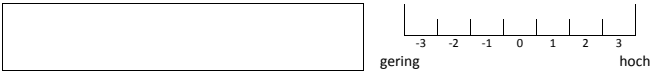


Ausführung der Aufgaben

Wie erfolgreich haben Sie ihrer Meinung nach die Aufgabe erreicht? Wie zufrieden waren Sie mit Ihrer Leistung bei der Verfolgung dieser Ziele?

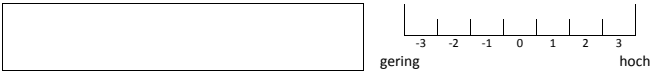
Anstrengung

Wie hart mussten Sie arbeiten, um Ihren Grad an Aufgabenerfüllung zu erreichen?



Frustration

Wie unsicher, entmutigt, irritiert, gestresst und verärgert (versus sicher, bestätigt, zufrieden, entspannt und zufrieden mit sich selbst) fühlten Sie sich während der Ausführung der Geste?



Geistige Anforderungen Technik: Tippen

Wie viel geistige Anstrengung war für die korrekte Ausführung der Geste erforderlich (z.B. Denken, Entscheiden, Rechnen, Erinnern, Hinsehen, Suchen …)? War die Geste leicht oder anspruchsvoll, einfach oder komplex, erfordert sie hohe Genauigkeit oder ist sie fehlertolerant?

Körperliche Anforderungen

Wie viel körperliche Aktivität war erforderlich? War die Geste körperlich leicht oder schwer, einfach oder anstrengend, erholsam oder mühselig?

Zeitliche Anforderungen

Wie empfanden Sie den Zeitaufwand für die Ausführung der Geste?

Ausführung der Aufgaben

Wie erfolgreich haben Sie ihrer Meinung nach die Aufgabe erreicht? Wie zufrieden waren Sie mit Ihrer Leistung bei der Verfolgung dieser Ziele?

Anstrengung

Wie hart mussten Sie arbeiten, um Ihren Grad an Aufgabenerfüllung zu erreichen?

Frustration

Wie unsicher, entmutigt, irritiert, gestresst und verärgert (versus sicher, bestätigt, zufrieden, entspannt und zufrieden mit sich selbst) fühlten Sie sich während der Ausführung der Geste?

Geistige Anforderungen Technik: Greifen

Wie viel geistige Anstrengung war für die korrekte Ausführung der Geste erforderlich (z.B. Denken, Entscheiden, Rechnen, Erinnern, Hinsehen, Suchen …)? War die Geste leicht oder anspruchsvoll, einfach oder komplex, erfordert sie hohe Genauigkeit oder ist sie fehlertolerant?

Körperliche Anforderungen

Wie viel körperliche Aktivität war erforderlich? War die Geste körperlich leicht oder schwer, einfach oder anstrengend, erholsam oder mühselig?

Zeitliche Anforderungen

Wie empfanden Sie den Zeitaufwand für die Ausführung der Geste?

Ausführung der Aufgaben

Wie erfolgreich haben Sie ihrer Meinung nach die Aufgabe erreicht? Wie zufrieden waren Sie mit Ihrer Leistung bei der Verfolgung dieser Ziele?

Anstrengung

Wie hart mussten Sie arbeiten, um Ihren Grad an Aufgabenerfüllung zu erreichen?

Frustration

Wie unsicher, entmutigt, irritiert, gestresst und verärgert (versus sicher, bestätigt, zufrieden, entspannt und zufrieden mit sich selbst) fühlten Sie sich während der Ausführung der Geste?

Geistige Anforderungen Technik: Rotieren

Wie viel geistige Anstrengung war für die korrekte Ausführung der Geste erforderlich (z.B. Denken, Entscheiden, Rechnen, Erinnern, Hinsehen, Suchen …)? War die Geste leicht oder anspruchsvoll, einfach oder komplex, erfordert sie hohe Genauigkeit oder ist sie fehlertolerant?

Körperliche Anforderungen

Wie viel körperliche Aktivität war erforderlich? War die Geste körperlich leicht oder schwer, einfach oder anstrengend, erholsam oder mühselig?

Zeitliche Anforderungen

Wie empfanden Sie den Zeitaufwand für die Ausführung der Geste?

Ausführung der Aufgaben

Wie erfolgreich haben Sie ihrer Meinung nach die Aufgabe erreicht? Wie zufrieden waren Sie mit Ihrer Leistung bei der Verfolgung dieser Ziele?

Anstrengung

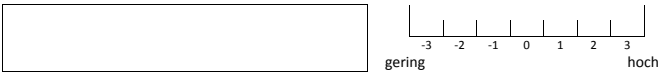
Wie hart mussten Sie arbeiten, um Ihren Grad an Aufgabenerfüllung zu erreichen?

Frustration

Wie unsicher, entmutigt, irritiert, gestresst und verärgert (versus sicher, bestätigt, zufrieden, entspannt und zufrieden mit sich selbst) fühlten Sie sich während der Ausführung der Geste?

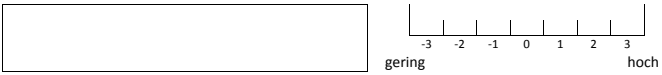
Geistige Anforderungen Technik: Pistole

Wie viel geistige Anstrengung war für die korrekte Ausführung der Geste erforderlich (z.B. Denken, Entscheiden, Rechnen, Erinnern, Hinsehen, Suchen …)? War die Geste leicht oder anspruchsvoll, einfach oder komplex, erfordert sie hohe Genauigkeit oder ist sie fehlertolerant?



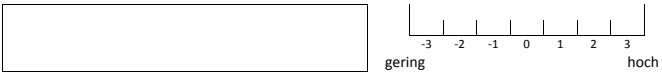
Körperliche Anforderungen

Wie viel körperliche Aktivität war erforderlich? War die Geste körperlich leicht oder schwer, einfach oder anstrengend, erholsam oder mühselig?



Zeitliche Anforderungen

Wie empfanden Sie den Zeitaufwand für die Ausführung der Geste?

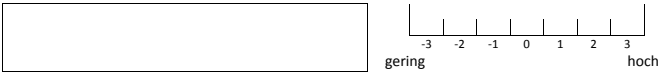


Ausführung der Aufgaben

Wie erfolgreich haben Sie ihrer Meinung nach die Aufgabe erreicht? Wie zufrieden waren Sie mit Ihrer Leistung bei der Verfolgung dieser Ziele?

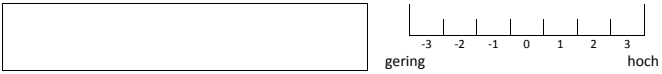
Anstrengung

Wie hart mussten Sie arbeiten, um Ihren Grad an Aufgabenerfüllung zu erreichen?



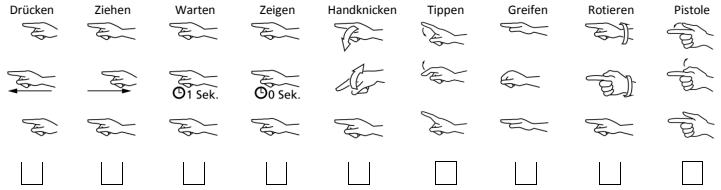
Frustration

Wie unsicher, entmutigt, irritiert, gestresst und verärgert (versus sicher, bestätigt, zufrieden, entspannt und zufrieden mit sich selbst) fühlten Sie sich während der Ausführung der Geste?

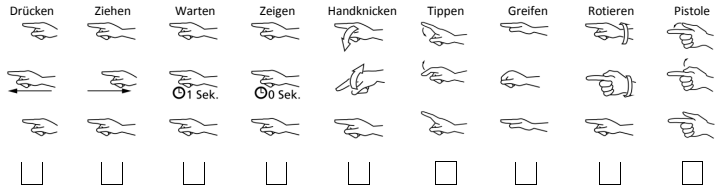


Abschließende Fragen

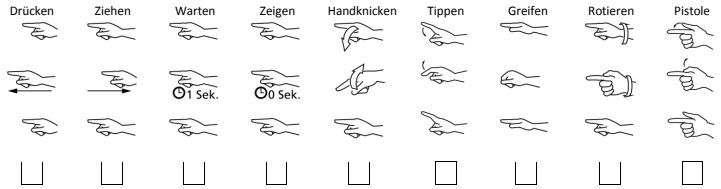
Welche Technik hat Ihnen am besten gefallen?



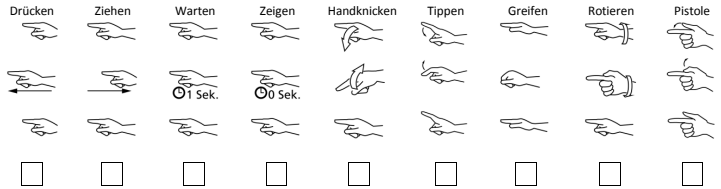
Welche Technik hat Ihnen am wenigsten gefallen?



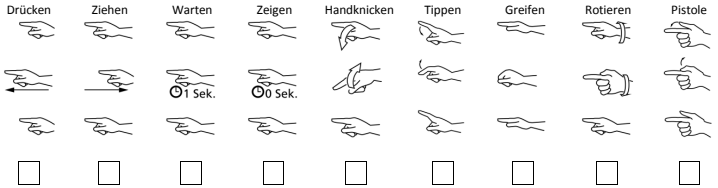
Welche Technik hat Sie geistig am meisten gefordert?



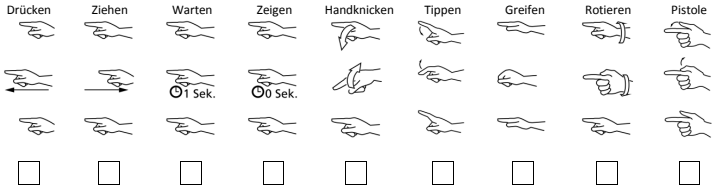
Welche Technik hat Sie geistig am wenigsten gefordert?



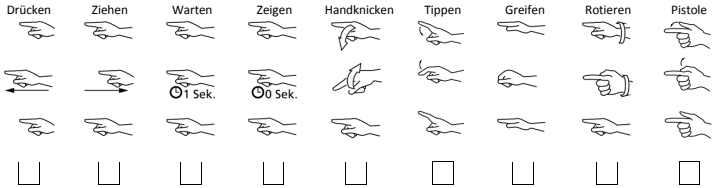
Welche Technik war körperlich am meisten fordernd?



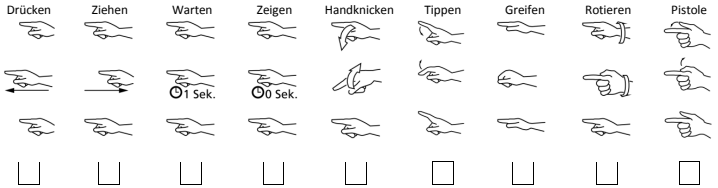
Welche Technik war körperlich am wenigsten fordernd?



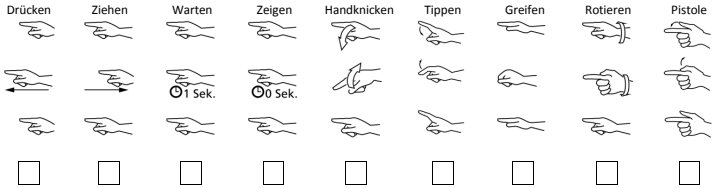
Welche Technik empfanden Sie am schnellsten?



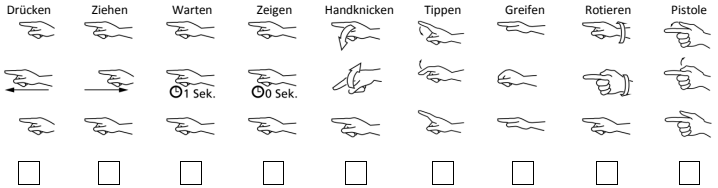
Welche Technik empfanden Sie am langsamsten?



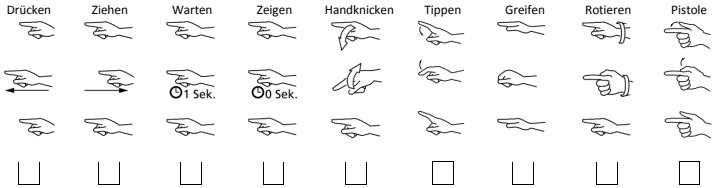
Mit welcher Technik konnten Sie die Aufgabe am besten erfüllen?



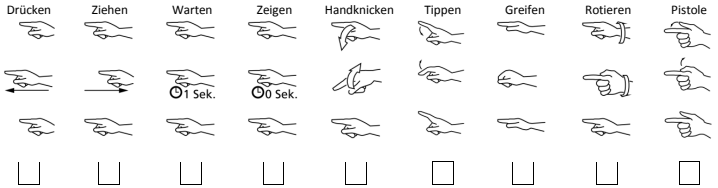
Mit welcher Technik konnten Sie die Aufgabe am schlechtesten erfüllen?



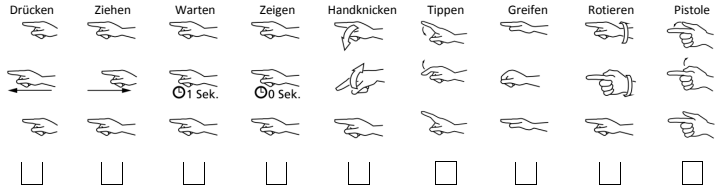
Welche Technik war am meisten anstrengend?



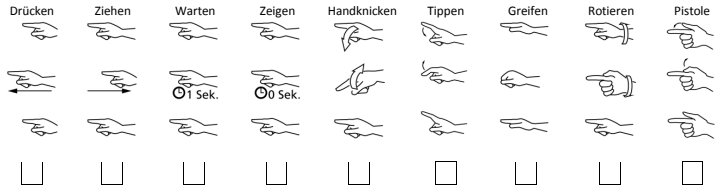
Welche Technik war am wenigsten anstrengend?



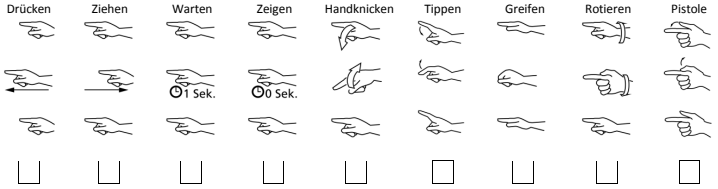
Welche Technik frustrierte Sie am meisten?



Welche Technik frustrierte Sie am wenigsten?

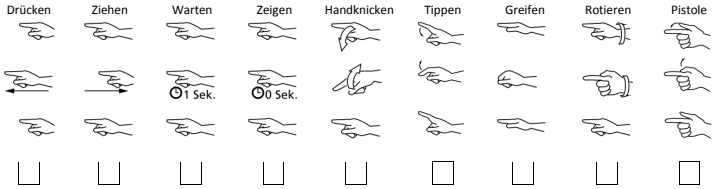


Mit welchen Techniken könnten Sie sich vorstellen über einen längeren Zeitraum zu arbeiten (Mehrfachnennung möglich)?



Bitte bringen Sie die Techniken in die Reihenfolge, in der sie Ihnen am besten gefallen haben.

(1 = am besten gefallen, 9 = am wenigsten gefallen)



Haben Sie weitere Anmerkungen oder Kommentare?

A.5 User study - Acceptable Latencies

Testszenario 1 – Nachbefragung

Fragen zur Person

Alter: _____

Geschlecht: _____

Fragen zur Erfahrungen im Bereich Computer

Ist Ihnen der Umgang mit dem PC vertraut? (Schulnote 1-6): _____

Wie häufig arbeiten Sie an Dual-Monitor-Systemen ?

	1	2	3	4	5	6	7	
sehr häufig	☐	☐	☐	☐	☐	☐	☐	nie

Wie häufig arbeiten Sie an Systemen mit mehr als zwei Monitoren?

	1	2	3	4	5	6	7	
sehr häufig	☐	☐	☐	☐	☐	☐	☐	nie

Fragen zum Spiel

Gab es Probleme beim Verstehen der geforderten Aufgabe?

_____
_____
_____

Hat Sie am Spiel etwas gestört?

_____
_____
_____

Hat Sie das Spiel gelangweilt?

	1	2	3	4	5	6	7	
ja, sehr stark	☐	☐	☐	☐	☐	☐	☐	nein, gar nicht

Ordnen sie alle Farben, bezüglich der Übertragungsgeschwindigkeit, aufsteigend an.

***Beispiel**.: rot war am schnellsten, grün am zweit schnellsten, blau und gelb am dritt schnellsten!*

***Beispiel-Lösung**: rot, grün, (gelb;blau).*

Farben siehe Bildschirm:

_____

_____

_____

Bitte ordnen sie den Farben der Formen Werte zu(1-7). Hat ihnen das Übertragen zu lange gedauert oder waren sie mit der Übertragungszeit zufrieden?

	1	2	3	4	5	6	7		
sehr schnell	□	□	□	□	□	□	□	sehr langsam	**Rot**
sehr schnell	□	□	□	□	□	□	□	sehr langsam	**Grün**
sehr schnell	□	□	□	□	□	□	□	sehr langsam	**Schwarz**
sehr schnell	□	□	□	□	□	□	□	sehr langsam	**Blau**
sehr schnell	□	□	□	□	□	□	□	sehr langsam	**Gelb**

Zusatzfragen

Welche Farbe empfanden sie als gerade noch akzeptabel?

_____

_____

Haben sie einen Unterschied zwischen Kreis und Rechteck bezüglich der durchschnittlichen Übertragungszeit feststellen können? (Bitte nur eine Antwort ankreuzen)

□ Rechtecke sind durchschnittlich schneller als Kreise übertragen worden.

□ Kreise sind durchschnittlich schneller als Rechtecke übertragen worden.

□ Kreise und Rechtecke sind durchschnittlich gleich-schnell Übertragen worden.

A.6 User study - Bridging Latencies

Testszenario 1 – Nachbefragung

Fragen zur Person

Alter: _____

Geschlecht: _____

Fragen zur Erfahrungen im Bereich Computer

Ist Ihnen der Umgang mit dem PC vertraut? (Schulnote 1-6): _____

Wie häufig arbeiten Sie an Dual-Monitor-Systemen ?

	1	2	3	4	5	6	7	
sehr häufig	☐	☐	☐	☐	☐	☐	☐	nie

Fragen zum Spiel

Gab es Probleme beim Verstehen der geforderten Aufgabe?

_____
_____
_____

Hat Sie am Spiel etwas gestört?

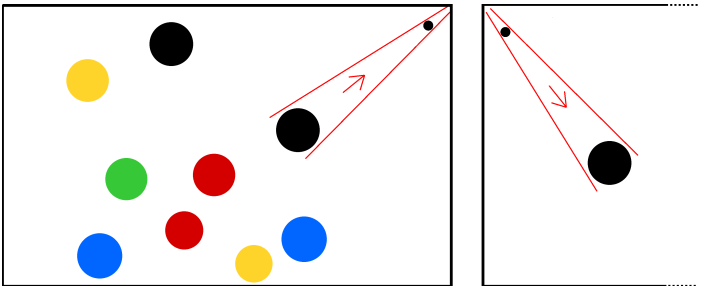
_____
_____
_____

Hat Sie das Spiel gelangweilt?

	1	2	3	4	5	6	7	
ja, sehr stark	☐	☐	☐	☐	☐	☐	☐	nein, gar nicht

Animationen

Shrink-Out/In (Schrumpfen)



Wie empfanden sie die Übertragungsgeschwindigkeit?

	1	2	3	4	5	
sehr schnell	☐	☐	☐	☐	☐	sehr langsam

Wie gut hat Ihnen diese Animation gefallen?

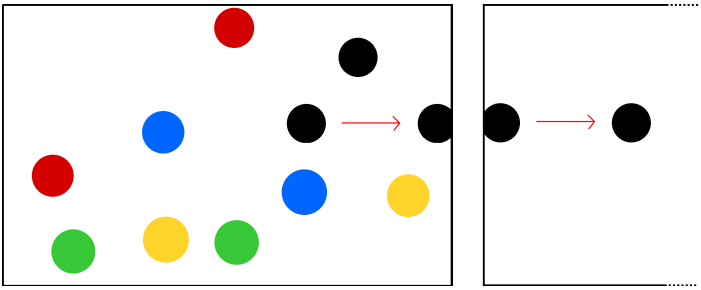
	1	2	3	4	5	
sehr gut	☐	☐	☐	☐	☐	sehr schlecht

Wie intuitiv verständlich hat die Animation auf sie gewirkt?

	1	2	3	4	5	
sehr verständlich	☐	☐	☐	☐	☐	überhaupt nicht verständlich

Was hat sie an der Animation besonders gestört oder was hat ihnen an der Animation besonders gefallen?

Slide-Out/In (Fliegen)



Wie empfanden sie die Übertragungsgeschwindigkeit?

	1	2	3	4	5	
sehr schnell	☐	☐	☐	☐	☐	sehr langsam

Wie gut hat Ihnen diese Animation gefallen?

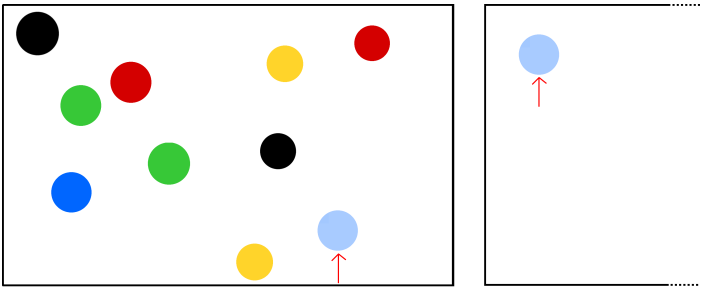
	1	2	3	4	5	
sehr gut	☐	☐	☐	☐	☐	sehr schlecht

Wie intuitiv verständlich hat die Animation auf sie gewirkt?

	1	2	3	4	5	
sehr verständlich	☐	☐	☐	☐	☐	überhaupt nicht verständlich

Was hat sie an der Animation besonders gestört oder was hat ihnen an der Animation besonders gefallen?

Fade-Out/In (Einblenden / Ausblenden)



Wie empfanden sie die Übertragungsgeschwindigkeit?

	1	2	3	4	5	
sehr schnell	☐	☐	☐	☐	☐	sehr langsam

Wie gut hat Ihnen diese Animation gefallen?

	1	2	3	4	5	
sehr gut	☐	☐	☐	☐	☐	sehr schlecht

Wie intuitiv verständlich hat die Animation auf sie gewirkt?

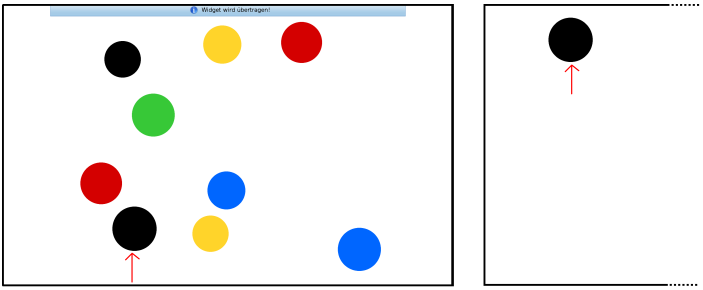
	1	2	3	4	5	
sehr verständlich	☐	☐	☐	☐	☐	überhaupt nicht verständlich

Was hat sie an der Animation besonders gestört oder was hat ihnen an der Animation besonders gefallen?

_____

_____

Notification (Benachrichtigung)



Wie empfanden sie die Übertragungsgeschwindigkeit?

	1	2	3	4	5	
sehr schnell	☐	☐	☐	☐	☐	sehr langsam

Wie gut hat Ihnen diese Animation gefallen?

	1	2	3	4	5	
sehr gut	☐	☐	☐	☐	☐	sehr schlecht

Wie intuitiv verständlich hat die Animation auf sie gewirkt?

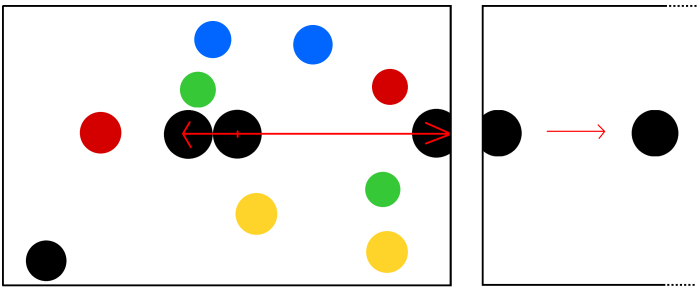
	1	2	3	4	5	
sehr verständlich	☐	☐	☐	☐	☐	überhaupt nicht verständlich

Was hat sie an der Animation besonders gestört oder was hat ihnen an der Animation besonders gefallen?

_____

_____

Slingshot (Schleuder)



Wie empfanden sie die Übertragungsgeschwindigkeit?

	1	2	3	4	5	
sehr schnell	☐	☐	☐	☐	☐	sehr langsam

Wie gut hat Ihnen diese Animation gefallen?

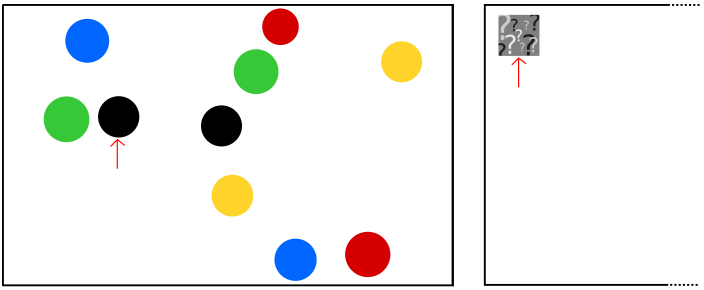
	1	2	3	4	5	
sehr gut	☐	☐	☐	☐	☐	sehr schlecht

Wie intuitiv verständlich hat die Animation auf sie gewirkt?

	1	2	3	4	5	
sehr verständlich	☐	☐	☐	☐	☐	überhaupt nicht verständlich

Was hat sie an der Animation besonders gestört oder was hat ihnen an der Animation besonders gefallen?

Placeholder (Platzhalter)



Wie empfanden sie die Übertragungsgeschwindigkeit?

	1	2	3	4	5	
sehr schnell	☐	☐	☐	☐	☐	sehr langsam

Wie gut hat Ihnen diese Animation gefallen?

	1	2	3	4	5	
sehr gut	☐	☐	☐	☐	☐	sehr schlecht

Wie intuitiv verständlich hat die Animation auf sie gewirkt?

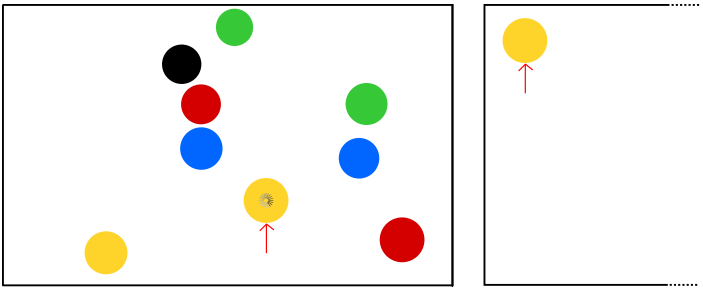
	1	2	3	4	5	
sehr verständlich	☐	☐	☐	☐	☐	überhaupt nicht verständlich

Was hat sie an der Animation besonders gestört oder was hat ihnen an der Animation besonders gefallen?

_____

_____

Loading-Animation (Ladeanimation)



Wie empfanden sie die Übertragungsgeschwindigkeit?

	1	2	3	4	5	
sehr schnell	☐	☐	☐	☐	☐	sehr langsam

Wie gut hat Ihnen diese Animation gefallen?

	1	2	3	4	5	
sehr gut	☐	☐	☐	☐	☐	sehr schlecht

Wie intuitiv verständlich hat die Animation auf sie gewirkt?

	1	2	3	4	5	
sehr verständlich	☐	☐	☐	☐	☐	überhaupt nicht verständlich

Was hat sie an der Animation besonders gestört oder was hat ihnen an der Animation besonders gefallen?

_____

_____

Bitte ordnen sie den Animationen nach Gefallen die Zahlen 1(was hat Ihnen am Besten gefallen) bis 7(was hat Ihnen am schlechtesten gefallen) ein Ranking zu. Jede Zahl darf nur einmal vorkommen.