Diploma Thesis

Design of Multi-Touch based User Interfaces for 3D-CAD Applications

[David Terbeek]

April 8, 2012

Westfälische Willhelms Universität Münster, Germany
Faculty for Mathematics and Informatics
Department of Visualisation

Supervising Professor : Prof. Dr. Hinrichs
Co-Supervisor: Dipl.-Inf. Dimitar Valkov

Declaration of autonomy - Eidesstattliche Versicherung

I hereby declare to have created the following work by myself, unless stated otherwise. Moreover this work has not been submitted for any other degree.

Hiermit versichere ich, dass ich die vorgelegte Diplomarbeit selbstständig verfasst, sie noch in keinem anderen Prüfungsverfahren vorgelegt, keine anderen als die in der Arbeit aufgeführten Quellen benutzt und sämtliche Zitate und Entlehnungen kenntlich gemacht habe.

Münster, April 8, 2012

David Terbeek

Abstract

Closing the gap between real-world interactions and interactions with virtual environments requires researchers to develop new means to capture the users intentions and supply new interaction concepts which lean on experience made outside the digital domain.

Due to its directness and variable amount of input points multi-touch technology incorporates some of the properties human interaction is relying on. Thus many researchers are focussing on exploring the capabilities of multi-touch to increase usability of various applications in contrast to common input devices. While multi-touch has a profound impact on the way we interact with 2D environments nowadays, we have not yet fully understood how to utilise multi-touch in unlimited 3D environments or, if it is a usable alternative in this context at all.

In this thesis we explore the capabilities of multi-touch input for 3D CAD and other 3D modelling applications. The main idea is that multi-touch could reduce the trade-off between interaction richness and complexity in these applications by combining interaction tasks, which work well together. This way, the need for mode switches might be reduced, which could lead to less GUI strain and more fluent interactions. Direct manipulation of scene content may also counterweight initial cognitive effort required of new users to succeed interacting with these evermore complex applications. Through the design and evaluation of several "mode-switch-free" multi-touch user interfaces, which allow users to perform different object and camera manipulation tasks, we identified many input and application specific design challenges. Although participants in our evaluation were able to solve the required tasks, thus some degree of usability was achieved by our interfaces, it is clear that further consideration in this direction is required to be able to make a well-founded statement on the usability of multi-touch for 3D interactions.

Contents

1 Introduction

Ubiquitous computing aims to make "digital technology" an integral part of our everyday life without noticing it. Technology should not be in the focus of attention, but only serve as an easy-to-use tool, working in the background easing up the actual task of the user. Thus multi-touch, as a tool of direct manipulation, offers new exciting possibilities for progression towards the vision of ubiquitous computing. We believe that multi-touch technology may be able to, as Mark Weiser said, *"weave itself into the fabric of everyday life until it is indistinguishable from it"*, [44]. However, we still have to explore, if and how multi-touch can be used in a comprehensive manner in many applications.

Therefore multi-touch has received considerable attention in the last decade. It super sets using single-touch technology by adding the ability to sense multiple points of direct contact on an interactive surface.
The availability of multiple input points allows combination of interaction tasks which are separated by menus, widgets, gestures and other mechanisms in single point interfaces. An interaction technique combining two tasks, which performance is similar or even better than carrying out the tasks consecutively, might allow us to abandon the referring mode switching technique, ultimately reducing GUI (Graphical User Interface) strain.
With current technologies, such as frustrated total internal reflection (FTIR) [17], diffused illumination (DI) [31], resistive-grid [5] and capacitive surfaces [28], multi-touch became ever more widespread and can currently be found in many devices such as smart phones, media players[1] large display or tablet systems.

The possibility of directly touching graphical representations is often argued to be more "natural" than working with indirect input devices such as keyboard and/or mouse. Moreover, it has been shown that direct input significantly outperforms indirect input for some common interaction tasks [14, 15]. Furthermore, multi-touch surfaces combine the advantages of direct and bi-manual interaction, which also has

[1] `http://www.zune.net/en-us/products/zunehd/`

been shown to improve interaction quality [10], providing developers new means to explore possible user interface designs.

Through commercial successful products such as Apple's iPhone, Rotate-Scale-Translate (RST) manipulations established a new way of interacting with two dimensional (2D) context, supporting the appeal of direct manipulation [35].

In addition to the point of contact, multiple additional attributes of the touch may be detected, most noticeably shape [36, 46] , angle of approach[24, 30] and the applied pressure [20], offering even more input information to further extend interaction richness.

However, common problems, such as the fat-finger problem [6] or the absence of visual feedback [45] and the latency involved when capturing touch input, are important factors influencing the design and quality of multi-touch interaction techniques.

While the two touch RST technique is currently wide accepted for 2D interaction, there does not seem not be a straight forward extension of this approach for interaction with three dimensional (3D) content. Due to the limitation of the user's actions being constrained by a two dimensional surface, interacting with 3D contexts may reduce the directness and ease-of-use by forcing the user to learn additional and/or more complex input techniques to access all dimensions.

3D modelling applications such as Autodesk Maya or Blender and especially 3D CAD applications like Autodesk AutoCad are complex and rich on functionality, thus require a lot of dedication and training to be used effectively. Since major qualities of multi-touch interaction are its directness and ease-of-use we are motivated to find out, if or to which extent multi-touch can be used to ease- and/or speed-up certain 3D interaction tasks as they are performed in 3D CAD and modelling applications. Although multiple view-ports are commonly used in the professional segment a perspective view is more immersive and might, in combination with multi-touch interaction, serve as a compelling starting point for new users, since this combination might require less cognitive effort. By performing several tasks simultaneously, similar to the RST technique, multi-touch could speed up design processes which require less precision, but fast results such as prototyping and digital storyboarding.

In this diploma thesis we will design and evaluate three multi-touch interfaces which allow users to perform common object manipulation tasks. Since camera navigation is an integral part of 3D CAD / modelling applications our prototype interfaces offer navigational metaphors for this purpose. We conducted a preliminary online survey to identify suitable gesture candidates and gain a first impression on user expectations to refine our prototype design. Although we could not draw definitive conclusions from the evaluative comparison of our refined interface prototypes, participants were in general able to solve the 3D navigation and docking tasks of the trial. Despite problems with the touch and gesture recognition systems, participants experienced interaction with a 3D environment solely with multi-touch without a mode switch for the first time and made many valuable suggestions for future designs.

The remainder of this thesis is structured as follows: The second chapter provides an overview about related work and outlines some basic design parameters and coherences which have been identified by previous studies focusing on multi-touch technology. The third chapter explains which factors and properties are important to the design of multi-touch user interface for 3D CAD applications, so that an initial design concept can be formulated. Chapter four provides methods and results of our online survey. The fifth chapter describes the design and implementation of our three prototype interfaces, followed by evaluation results of these interfaces in chapter six. The last chapter summarises our work and provides an outline of future elaborations.

2 Interaction Paradigms & Related Work

2.1 Human Computer Interaction

Human Computer Interaction (HCI) aims to enable interaction with a system so that the user can reach their desired goals with minimal effort. Hereby interaction is not limited to the input or user part of the action, but also on the output the system provides forming a cycle of action and perception. Abowd and Beale [1] formulate a model for this interaction cycle, which is shown in figure 2.1.

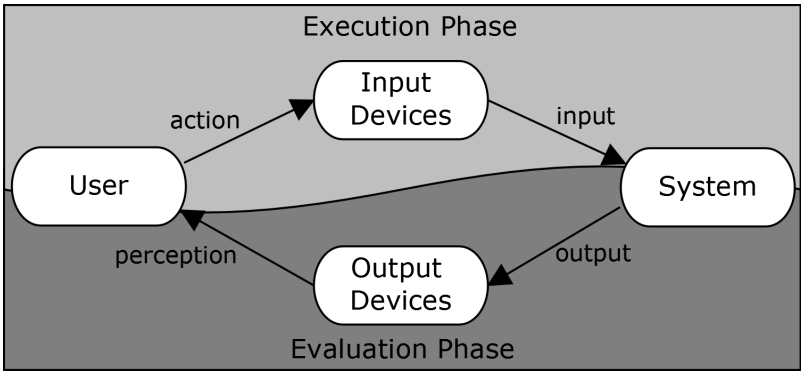

Figure 2.1: The Interaction Cycle by Abowd and Beale [1]

The cycle consists of two stages: the execution phase and the evaluation phase. In both phases user and system are involved. Input and output devices together form an interface through which communication takes place.

The interaction starts off with the execution phase and is initiated by the user aiming to reach an objective, hence formulating a task to reach this objective by means of the system. One or more input devices are then used to change the systems state. In the evaluation phase the changed system state represents itself through the output device. The user interprets this output and compares it relative to his objective. A new interaction cycle can start as long as the objective has not been reached [40].

This cycle however, only explains the general progress of interaction. The layered model, as shown in figure 2.2 can help identify software and hardware requirements of the system [2].

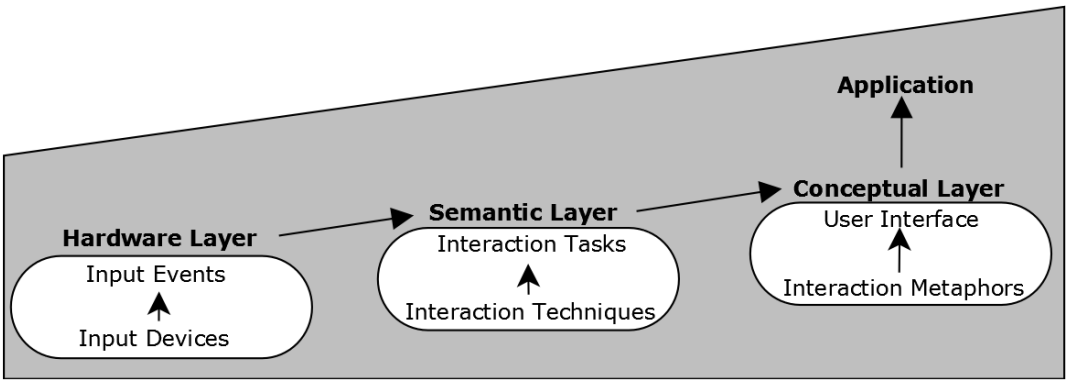

Figure 2.2: The layered interaction model [1]

The models bottom layer is called hardware or physical layer. Following the hardware layer are the semantic and the conceptual layer. The first layer is defined by the systems input devices and their input events such as mouse/touch motion, touch appearance and buttons pressed. Interaction tasks like translation and rotation can be performed within the system by the usage of several interaction techniques in the second layer. There are some basic interactions as pointing and clicking which are indivisible hence they cannot be decomposed into smaller units. The interaction techniques however, are not limited to these basic interactions. Instead these basic techniques can be used to build more complex interaction techniques, for example the two finger pinch. The third layer describes the system from the users point of view. Several interaction techniques can be combined to form a contextual interaction metaphor: e.g a camera navigation metaphor or a object translation metaphor, if the metaphors underlying interaction tasks can be performed without a mode switch. All metaphors/interaction techniques together form the user interface - the part of the system which is closest related to the user and best described from the users point of view. [40].

2.2 Usability

Besides investigating the ability of various technologies to reduce the barrier between humans and computers, user interface design is a another major aspect of HCI, as user

interfaces provide a bridge between input devices and application.

In general, it is easy to create a user interface. The only thing to do is give the user the possibility to see and change the states of the system. It is hard to create a good interface in particular, because it matters in which way access to the system is given to the user. Even the tiniest changes to a user interface could alter the way users experience interaction with an application. Roughly speaking, every interface design decision made in reference to the layered model might influence "usability" in some manner.
With a variety of input devices available, developers are encouraged to consider which device(s) fit their application best in order to reduce complexity and overhead of interactions potentially increasing overall performance, learnability and motivation when using the application. Usability is often mentioned together with "intuitiveness". Since everyone has a personal understanding of the meaning of "intuitiveness", usually, a definition is ambiguous and can be misinterpreted. Therefore we give the following definition of usability suitable for our context.

Usability is a relative metric over the comparison of the following two user interface properties:

+ Functionality - Providing a set of interaction tasks that allow the user to carry out his tasks.

+ Ease-of-Use - Providing access to these tasks in such a a way that it is easy to learn the system and easy to carry out the tasks.

If the user cannot figure out how to reach his goal or finds it too complicated, it can be considered a usability problem. Or, if users are forced to make a decision which does not help to solve the actual task - for example unnecessary confirmation dialogues - the user might get frustrated simply because reading a text box was not his intention. Spolsky puts this together as: *"Choice is important for users, but providing too much options could just be a developers excuse to not being able to find a good solution by himself. Users should only make choices which are relevant to their tasks thus interface design is the art of making choices" [38].*

Independent of the quality of the interface, metaphor or interaction technique, the user would need a certain amount of time to "learn" using the technique(s). Knowing the constraints and properties of an interface / interaction technique helps to plan tasks

ahead possibly reducing necessary corrections and redundant interactions. The rate at which users do significantly improve their performance, may be used as a relative measurement for the interface's ease-of-use.
Besides ease-of-use and functionality, subjective motivation to use a technique should be considered, too, when trying to make an assumption on its usability. Since users might simply prefer the technique which is more fun, but still performs well enough [34]. On the same matter the reverse can be said. Although a technique/interface might have some usability issues its application's purpose could be of such great interest that its used despite of usability problems. Maybe because no alternative is available, maybe because the issues do not occur when using the main features of the application. [11, 19].
Therefore "understanding and weighing the relative implications on use of the input devices properties is necessary in order to make informed decisions" [9].

There are several examples for touch- especially multi-touch technology increasing interaction richness and ease-of-use in many 2D applications. However, higher grade of usability is not simply guaranteed by using multi-touch technology in contrast to common input devices - it has to be carefully evaluated, if and to what extend touch input can improve user experience and performance in various applications.
We have decided to explore the usability of multi-touch for 3D applications especially 3D CAD and 3D modeling application. We think that multi-touch, if utilised in the right fashion, could ease up interactions with these applications due to its directness in comparison to common input devices. It is also thinkable that multi-touch could increase work flow, leading to faster results because of the ability to manipulate several properties at once. But, although it sounds reasonable that adapting features like direct interaction into the design of 3D interaction techniques may effect usability in a positive way, we have to consider that it may not - and come up with alternative designs. An example could be the indirect "Z-Technique" for 3D object translations by Martinet et al. [29].

2.3 Related Work

When using input devices like mouse and keyboard or even more specific devices, such as a 3D-mouse to interact with virtual environments (VE) , users most of the time cannot rely on experience gained in their everyday life. Instead they are forced to attain new skills in order to be able to interact with the VE. Hence interaction with a VE might get inconsistent with interaction with the rest of our environment. Tangible

User Interfaces (TUI) try to reduce this gap by making user interfaces more reliant on everyday interactions, allowing to get in physical contact with the "atoms" of the virtual world [25, 26]. Therefore TUIs might have a high potential to reduce the trade-off between interaction richness and interaction complexity in numerous applications, by e.g. reducing necessary GUI interactions for a task to be completed.

Similar to TUIs, which use tangible physical forms to interact with virtual environments, touch technology, in the words of Hiroshi Ishii, "tries to coincide input space and output space to realise some form of coupling of physical and digital worlds" [25]. In the context of multi-touch the coincidence between input and output space mostly refers to directness of input and additional degrees of freedom to mimic real world interactions, thus reducing the need for explicit interaction mode switches (buttons, scroll-bars, etc) and therefore reducing GUI strain.
TUI's however, currently serve very specialised purposed applications - e.g using physical models in urban planning [27]. Although multi-touch may be used in a less specialised manner - direct interaction involves the use of existing skills and work practices.

Working with mouse and keyboard, users usually handle tasks in a sequential manner, however, additional degrees of freedom through multiple points of contact allow certain tasks to be carried out simultaneously. Therefore multi-touch greatly expands the types of gestures that we can use in interactions, which might ultimately lead to an more eloquent interaction vocabulary in contrast to common gestures such as pushing a button and dragging, which we use nowadays.[9]. On the one hand touch technology can be used to simulate mouse like behaviour. But as usability of the same interface may change from device to device [9] using touch or multi-touch input as a replacement for mouse and keyboard suffers from the same difficulties touch input has to deal with in general. Problems like occlusion by the finger, input lag and missing tactile feedback may require various techniques assisting the user on his tasks to be able to provide usable alternatives for input, if a mouse and keyboard is not viable or available [6]. On the other hand instead of adapting usage of touch input to the existing interface the interface can be adapted to touch input. A very recent example for this is the "ribbons" user interface in windows 8[1] which aims to provide improved input quality on tablet computers while maintaining the same functionality as the standard windows UI [12].

[1] http://http://windows.microsoft.com/en-US/windows-8/preview

A lot of work has been done exploring the feasibility of single-touch interaction. Examples for early contributions on touch technology are the "PLATO IV Touch Screen Terminal" [7] and the "one-point touch input of vector information" [20] demonstrating that touch input involves more than input point information. Since they detect touch force in X,Y and Z(i.e.,sheer in X ,Y and pressure in Z) and torque in X,Y and Z in addition to the position in X and Y.
As early as 1983 Nakatani et al. provide a discussion about the properties of touch screen based user interfaces which they called "Soft Machines" for example the "one-to-one" mapping of controls and operations [33].
Though single-touch input provides powerful means to enhance user interaction and experience, we are still bound to the concept of GUI manipulation and its sequential work flow.

Multi-touch loosens this restriction, thus great effort has been put into identifying proper application domains, interaction techniques and gestures for multi-touch input. Especially 2D multi-touch interaction has been explored in several research projects [6, 13, 21].

A detailed overview about previous work on the issue of interactive tabletops is provided by Grossman et al. Moreover, they provide a taxonomy to classifiy the work done in this area, taking any system which presents a 3D virtual environment on or above a horizontal surface into consideration [16]

By merging multi-touch with a physics engine Wilsen et al. [46] offer means to interact with digital objects in ways analogous to manipulation of real objects. Detecting not only touch point information, but also more sophisticated shapes like hand palms and object contours they demonstrate that multi-touch interaction exceeds (multiple) point of contact manipulations.
Because a pen complements some of the design properties of the finger its combined usage has gained more and more attention when manipulating 2D content [8, 21]. Today multi-touch technology is used to support working with a pen/stylus supplying bi-manual interaction by using the non-dominant hand for single- or even multi-touch interaction and a pen in the dominant hand for higher precision tasks [22].
Just recently Benko et al. [3] demonstrated multi user 2D multi-touch interactions on a spherical display further extending the possible applications for touch input.

To address the problems occurring when using single- or multi-touch input, such as the "fat finger problem", several techniques have been introduced [6, 43]. By zooming into the area of interest, or controlling a cursor offset using a second finger pixel accurate targeting and positioning can be achieved, thus reducing inaccuracy by increasing interaction complexity.
Proposing a new model of touch inaccuracy the "Perceived Input Point Model" [24] by Holz et al. explains 67% of the touch inaccuracy that was previously attributed to the fat-finger problem. This points out that touch interaction might not be as inaccurate as assumed. To further decrease touch inaccuracy related to the fat-finger problem "Ripples" was introduced [45] providing visual information to the user about successes and potential errors in their input actions.

In addition to these general problems with touch interaction there are more input technique specific problems to handle.
Although multi-touch interfaces may allow users to e.g. translate, rotate and scale digital objects in a single interaction with a two finger pinch gesture, this freedom represents a problem, when the user intends to perform only a subset of manipulations. Because it is difficult to maintain the same distance between two fingers, a rotate manipulation will most of the time be accompanied by some unintended scaling.
Nacenta et al. explore the use of several gesture recognition techniques to address this issue which they call "separability of spatial manipulations" [32].

Recent research explores the possibilities of multi-touch interaction for interaction with 3D applications. The "Balloon Selection" technique [4] is another example for the various fields of application for multi-touch technology. By using a bi-manual two finger gesture on an interactive surface an object in a 3D volume can be selected, showing how multi-touch can help to reduce error rate of selection tasks in comparison to other known techniques. Extending the balloon selection Strothoff et al. developed the "triangle cursor" which offers 3D object selection and above that, three degrees of freedom (DoFs) translations and object yaw rotations at the same time. [39]

Exploring rotation and translation in shallow (limited) depth environments, Hancock et al. [18] proposed several interaction techniques, ranging from one point translations to more complex techniques with three points of contact allowing various rotations. They point out that users are not only capable of performing more complex (multi-finger, bi-manual) interactions, but also prefer them. The higher number of touches provided users with the opportunity to independently control more DoFs increasing

flexibility of how users decide to perform the interaction.

To improve interaction on large 3D data sets for example medical volume data or astronomical simulations Yu et al. [47] investigate the use of large touch-sensitive displays in this context. Both introduced techniques, for single- and multi-touch, provide the user seven DoFs. By specifying special areas of control they separate interaction tasks to reduce erratic behaviour as described by Nacenta et al.[32].

Martinet et al. explore the design of free 3D positioning techniques for multi-touch displays. In addition to a semi-direct bi-manual positioning technique in perspective view they provide two positioning techniques allowing free 3D position through multiple view-ports. They found evidence that most users prefer the semi-direct over the view port technique because of its perspective view and ease-of-use [29].

Instead of implementing specific multi-touch input techniques Reisman et al. extend the principles of 2D RST interaction to 3D by offering a more generalized method. Their model allows manipulation of scene content regardless of the amounts of fingers on the surface. This is possible because their model ensures that any of the initial points of contact stay underneath the fingers during the interaction. The resulting manipulation is then derived by solving a minimisation problem between the possible manipulations[35].

Nowadays visualisation of 3D content is not restricted to a 2D projection on the screen, but rather can be rendered stereoscopically, so that users perceive the output as a volume with spatial depth. In this context Valkov et al. investigated the possibilities of combining multi-touch interaction with stereoscopic vision on wall-sized displays [42].

3 Initial Design Considerations

If a user cannot figure out how to perform a task or finds it too complex it could ultimately lead to frustration and abandonment of the application.
Multi-touch could allow users to interact with various applications in a more compelling and easier way allowing them to faster achieve initial results. This might motivate them to keep on using an application, if another alternative is available. Moreover, touch input could reduce GUI strain, thus reduce interaction complexity in contrast to the cognitive load put on new users nowadays, notably in complex 3D applications like AutoCAD.
However, it is probable that multi-touch interaction will not be used as a stand-alone input concept for 3D CAD/modelling applications, but rather accompany existing devices as they may be used for different purposes within the application. Because different tasks within an application might benefit from certain input device properties to different extends [9]. Multi-touch could be used for making large and swift but rough adjustments to get a view on alternative designs or for digital storyboarding with a specialised interaction task subset optimized for "quick and dirty" prototyping.

This chapter will discuss some of the relations and properties which can influence the usability of a multi-touch interface in the context of 3D manipulations. Moreover, we will line out an initial design concept for an interface which allows several camera and object manipulations without a mode switch. This initial design will act as a baseline for our prototype implementation.

3.1 Camera-Based Multi-Touch Setups

For detecting touch contact and movement on a screen several technologies are available. Each of them has specific attributes which could significantly influence usability, therefore we have to consider which technology fits our purpose best.
Capacitive or resistive multi-touch displays could offer high performance touch recognition by electrical signalling, but optical recognition systems are much more cost-effective and therefore more wide spread in larger display systems nowadays.
FTIR and DI (see figures 3.1, 3.2) are two ways of optical touch recognition with infrared

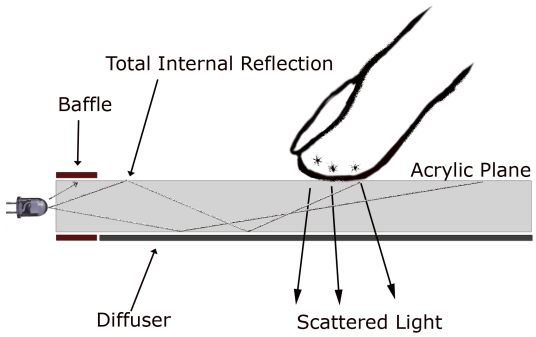

Figure 3.1: FTIR: Light fully reflects at a certain angle and therefore is kept inside the medium until it is reflected by a finger at a different angle.

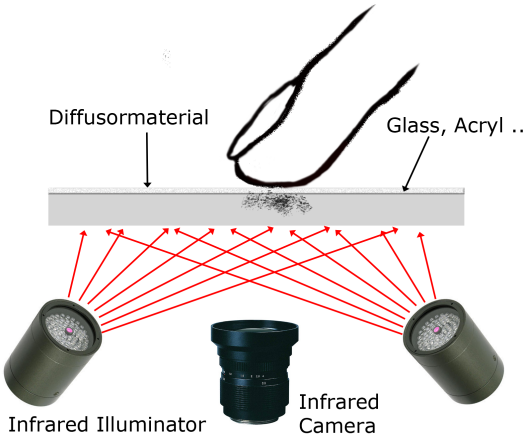

Figure 3.2: DI: Differeces in the amount of reflected light serve as a cue for touch recognition

light. While FTIR facilitates the property of light to reflect completely when trying to cross a medium boundary at a certain angle, diffused Illumination measures the amount of light reflected on a diffusion material surface.

Since both of these technologies are optical capturing systems there are many possibilities to post-process the images tracked to gain meta-information about the touch points, such as point-size which is often taken as a cue to pressure or the recognition of object shapes, so called fiducials. This meta information then can be used to either create new input techniques or stabilise recognition of existing ones. Optical capturing though usually comes at the price of increased input lag and high CPU load. Since image processing at interactive frame rates (30+) is a complex computation task, its usage is accompanied by a trade-off between resolution of the recognition system, which increases theoretical precision and its frame rate, resulting in more or less immediate feedback of the system.

Implementation and evaluation of our interface prototypes requires our design to orient on the hardware limitations induced by the touch screen available for testing. Since the current version of our touch capturing solution delivers touch point position and ID only, we cannot include usage of a stylus or fiducials into our initial design considerations. Moreover, optical tracking always involves some kind of jitter which could cause a touch to be recognised as moving although its finger was not. Furthermore, many light sources are emitting infra-red light, as does the projectors light bulb. Since FTIR and DI depend on light reflections, using an infra-red light absorbing foil in front of the projector's lens may reduce false recognitions and touch losses.

3.2 Multi-Touch with 3D Content

2D input devices are a common choice for interaction with a 2D application, that way the DoF the input device offers (input DoF) can be mapped directly to the applications scene dimensions (output DoF). Therefore choosing a 3D input device for a 3D application may be a promising approach, but there is evidence pointing out that this may not be the case [4].
Multi-touch however, can be seen as an input device with variable input DoF. It has 2*N DoFs, with N constrained by the amount of fingers used.
Mapping these variable degrees of freedom to input tasks so that the user finds the resulting techniques easy to use and replicable, is one of the main challenges when developing a multi-touch interface.

Mapping of Degrees of Freedom

A major advantage of multi-touch is to theoretically be able to manipulate multiple properties at once - making the interaction technique to manipulate these properties in theory more effective than iterative execution of its sub-tasks. Acceptance of the RST technique in 2D applications demonstrates how the additional DoFs can be used to create a touch technique which enables interaction with three properties at the same time without an interaction mode switch. Therefore we want to waive on using widgets or buttons for mode switching in our initial design to investigate whether and to what extend this advantage can be utilised for 3D interactions.

However, when aiming to use multi-touch input for 3D interactions without a mode switch we have to decide between mapping input DoFs to the different interaction tasks and/or to different output DoFs of a single interaction task.
An example: A perspectively projected object on a 2D screen may be translated in three directions either in its local or in the scenes coordinate system, thus a local or global translation has three output DoFs. Straight forward, one touch can be used to manipulate two DoFs. Therefore we have to choose which coordinate system and which output DoF should be manipulated, if we decide to use one touch movement for 3D object translation. We could then use a second touch to manipulate the remaining dimension of the beforehand chosen coordinate system. We might want to use two touch movement for some other manipulation mode though, for example camera orbit or object scaling. On the other hand, if we assign two touch movement to camera orbit we can change the angle from which the object is viewed and map input DoFs to

output DoF depending on the camera viewing direction. That way we can translate an object either locally or globally into every direction with one touch.
This mapping decision problem magnifies itself the more interaction tasks should be accessible without a mode switch.

Separability of Spatial Dimensions

Directly connected to the mapping of input DoFs is an aspect described as separability of spatial dimensions by Nacenta et al. [32]. They observed that using an interaction technique, which is able to manipulate multiple properties at once, most of the time involves an unintended manipulation of at least one of the other properties, if only one property was subjected to change. For example, when trying to use the RST technique for rotation only, the object most likely will be accompanied by some kind of scaling and translation. A highly usable interface however, should reduce unintended effects to a minimum, thus reducing the users needs to correct or even undo their previous actions.
Therefore providing separated interaction techniques for each task and each tasks DoF in combination with techniques supporting simultaneous manipulation of multiple properties at once seems beneficial to the user. Mechanisms such as magnitude "filtering" [32] could hereby help reduce the amount of different gestures the user has to learn, as they separate tasks not by technique but by classification of the input motion type. Thus pinch-zooming with the RST technique may for example not involve a rotational component if the angle between the two touches has not changed "too much" during the interaction.

Illustrating this issue for 3D interactions, we could use a two finger pinch gesture to globally scale an object and apply translation to the object in regards to the midpoint movement of the two touch-points ensuring direct interaction. On the other hand disabling this translation could result in an in place scaling of the object, which also may be a desired result in precision demanding 3D CAD applications.
Differentiating between resting and moving touches could also help separating tasks. We could extend the two finger pinch gesture to support both behaviours described above. Performing a pinch gesture with a resting finger will effect only the scale of the object while simultaneous movement of both fingers will result in scaling and translation.
Separation of tasks and tasks DoF on the one hand could increase precision while on the other hand simultaneous manipulations could speed up the design process.

Finding a good balance between separated or simultaneous manipulations is a major design challenge.

The Fat-Finger Problem

If an object is too small, direct manipulation of it can be difficult, because the user may, for example, not be able to touch with two fingers inside an object to start a scaling technique. If this makes the intended task impossible to be carried out, the user will be forced to increase the screen size of the object by performing another interaction task like camera zoom, thus increasing the overall time the object manipulation task takes to be finished. Allowing to perform a technique anywhere on the screen independent of object position and extends could help reduce these issues. This however, can result in a sacrifice of directness as users are no longer required to touch an object in order to manipulate it. Moreover they are forced to explicitly select or deselect it.

Direct Manipulation in 3D

Finally, there are some situations in 3D applications where the result of direct manipulation may not be congruent with user expectations. If a users camera viewing direction is exactly or nearly exactly aligned with one of the global coordinate system axes and the user intends to translate an object along that axis in a direct manner, it might result in unpredictable translation of the object like jumping to the front and back as extends of the translation plane are very small. This indicates that direct manipulation may not be an unfettered benefactor for multi-touch 3D interactions. In figure 3.3 this matter is illustrated. However, restricting manipulation to the axes which align best with the camera's view increases the need for additional camera translations to access the translation of the remaining axis, possibly increasing overall task manipulation time.

3.3 Basic Interaction Tasks

To be able to make reasonable and informed design decisions, understanding not only the relations and properties influencing usability when using multi-touch is important. Precise knowledge of the interaction tasks which should be accessed through the interface is crucial, too.

Today's CAD applications offer a huge variety of manipulation modes and specialised functions. It remains to be seen whether usable multi-touch interface solutions

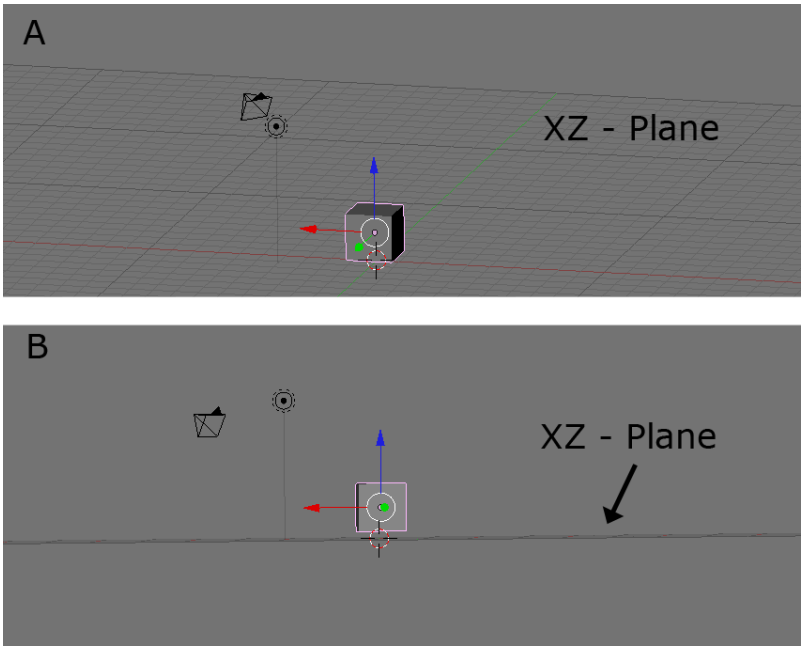

Figure 3.3: The XZ plane can be viewed as a plane in A, but is reduced to a line in B. Since the object is aligned to the global coordinate system its planes have the same alignment as the global planes shown in the figure. Direct translation along the green object axis or the global z-axis can now lead to unexpected behaviour.

which incorporate these amount of possible interaction tasks exist. Identifying how to utilise multi-touch technology in a way the user can benefit from it, requires us to evaluate its usability in a more functional restricted 3D application in order to establish a baseline for future adaptions. This might allow us to identify input techniques which perform within user expectations and secondly reduce the complexity of the previously described "DoF mapping problem" (see 3.2) to a reasonable level.

There are many different 3D design applications for various specialised purposes such as AutoDesk 3DS Max, Maya and AutoCad especially for the professional segment as well as Blender and Google SketchUp, which are free to use. Despite serving different purposes they all have a commonality - basic camera and object manipulations - which are also the first tasks to learn for new users. Most 3D modelling applications offer means to orbit, pan and zoom the camera with various widgets, additionally they offer wide spread navigation metaphors like the "fly", "walk" and "look around" navigation, which can sometimes be used simultaneously depending on the input devices available. Moreover, all applications provide one or more ways to access various basic object translate, scale and rotate operations.
By supporting panning, zooming and orbiting as well as 3DoF object translation,

scaling and rotations a prototype might convey an impression of free 3D interaction with multi-touch input. Providing convenient access to these basic manipulation tasks with multi-touch input might serve as an entry point for new users and ease their learning curve while exploring the possibilities of the 3D (CAD) application.

The following list provides details on every camera navigation and object manipulation task which users should be able to access through our prototype interfaces.

Camera Navigation Tasks

Panning

Figure 3.4 shows the initial and resultant screen content of a camera panning action. This task requires a two DoF input which motion is usually mapped to a local xy-translation of the camera. Either the relative motion of the input device is multiplied by a factor and then added to the cameras position, or in case of direct interaction, the user "grabs" a part of the scene and drags it around. In this method the camera is moved in a way ensuring that the point of the world which got grabbed by the user always stays underneath the input point creating the impression to translate the scene, but not the camera.
To create this impression often the unprojected coordinates of the input point are used. This unprojection converts the 2D motion vector in screen coordinates into a 3D motion vector in world coordinates in respect to the cameras current position. By multiplying this vector with the camera distance to the most perpendicular world plane we can create the impression of dragging that plane. A pseudo code reference is given in listing 3.1.

Let p_0 be the cameras position before the interaction. The new position p_1 after a motion event is given by:

$$p_1 = p_0 + (\Delta(i_1 - i_0) * \lambda$$

with $p_n \in \mathbb{R}^3$, $i_n \in \mathbb{R}^2$ the screen coordinates of input event #n, λ the distance of the camera to the reference plane and Δ the unprojection of the input's motion vector.

```
1 vector2 displayDimensions(){
  real outleft, outright, outtop, outbottom;
  cam->getFrustumExtends(outleft, outright, outtop, outbottom);
  double factor = 1/cam->getNearClipDistance();
  double h = (outtop-outbottom)*factor;
```

```
  double w = (outright-outleft)*factor;
  return vector2((float) w, (float) h));
}
vector3 unprojectVector(vector2 inputpos){
  vector2 dims = displayDimensions();
  vector3 worldVec = cam->getRight()* (float) (0.5 * inputpos.x * dims.x) +
              cam->getUp()* (float) (0.5 * inputpos.y *dims.y);
  return worldVec;
}
void translateCamera{vector2 inputposition, vector<int>ids){
  if(ids.haveChanged()){
    flaot dist = getDistToMostPerpendicularPlane(camera.position, camera.orientation);
  }
  vector3 sceneTranslation = unprojectVector(inputposition);
  camera->move(vector3(-sceneTranslation.x *dist, -sceneTranslation.y *dist, -
      sceneTranslation.z *dist));

}
```

Listing 3.1: Direct Camera Panning

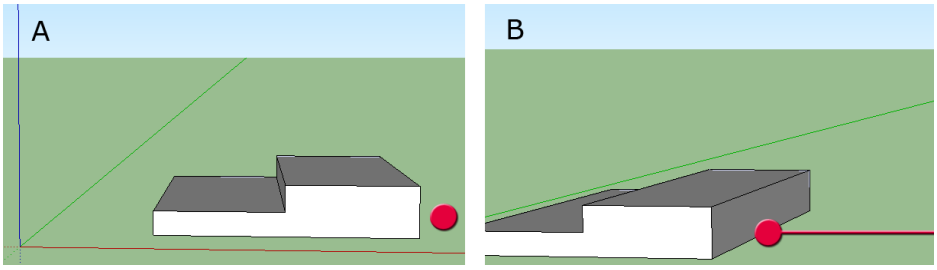

Figure 3.4: Local x-axis camera translation to the right. While in fig. A the coordinate systems point of origin is still visible, it is not in fig. B. The input point (red) hereby stays in contact with the point of the scene first "touched".

Zooming

Camera zoom is often used together with camera panning. If the operator uses for example a common scroll mouse the motion of the scroll wheel can be used for moving along the camera's local z-axis. At the same time the mouse's motion may still be applied to pan the camera. Sometimes the input's initial position is projected into the scene and used as a focussing point allowing the user to translate the camera in the direction of the point the camera is zooming to, so that the focussing point moves towards or away from the centre of the screen. Figure 3.5 shows the results of a camera zoom in Blender, which translates the camera along its local z-axis on each mouse wheel motion event.

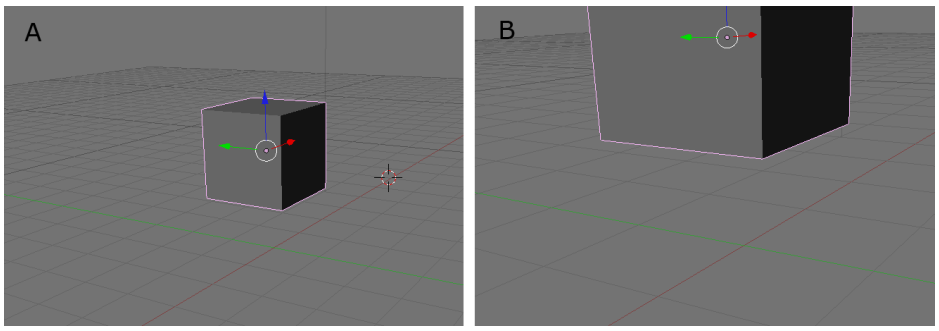

Figure 3.5: Camera zoom. In (B) the object appears so have increased in size, but the grid reveals that the camera has been moved to the front.

Thus the cameras initial position p_0 transits into the new position p_1 with:

$$p_1 = p_0 + \partial * \begin{pmatrix} 0 \\ 0 \\ x_i * f \end{pmatrix}$$

with the relative motion $x \in (-1, 1)$ multiplied by a constant factor f on each input event $\#i$ and ∂ representing the camera's orientation as a quaternion $Q \in \mathbb{H}$.

Listing 3.2 provides a pseudo code reference - the method is hereby called each time the mouse sends a wheel motion event. Means to create a direct impression of zooming similar to the camera panning behaviour will be explained in chapter 5 as this behaviour depends on the input technique chosen for the task.

```
void zoomCamera{int relmotion){
  cam->moveRelative(Vector3(0,0,relmotion*factor);
}
```

Listing 3.2: Camera Zoom with relative motion

Orbiting

Orbiting (shown in fig 3.6) allows a user to view an object from any direction without changing the cameras roll angle. The input device's motion is mapped to a rotation around a plane (usually xz-plane) while constant distance to the centre of the rotation is kept. A common model for implementation is to translate the camera to the gravity or centre point, then apply the rotation to its local axis and finally move the camera back the same distance it was translated forward before. Listing 3.3 shows an implementation of this orbiting behaviour.

A point $P = (x_p, y_p, z_p) \in \mathbb{R}^3$on the sphere defined by the radius $R \in [0, \infty]$ and the centre point $C = (x_c, y_c, z_c) \in \mathbb{R}^3$ is given by:

$$x_p = x_c + R * \sin(\phi) * \cos(\theta)$$

$$y_p = y_c + R * \sin(\phi) * \sin(\theta)$$

$$z_p = z_c + R * \cos(\phi)$$

with ϕ and θ being the inclination and elevation angles with $(0 \leq \theta \leq \pi)$ and $(0 \leq \phi \leq 2\pi)$. Thus the cameras initial view point $V(p_0, d_0, u_0)$ transits into the new view point $V(p_1, d_1, u_1)$ with

$$p_1 = \begin{pmatrix} x_c + D * \sin(\phi + f_x) * \cos(\theta + f_y) \\ y_c + D * \sin(\phi + f_x) * \sin(\theta + f_y) \\ z_c + D * \cos(\phi + f_x) \end{pmatrix}$$

$$d_1 = (p_1 - C)$$

$$u_1 = d_1 \times (d_1 \times u_0)$$

with $f \in \mathbb{R}^2$ indicating the angular change by the input device motion in x and y. $D \in \mathbb{R}$ being the distance between C and p_0: $\|(p_0 - C)\|$ and d_i representing the camera's direction vector and u_i the camera's up vector.

```
1
void orbitCamera{vector2 motion, vector3 centerpoint){
  float factor = 1;
  float dist = length(centerpoint-cam->getPosition());
  cam->setPosition(centerpoint);
6 cam->yaw(Degree(factor*motion.x));
  cam->pitch(Degree(factor*motion.y));
  cam->moveRelative(Vector3(0,0,dist));
}
```

Listing 3.3: Camera Orbit

Camera Roll

Rotation around the cameras local z-axis is called rolling. However, there is no independent camera roll mode in either Blender nor SketchUp. But it is possible to roll the camera node and take over its view in Blender since it has a camera node separated from the view port for animation functionality. Instead of a camera roll mode SketchUp has two different orbit modes. If the user presses STRG while orbiting, he/she will

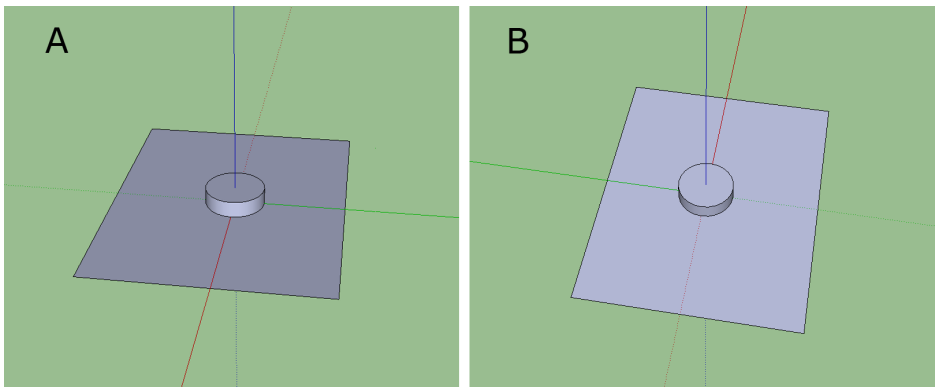

Figure 3.6: Camera Orbit. (B) shows the object from the opposite side and also from a higher altitude.

change yaw, pitch and roll of the view port making it difficult to independently control the DoFs of this technique. If STRG is not pressed, the scenes y-axis will always have a fixed orientation pointing either up- or downwards. The unrestricted orbit rotation is available in Blender, too.
Figure 3.7 shows an example of a camera roll and listing 3.4 provides a short reference implementation using two input points. Again we describe the change of the cameras initial view point $V(p_0, d_0, u_0)$ to the new view point $V(p_1, d_1, u_1)$ on a motion event of the input device:

$$p_1 = p_0$$

$$d_1 = d_0$$

$$u_1 = R(\alpha, d_1) * u_0$$

with $R(\alpha, \xi)$ being the rotation matrix defined by the rotation angle $\alpha \in \mathbb{R}$ and the axis of rotation $\xi \in \mathbb{R}^3$.

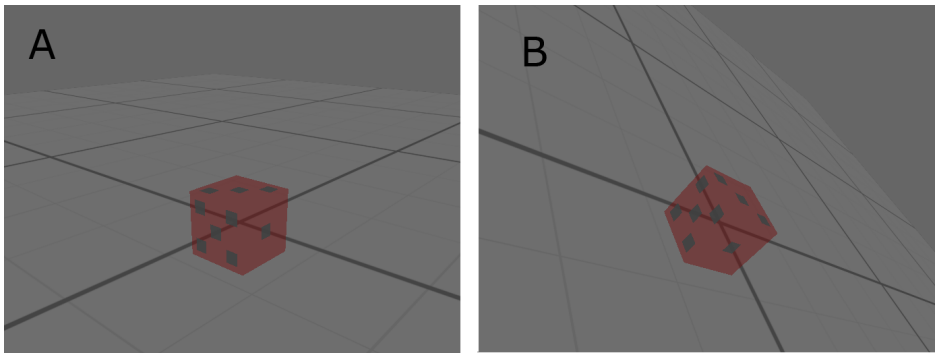

Figure 3.7: Camera Roll. From (A) to (B) the camera has been rotated around its local z-axis by about 45 degrees..

```
void rollCamera{vector2 oldInputA, vector2 newInputA, vector2 newInputB, vector2
    oldInputB){

  vector2 newDir, oldDir;
  oldDir = oldInputA - oldInputB;
  newDir = newInputA - newInputB;
  oldDir.normalise();
  newDir.normalise();

  float angle = atan2(oldDir.y, oldDir.x) - atan2(newDir.y, newDir.x);
  cam->roll(Radian(angle));
}
```

Listing 3.4: Camera Roll

Object Manipulation Tasks

Selection

Differentiating between objects in the scene usually takes place by selecting a certain object to restrict following manipulations to this object only. Without a selection technique the application might struggle to identify the object(s) to which the next manipulation should be applied. For the same reason the application might even be unable to provide access to the currently possible manipulation tasks for a certain object, since these can differ from object to object. From a users point of view a selection technique is not needed to perform the task, but rather needed for the application to understand the users intention. Thus selection is a technique which does not affect the objects properties directly, but rather serves as a special form of a mode switch. Single object selection usually is carried out by a single, double or triple click, to account for different levels of selection: e.g, the current face, the face's outline or the whole object this face is attached to.
Of course, a selection technique might not be required, if direct interaction is enforced for every manipulation - an object would then be selected the moment it is touched and a gesture is recognised. Depending on the application a selection technique might not be needed. However, to reduce fat-finger related side effects by e.g. using a variable input off-set we need to provide a selection technique because the intended target cannot be determined by ray casting the input points.

Translation

Object translation is often needed when the users has several objects in his scene and wants to arrange them in a certain way. Without other objects or scene elements the position of a single object is of less interest as long as the user can view it from any direction to manipulate its other properties. In many applications object translation is performed by moving the cursor inside the objects extents and then dragging it to the desired location. The resulting movement of the object, though, differs from application to application. SketchUp maps the motion of the input device to translation on a plane of the world coordinate system. There is also a possibility to snap to a certain axis - even, if its not part of the active plane.
Blender applies the translation restricted parallel to the view port, making it difficult to align the object axes with the world axes. These modes however, are only the default translations available when pointing on an object and dragging it, they are accompanied by many specialised sub-modes, accessible through menus and mode switches. Figure 3.8 shows a comparison of the object translations in Blender and SketchUp, both performed by mouse dragging. Listing 3.5 gives a basic implementation for direct object translation on a plane in 3D with one input point.

Let p_i be the objects position at time #i with $p \in \mathbb{R}^3$. The old position p_0 before an input event is transformed to p_1 after the input event with:

$$p_0 = p_0 + \Delta t$$

Hereby $\Delta t \in \mathbb{R}^3$ is the motion vector implied by the two input events converted to world space. This conversion can be done in several ways (see fig. 3.8) and it is up to the designer to chose the behaviour which fits the intended purpose best.

```
void translateObject{vector2 inputmotion){
  float distanceToObject = abs(length((cam->getPosition() - object->getPosition())))
  vector3 translation = convertScreentoScene(planeOfReference, distanceToObject,
                              inputmotion, bool localtranslation);
  object->translate(translation);

}
```

Listing 3.5: Object Translation

Scaling

Changing size of an object on the screen can be achieved by either camera zooming or by scaling the objects extends in all dimensions by the same amount. However,

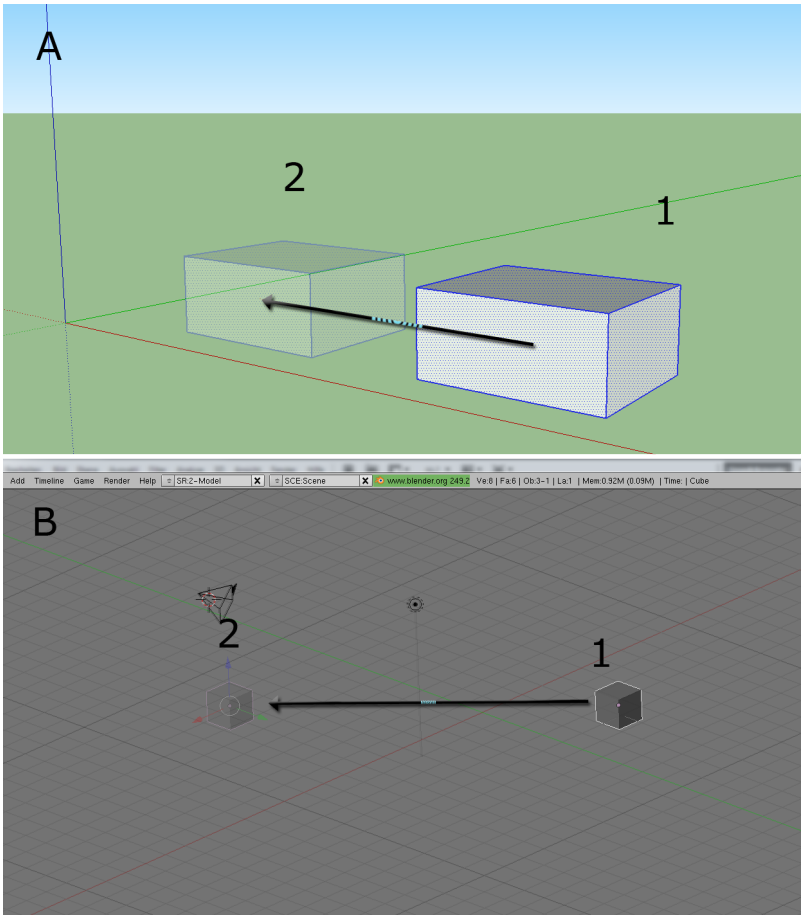

Figure 3.8: Object translation in SketchUp (A) and Blender (B). While in (A) the object is translated snapped to the global x-axis (red) the object in Blender translates parallel to the the camera's xy-plane

scaling only the length or width of an object allows to change basic geometry objects like a cube and a sphere into more sophisticated shapes.
Again accessing these tasks differ from application to application. On the one hand we can use a mouse gesture which enables global scaling for the currently selected object without any widget, on the other hand we will then need e.g. a context menu to access other scaling modes - as done in Blender. In contrast to this SketchUp offers a widget providing several anchor points to restrict scaling of the object to various dimensions. Figure 3.9 shows the SketchUp scaling widget behaviour in three steps. Listing 3.6 provides a basic pseudo implementation for global object scaling with a mouse.

Let s_i be the objects relative scale at time #i with $s \in \mathbb{R}^3$. The old scale s_0 before an input event is transformed to s_1 after the input event with:

$$s_0 = s_0 + \Delta m$$

Hereby $\Delta m \in \mathbb{R}^3$ is either a constant or a factor calculated by the motion vector implied by the two input events to scale the object in relation to the input motion creating an impression of direct manipulation. However, when only one input point is available, restricting scale to a certain object axis or plane requires a technique to convey the users intention to the application.

```
void globalScaleObject{vector2 relInputMotion){
  float factor = relInputMotion*5
  vector2 newScale = object->getScale();
  newScale = vector3(curScale.x+factor,curScale.y+factor,curScale.z+factor);
  object->setScale(newScale);
}
```

Listing 3.6: Object Scale

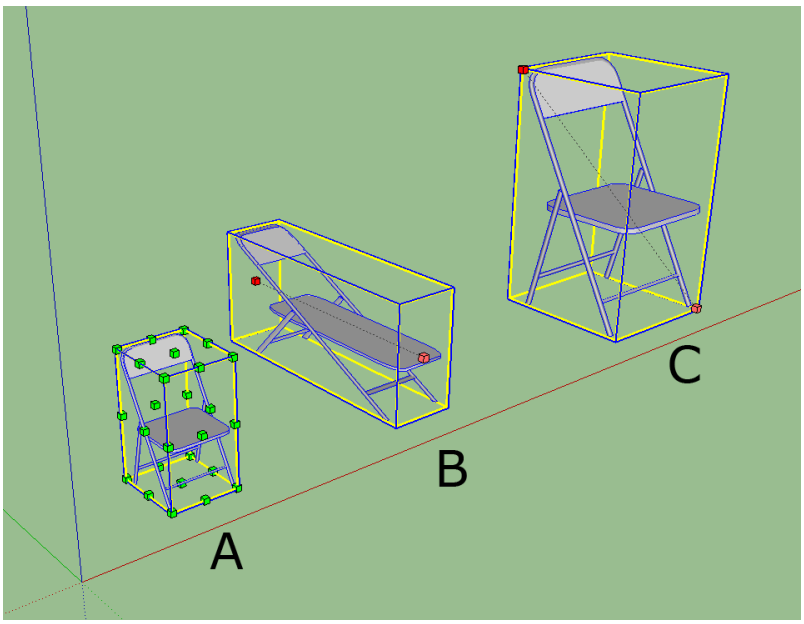

Figure 3.9: Object Scaling in SketchUp: The different anchor points of the widget restrict the scaling to certain dimensions. The initial object (A) is scaled in width (B) and globally (C).

Rotation

Inexperienced users often struggle to carry out their intended object rotations at first. This might be due to the more complex nature of the task or difficulties comprehending the results provided by the various rotation modes of the application. Object centre rotation for example is a three DoFs task, therefore it is difficult to manipulate each axis independently with only one 2D input point - similar to camera orbit without a fixed axis. Thus providing object centre rotation with one input point would require either a rotation widget for independent axis manipulation or a technique like the ArcBall [23]. Although the ArcBall provides access to all three rotational DoFs it sometimes is difficult to anticipate where to start and end the dragging motion to perform certain rotations [23]. There is neither a object centre rotation mode in SketchUp nor a default mouse drag technique mapped to any sort of rotation, instead a single rotation tool similar to a "Winkelskala" (see fig. 3.10B) provides axis restricted rotation depending on the point its attached to. This requires several iterative manipulations to the object, if the rotation involves several DoFs. Blenders default mouse-drag rotation is also not mapped to an ArcBall like behaviour, but to rotation around an axis parallel to the cameras look direction.

Let o_i represent the objects orientation at time #i with $o \in \mathbb{R}^3$. The components x, y, z of o hereby represent the roll (rotation around x-axis), pitch (y-axis) and yaw angles (z-axis) of the object. The old orientation o_0 before an input event is transformed to o_1 after the input event with:

$$o_0 = o_0 + \Delta a$$

Hereby $\Delta a \in \mathbb{R}^3$ can be a constant factor in one component to rotate around a single axis only, or the resultant angle changes of a more sophisticated rotation technique as the ArcBall or a two finger rotate technique rotating a 3D object around a single axis in regard to the angular change between the two input points.
A pseudo implementation is provided in listing 3.7. In this example the object will rotate around its y- or x-axis depending on the provided rotation axis.

```
void rotateObject{float relmotion, axis axisToUse){
  if(axisToUse == AXISX){
    object->rotate(AXISX, relmotion);
  }else{
    object->rotate(AXISY, relmotion);
  }
}
```

Listing 3.7: Object Rotation

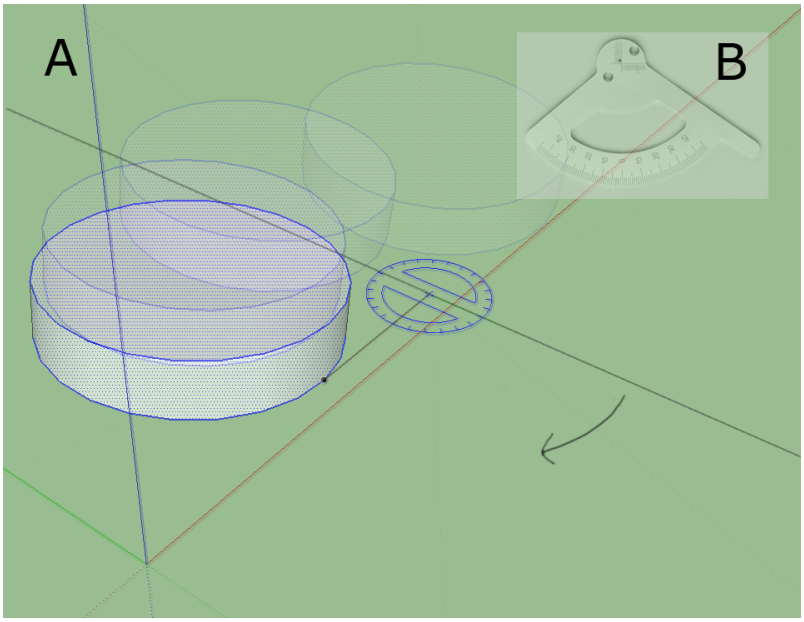

Figure 3.10: Object Rotation in SketchUp: Attaching the rotation tool outside the objects boundaries on the xz-plane results in a rotation around the axis in that point perpendicular to the xz-plane.

3.4 Initial Prototype Design

All listed task should be in some form available in our interfaces. Formulating an initial design requires a basic idea of how to map all these tasks to input techniques. Moreover, we need to decide, if and how to separate task DoFs within the same techniques or by supplying several interaction techniques for the same tasks, each referring to different task DoFs.

A straight forward attempt to map the basic camera navigation tasks to touch input gestures would be to assign different amount of touches for each interaction task, hence creating a new interaction technique for each task, although it may require less DoFs than the input points offer. However, due to hand or finger constraints such as the difficulty of independently moving the pinky and ring finger, usage of gestures requiring a lot of touches could be problematic for some users. On the other hand separating techniques only by touch count may reduce unexpected behaviour or even reduce input lag, which could be introduced by a gesture recognition process requiring extra time to resolve ambiguities, when the same touch count is used for multiple techniques. In general, available input points are constrained by the amount of fingers

humans have, thus capping the amount of techniques separable by touch count to ten.

Presuming a fixed number of techniques which can be made available through an interface by this method, we could reduce the maximum number of touches required for a technique by combining interaction tasks which work well together into a single technique.
This way we can more effectively make use of the additional DoF the input points offer utilising a major advantage of multi-touch interaction - simultaneous manipulation of different properties. Moreover, we would then be able to increase the amount of techniques we can support before needing to use mode switches.

With the above method panning, orbiting, zooming and camera roll can be mapped to one, two, three and four touch movement by reducing multiple points to a single one with e.g the point of gravity in a triangle specified by three touches. However, we can combine these tasks into two interaction techniques requiring one and two touch point input. This way camera zoom and roll would be assigned to the distance and angle change between two moving touch points. Further camera panning can be assigned to the midpoint translation of both touch points, which leaves one touch movement for camera orbit since the four DoFs of the two touches are exhausted by the three DoFs translations and the one DoF rotational task.
This mapping however, is not unique - swapping camera orbit and panning won't change the functionality of the interface, but perhaps alter its usability because of different user expectations. To create a single input technique for all these tasks three input points are needed as they provide the necessary six input DoF for the six output DoF of the tasks. It seems reasonable that users may prefer the technique which involves fewer fingers, if two alternative techniques for the same tasks are available. Therefore we prefer the use of multiple techniques over a single technique requiring more input points.

While it is straight forward to find an initial mapping for the camera navigation tasks, combining object manipulation tasks, eventually with separated DoF, is more complicated.

Since one touch movement is often used for object translation in 2D applications, we take this as a starting point. We can map one touch movement to global object translation along two variable axes, making it dependant on the cameras view point, resulting in either global xy-, yz- or zx-plane translation. To be able to translate the

object along a fixed axis with one touch we can for example use a snap to grid feature separating task DoF within the same technique. This view dependant technique may not be as usable as the Z-Technique [29], because the user cannot translate the object into every direction without needing to perform another technique. However, this combination of techniques provides three DoFs translation for an object and therefore could prove a solid starting point as we focus on the overall usability of the combination of techniques. Thus we have to balance between the number of techniques and the amount of possible mappings, likely preferring the solution involving fewer touches to establish a minimal base line. This might allow us to better estimate effects of additions and adaptions of the initial design (see chapter 5).

The distance change between two moving fingers can support either global or axis restricted object scaling. Since both techniques are separately available in many CAD applications, providing two techniques for these two tasks seems a more comprehensible approach than only supporting a single two touch scaling technique with separated task DoFs. Therefore axis restricted scaling should be performed by two touches while three touches should allow the user to scale globally. Two of the three touches, if not changing relative position to each other, can be reduced to a single point which reduces this three touch technique effectively to a two point scaling technique. The third touch hereby can be seen as a necessity to distinguish between the two techniques, but the additional DoFs the point provides remain unused.
Still the distance change and the angle between the two touches used for direct scaling offer only two DoFs. While offering sufficient DoF for global scaling, we either need a third scaling technique to manipulate the remaining object axis or make the two touch scaling technique camera view dependant similar to the translation technique.

With a one touch 3DoF translation, a two touch 3DoF separated axis scale which are both view point dependant and a three touch global scale technique, we require at least four touch points to provide object centre rotation with an ArcBall controller, while still separating all interaction tasks from each other by touch count. This leaves five input points to the orbit technique and six finger movement to the camera rotate-zoom-translate technique.
This mapping is scene-independent since distinguishing between the techniques is not dependant on information provided by the scene, but merely on touch count. Though it is possible to reduce touch count further by taking meta data from the scene into consideration. But to reduce fat-finger related issues and to provide users the option to chose between touching the object and performing the intended technique anywhere

on the screen, we chose a mapping not relying on any scene information.

Table 3.1 summarises our initial design concept. Two view point dependant object manipulation techniques - translate and scale are combined with an ArcBall controller, global object scale, camera rotate-translate-zoom-roll and orbit. For each of our three interface prototypes this initial design is subjected to change in order to explore specific design issues in chapter 5.

If these adaptions include the usage of multiple techniques with the same touch count, our application would need a mechanism to recognise and distinguish between these techniques. For example by classifying the type of input motion on the screen into different categories to gain additional information for the separation of tasks and task DoF. Quality of this gesture recognition system may directly influence usability of the underlying interfaces. Bad recognition quality is likely to alter usability in contrast to a stable and robust recognition system used with the same interface.
Detecting an interaction technique during the first frames of the interaction and then locking this mode until the touches are removed may provide stable recognition assuming the first frames are sufficient time to make a reliable guess for the intended interaction technique. On the other hand locking the detected technique exclusively until the touches are removed makes it impossible to add or remove touches during interaction to change between techniques. Supporting this however, could reduce fatigue and may allow more fluent interactions. We will explain our gesture recognition system alongside our design refinement in chapter 5 as these are co-dependant factors.

Task	#Touches	input DoF	task DoF	effective DoF
Camera Pan-Zoom-Roll (PZR)	6 (2x3)	12	4	4
Camera Orbit	5	10	2	2
Object Translation	1	2	2	3 (PZR)
Object Axis Scale	2	4	2	3 (PZR)
Object Global Scale	3	6	1	1
Object Centre Rotation	4	8	3	3

Table 3.1: Initial mapping of interaction tasks to interaction techniques.
Effective DoF: The number of DoFs a technique is able to manipulate by combination with other techniques.

3.5 Incremental Design Process

Ultimately achieving some degree of usability with a multi-touch interface for a certain 3D application would require the designer to consider the trade-off between precision and speed through the hardware, the relation between directness and separability and the various possibilities to map available input DoF to output DoF. Yet the final product should be easy to use while at the same time suffice user demands for accuracy and functionality. However, if at all or to which extend multi-touch technology can improve usability of new interfaces has to be carefully iterated and reevaluated from application to application.

We have explained multi-touch related design properties and discussed possible implications on the design of 3D interaction techniques. Furthermore we chose a set of 3D interaction tasks which we see fit to provide basic 3D navigation and object manipulations orienting on the core functionality of 3D CAD and design applications. This allows us to line out the remaining design process.

Step One)
Find alternative and additional interaction techniques for the basic 3D interaction tasks refining our initial design (see table 3.1) to be able to make comparisons between interfaces.
A multi-touch interaction technique hereby can be distinguished by a combination of the following parameters:

Amount of touches Touch count and, if supported by the hardware, further tracking of fingers and hands to identify which touch belongs to which finger and / or user.

Type of Movement Developing a model describing the touch movement may assist in gesture detection and increase interaction richness.

Recognised Shape / Fiducials If the hardware supports detection of shapes and arbitrary objects or touch point-size, these attributes can help to further distinguish between interaction techniques or increase interaction richness.

Step Two)
Find new combinations of interaction techniques which do not cancel each other out during gesture recognition yet reach the desired functionality requirements defined in step one.

Step Three)
Evaluate the usability of the resulting interfaces.
Although each interaction technique chosen for an interface may be usable to some degree its combination does not necessarily need to be of the same quality. Evaluation and reiterating over the previous design making adaptions due to evaluation results is hereby common design practice.

4 Evaluation of User Expectations

In order to identify input techniques, which are to some degree usable for 3D interactions, we can either try to extend successful techniques from 2D context like the RST technique to 3D or create new techniques, which perform particularly well in the 3D context. Most likely there exist several input techniques for a certain task, differing in their degree of usability. However, usability of a single technique might be altered by combining it with other techniques to create interfaces satisfying the functionally requirements of its application. To gain a first impression on this relation between usability and combination of techniques we performed an online user survey. By asking participants to choose their preferred input gesture from a predefined set for a single and combination of 3D interaction tasks we expected to gain hints on how to tackle this combinatorial problem and in addition find alternative candidates for input techniques which the user might prefer.
We expected results of the survey to be able to assist in refining certain design decisions for our prototype application. We discovered that in general participants are able to cope with the complexity of the theme, although they did not have any possibility of trying out the suggested gestures. But especially the second half of the survey did not yield distinct results promoting a presented technique above others. On the other hand participants often relied on choosing techniques which they already knew from 2D interaction like the one finger movement for object translation. The question whether these results occurred because of our small sample size, the complexity of the theme, missing experience from the participants or faulty survey design can only be guessed. Nevertheless we received some interesting suggestions for interaction-techniques from participants thinking "out-of-the-box" which will be presented in the results section of this chapter.

4.1 Setup and Methods

The survey was created with LimeSurvey[1] in English and German and is divided into six blocks. It can be found at `http://www.suitamoi.de/survey/`.

[1] `http://www.limesurvey.org/`

Initially, participants watched an introductory video explaining the motivation and context of the survey, while also giving some background information about multi-touch to the participants. Then the participants had to answer general questions regarding gender, age, experience with computers, touch screens and their grade of education. After a brief training section explaining the upcoming tasks the participants had to chose a gesture they felt most appropriate for object manipulation tasks like scale, rotate and translate in the context of interacting with 3D objects in a 3D scene.

We somewhat expected that participants might have difficulties answering the questions. Because of the missing opportunity to try out the presented options and the complexity of the theme requiring the participants to rotate, scale and translate - sometimes simultaneously - in their minds. To assist the participants each question featured a video clip demonstrating the interaction task a gesture should be chosen for. Nevertheless understanding and interpreting the presented gestures was still completely dependant on imagination as they were presented in form of a textual description. We therefore chose to create an initial training block which should on the one hand introduce the participants to the theme while on the other hand determine the participants ability for three dimensional visualisation and their observation skills. A large number of participants not being able to differentiate between simple three-dimensional manipulations by watching at a set of before/after pictures might indicate that this theme in general is too complex to be discussed in an online survey or that our surveys methods for presenting questions and answers are not sufficient enough for the participants to be able to make well-founded decisions.

The layout of a question from the survey's main blocks can be reviewed in figure 4.1. In this case a question regarding object translation in the xy-plane and its possible answers are shown. Each block of the trial focussed on one interaction task, for example translation. Each question in a block referred to different DoF(s) of the manipulation task which should be manipulated. The translation block therefore has a question regarding xy-plane translation of an object, a question for z-axis translation and for a combination of both tasks. All three survey main blocks are designed analogous to this example. In the last (optional) block of the survey the participants chose combinations of all previously presented gestures to perform object manipulation tasks such as translate-rotate, scale-translate and rotate-scale in combination. After each block the participants had the option to provide feedback describing his/her ideas by drawing a picture or writing a text.

Figure 4.1: The layout of the questions in the three main survey blocks. This example shows the first question of the translate block regarding the global xy-translation of an object.

4.2 Results

Block 1 - General Results

107 people have taken part in the survey of which 36 (27 male, 9 female) completed the whole survey - all other submissions were removed from the data. The 36 participants who finished the trial had an average age of 28,67 years (SD:14.82). 41,67% of the participants stated to have very good computer skills and another 38.89% estimated good skills in comparison to the average level (16.67%). Only one participant stated to have worse than average experience with computers in general. 63,89% of the participants use a computer for business and private matters, while 30.56% use it exclusively at home and 5.56% only at work. 66% of the participants own a device that can be controlled with touches, of those two third, only one stated to see room for improvements. All other were quite pleased with the controls *[23.07% - "It's very good"; 38.46% - "It's good"; 26.92% - It's ok"]*. Of the 12 participants who did not own a touch device, three were planing to get one in the near future with the touch support being a main reason for their decision. Because the survey was proposed to many different online communities to gain as much attention as possible education level of the participants vary greatly. Though most of the participants were IT students or professionals there are various submissions from people who might not have any experience with the theme of this survey.
Before getting to the training section of the trial participants also had to pass a remark on several statements regarding some multi-touch design properties such as precision, physical strain and motivation to use. About 69% of the participants stated to sometimes wish for another input device aside the mouse/keyboard combination *[27.78% - "I agree"; 41.67% - "I mostly agree"; 19.44% - "indecisive"; 2.78% - "I mostly disagree"; 8.33% - "I disagree"]*, but at the same time they were not sure, if "there is no better input device than the mouse", as two third being indecisive and ~40% disagreeing with this statement *[5.56% - "I agree"; 11.11% - "I mostly agree"; 33.33% - "indecisive"; 30.56% - "I mostly disagree"; 13.89% - "I disagree";5.56% - "I don't know"]*. Moreover, ~64% agreed that touch input can be imprecise from time to time *[13.89% - "I agree"; 50% - "I mostly agree"; 11.11% - "indecisive"; 16.67% - "I mostly disagree"; 8.33% - "I disagree"]*. All participants somewhat agreed that "when working with a computer I want things getting done fast and accurate" *[69.44% - "I agree"; 30.56% - "I mostly agree"]*. About 87% of the participants rejected the statement that touch-screens are no "fun" *[5.56% - "I agree"; 2.78% - "I mostly agree"; 2.78% - "indecisive"; 25% - "I mostly disagree"; 52.78% - "I disagree"; 11.11% - "I don't know"]*. Answers to the question, if "working 20 minutes with a touch screen can be exhausting" were very wide spread, though. In figure 4.2 a pie-chart for the given

answers can be reviewed. Many participants also stated to be ready "to invest some hours to learn using a new program, if it's worth it" *[38.89% - "I agree"; 52.78% - "I mostly agree"; 5.56% - "indecisive"; 2.78% - "I disagree"]*. ~72% of the participants "could imagine using touch screens more in their free time in the future" *[27.78% - "I agree"; 44.44% - "I mostly agree"; 22.22% - "indecisive"; 2.78% - "I mostly disagree"; 2.78% - "I don't know"]* and ~43% could think of an application for touch-screens in their field of expertise" *[16.67% - "I agree"; 11.11% - "I mostly agree"; 19.44% - "indecisive"; 13.89% - "I mostly disagree"; 30.56% - "I disagree"; 8.33% - "I don't know"]*.

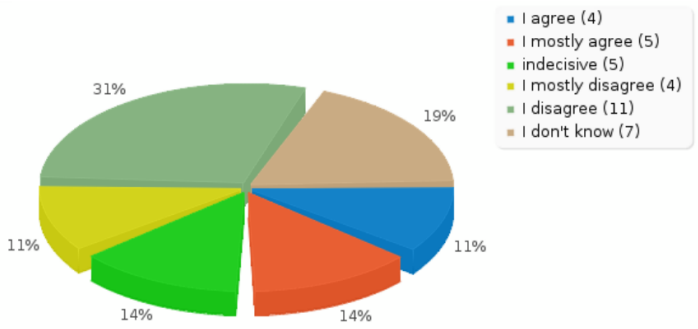

Figure 4.2: Pie chart representing the answers given in the survey to the statement: "Working 20 minutes with a touch screen can be exhausting."

Block 2 - Training Block

The training block aimed to deduce, if participants are able to observe and explain differences in a 3D scene by viewing at a set of before/after pictures. At first the participants were explained the differences between a two-dimensional translation in a two-dimensional scene and a three-dimensional scene and translation.
The first task showed an object being translated to the back of a scene parallel to the global z-axis with one touch point. All participants recognised the backwards translation, but only 56% were sure that the object has not been moved parallel to the y-axis *[22.22% - "Yes"; 55.56% - "No"; 22.22% - "uncertain"]*.
In the second task participants should compare object translation and camera panning which both were performed by single touch movement. Then three questions regarding these pictures were to be answered. ~47% of the participants wished for a "clearly distinguishable gesture for camera or object translation" yet another ~42% rejected this idea, additionally 61.11% of all participants identified correctly that simultaneous translation of object and camera is theoretically possible. Furthermore 15 (41.67%) participants found this a desirable function. The last question required the participants to identify the reason why one touch movement could be used for two interaction tasks in this example. 80.56% of the participants agreed that the point of contact on

Rank	Application	Accumulated Score
1.	Games	222
2.	Presentations	214
3.	Media Libary	210
4.	Digital Photo Enhancement	180
5.	CAD	175
6.	Webdesign	109
7.	office tasks	76
8.	software development	74

Table 4.1: Results of the Application Ranking Task

the screen is responsible for this behaviour - as the pictures of the task once showed a touch in contact with an object and once with the world space.
The last task required the participants to create a ranking of applications they think could benefit the most from multi-touch input. The first entry in the list gave 8 points down to the eighth entry giving one point. The final ranking is a sum of all individual rankings (see table 4.1).

Block 3 - Object Translation

Participants often were able to choose more than one answer depending on the number of possible answers, thus the sum of all votes may exceed the number of participants. In the first two tasks of this block the participants had to choose a gesture for translating the object in the global xy-plane and along the z-axis. After that they had to choose a combination of gestures to carry out both translations together - for example translation to the top-left and backwards at the same time or with as little effort as possible switching between them. For the xy-plane translation of an object 83.33% of the participants chose "touching the object with one finger and moving the finger to translate the object" as their preferred gesture. Result for the z-axis condition are more widespread, yet three of the proposed six gestures each received ~33% of all votes. A description of the "Z-Technique" [29] as well as a point of view dependent translation technique received 12 votes. Only the possible usage of a certain amount of fingers to translate the object backwards received one additional vote.
For combining both translations though, participants were in favour of one finger movement for the xy-plane (58.33%) and two finger movement for the z-axis translation (36.11%). The point of view dependant technique which received a fair amount of votes in the z-axis condition was only voted by 16.67% of the participants.

Block 4 - Object Scaling

In the first task of the fourth block two third of the participants chose to push or pull with two fingers touching the sides of an object to scale it. Scaling in this task was restricted to either width, height or length of the object. Global (all dimensions at once) and free scale (custom shaping) were available in the combination task though. However, votes for the combination task are distributed over all possible answers, moreover, about 25% of the participants chose "no answer" for any of the tasks. In the scaling block the task of destroying or deleting an object was introduced, too. A video displaying the tearing of an object with help of a physics engine in 2D was shown beforehand. Participants favoured pulling the object above a recycle bin (~61%) or drawing an "X" on it (~47%) over the tearing solution (~11%).

Block 5 - Object Rotation

The rotation block required the participants to chose a gesture for rotation around an objects centre and around any side, face or edge. For the centre rotation about 61% of the participants would use a one finger ArcBall rotation similar to the one provided in the example video. For the face/edge/point rotations two possible answers suggested the user to specify a point, face or side with a designated number of fingers touching the object to restrict the rotation. Participants then preferred to use an additional finger (47.22%) to rotate the object instead off moving the fingers which specified the axis, face or point of rotation (19.44%). This indirect rotation technique and the ArcBall were also voted most in the combination task. Also participants had to choose the, in their opinion, worst gesture. However, each technique received between six to eight votes except the one finger centre rotation.

Block 6 - Combination of Interaction Tasks

About 60% of the participants worked on the last block. Since it was yet more complex than the previous blocks we wanted to make sure only motivated and still concentrated participants took on these questions by making this block optional.
Instead of choosing a fitting gesture to be able to access different DoFs of the translation,rotation or scaling tasks, e.g. axis restricted scale and global scale - participants should point out possible gestures or gesture combinations to carry out a combination of basic tasks such as scaling+translation and rotation+translation in this block.
At first they were asked which tasks would be qualified to be combined with each

other. 50% of the participants voted translation+rotation and translation+scaling interesting for simultaneous use. Scaling+rotation was voted by 41.67% of the participants. Finally participants who chose to answer the following three questions regarding the combination of tasks favoured one finger movement (41.67%) for translation. As in the scaling block votes are distributed over all answers for the scaling part of the scaling+translation technique.For the rotation+translation as well as the rotation+scale combinations no clear preference can be deduced from the results,either.
Survey contents and all results can be reviewed on the materials DVD.

4.3 Discussion

Survey Block 1

A large proportion of the participants acknowledged unconsciously that "every input device serves a certain purpose better and performs worse in another situation"[9] as they sometimes wished for alternative input devices than mouse and keyboard. All participants agreed that "when working with a computer I want things getting done fast and accurate", pointing out the utmost importance of these two properties in user interface design. On the other hand, half of the participants disagreed to the statement that working with touch screens can be exhausting. Reason for this may be due to the relatively small display size of smart phones as they are the most widespread devices supporting touch input nowadays. Participants experience with touch screens therefore most likely relies on their experience made when using smart-phones. In contrast to imprecision, which is a problem independent to the screen size, physical exhaustion may not be of such influence on the usability in small scale devices as they can be aligned differently as soon as the user feels uncomfortable with the current position. Also participants were dedicated to learn usage of an application, if "it's worth it" which could indicate that ease-of-use, although a property which greatly influences usability, could possibly be outperformed by the applications purpose sometimes. Since participants could think of various applications for touch technology in their field of expertise, we confidently state that possible applications for touch-screens are far from being exhausted.
Regarding the application ranking made in the training section CAD and 3D design applications were ranked fifth place. But there is a considerable gap between rank five and six dividing the applications into two groups for potential use with and without touch input from a users point of view.

Survey Block 2

Results of the translation block suggest that participants are well aware of the difference between choosing a single technique for a single task and choosing a combination of techniques for different DoF of the task in the object translation block. As a part of the combination task, participants were also asked to chose the gesture which they thought would perform worst for the proposed task. In this context the description of a translation technique using a physics engine was voted most often, followed by the point of view dependant technique. Although the point of view dependant 2DoF translation technique has received some votes in the second task of this block (z-axis translation) it is now considered a "bad" technique. Reasons for this discrepancy could be the abbreviation of answers in the last question which could have lead to misinterpretation of the answers. This is supported by one participant submitting a drawing of a point of view dependant input technique as a possible input technique (fig. 4.3).

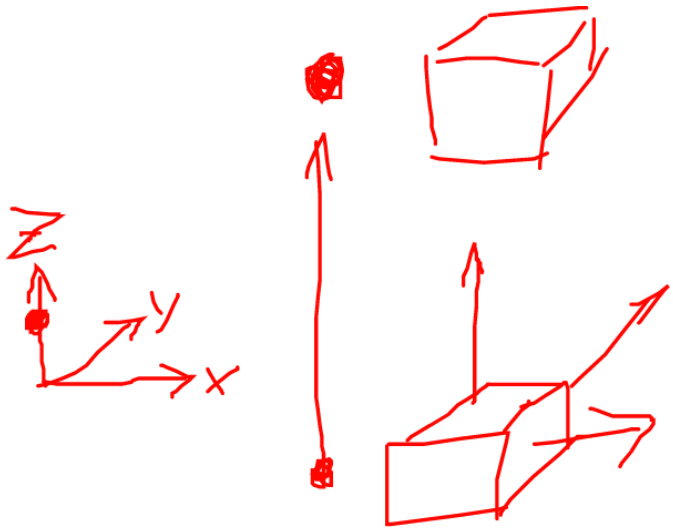

Figure 4.3: Custom submission of, as we interpret, a camera view point dependant translation technique. It is also possible that usage of an axis restricting widget was illustrated by the participant.

By formulating possible answers as short as possible we intended to reduce the amount users had to read to keep their motivation and concentration at a high level. Unfortunately this increases the possibility of interpreting the answers in several ways. This could have been a general problem during the whole survey as results of a task are usually more widespread, if the answers contained abbreviated versions of the

answers presented in the previous tasks.
Providing possible gestures in an unambiguous and more illustrating manner might lead to significantly clearer results.
Three DoF translation is suggested by a participant by touching both, the object and the axis the translation should be applied to and then moving both touches. This however, would require a visual axis-widget attached to the object to switch modes - especially when manipulating more complex shapes. Another participant thought of the possibility to tilt the screens from vertical to horizontal to switch between xy- and xz-translation with one finger. The five finger pinch gesture Apple is using to close an application e.g. on the iPad[2] was suggested as a possible candidate for back and front translation, too.

Survey Block 3-5

Regarding the results of the scale block either combination of five scale tasks (length, height, width, global, and free form) might have been too complex for the participants or the available answers were to similar or imprecise formulated. Also it might be possible that participants did not understand why they had to chose more than one scaling technique, hence they tend to choose a two finger technique for each scaling task, if the answer was available. Introducing three interaction tasks in the first video of this block instead of one video for each interaction task (width, height, length) could have affected participant answers. The video suggests that these tasks should be performed by the same technique possibly influencing participant's decisions with a potentially confusing condition. Moreover, choosing a technique which allows for all tasks (length, width, height) to be carried out with one gesture, somewhat overlaps with the combination task of this section, possibly confusing the participants, too.

These issues directly relate to the results of the next two survey blocks. However, results of the rotation block suggest that the ArcBall rotation made a positive impression with the participants, although it is not common in 3D CAD applications. It is possible though that they might be heavily biased by the video previously showing this exact behaviour as an example for object centre rotation.

The most valuable contributions gained in the last (optional) block of the survey are the written submissions regarding the first question of the block about general combination of tasks. Four participants submitted that they did not like the idea of any

[2] `http://www.apple.com/ipad/`

combined translation, rotation or scaling tasks suggesting that they prefer separation of input tasks over combined usage. Also one participant pointed out that camera navigation should be always available in combination with every object manipulation task.

General Discussion

Results of the study are two-faced. As the first part of the study suggests that the theme of touch interaction for 3D applications is not too complex for participants to cope with, results of the second part indicate that they might have had problems choosing an answer since often votes are distributed over all answers. On the other hand this could be due to too minor differences between the presented gestures - especially in the rotation and scaling case and/or insufficient means providing answers in a comprehensible and unambiguous way.

Due to the results of the translation block we may choose to provide one and two touch translation in our prototype application in comparison to a point of view dependant translation technique. Secondly, we aim to allow as many scaling tasks as possible to be performed by two touch movement. For rotation a combination of the ArcBall rotation and another technique allowing to restrict the rotation around the object's centre to a certain axis seem beneficial. This might provide users with an alternative in cases they struggle to perform the intended rotation with the ArcBall [23].
Moreover, being able to rotate around a point which is effectively a rotation with simultaneous translation, seems of interest to many participants, thus we might provide it in one of our prototype interfaces. As a contrast we might also provide a rotation+scaling technique as this combination of tasks gained considerably less votes in the referring question.

Despite the complexity and length of this survey many design issues remain untouched. Most importantly the difference between separation of task DoF might not be pointed out precise enough. In the translation case for example the back-front movement was shown in the video as an axis restricted translation ignoring two available degrees of freedom ((z)x or (z)y movement). To not confuse the participants we pointed out the difference to the first task (xy-plane). However, participants could have interpreted the question differently. Some participants might have chosen their answers to replicate the behaviour of the example video as precise as possible while others might have interpreted the translation of the second video as a zx- or zy-translation,

since explanation of the video did not explicitly restrict the translation to the z-axis only.

Providing a survey which explains participants the possible gestures more clearly, maybe by providing animations for each gesture, could lead to more distinguishable results similar to the translation block. Also restricting the survey to either professionals or beginners could provide valuable insight on different user expectations.
On the other hand providing any answer possibility at all, could alter with the participants' decisions. Yet, not presetting any answers could lead to very low participation rates and difficult result analysis. Nevertheless we think that overall difficulty and length of our survey were too high. Future surveys could be designed in a trialed fashion in several steps. Each survey should cover the topic with less depth starting with the focus on identifying possible techniques for separated tasks ignoring all combination issues. The following survey(s) should use these results to explore certain design issues with more detail. The follow up surveys therefore should only be available to the audience who participated in the first survey.

5 Prototype Design & Implementation

Orienting on our initial design and the survey results, we implemented three prototype interfaces with various differences compared to each other, which explanation will be subject of this chapter.

5.1 Gesture Recognition Framework

Directly connected to the design of our interfaces is providing a system which is able to distinguish between the various interaction techniques incorporated in the interface. On one hand, the more means this system offers to differentiate between input gestures, the more diverse these gestures can be. Thus more interaction tasks can be theoretically combined without the need to use GUI elements for switching interaction modes. On the other hand, these means might put new constraints on the user, possibly increasing complexity of interaction with the system. Therefore usability of our interfaces is constrained by the quality of our gesture recognition system.

A common model to use is the differentiation by touch count. Two touch movement can for example be used to rotate, scale and translate an object in 2D. However, providing a technique for each task to separate them would need more active touches than two, if the touch count model is still used to map input to output. By extending the recognition model with "gesture matching, magnitude filtering or handles", as described by Nacenta et al. [32], unintended manipulation can be reduced by separating interaction tasks within the same technique, which would otherwise be performed simultaneously.

Yet an advantage of multi-touch input is to be able to perform several tasks simultaneously. But it depends on the application and user preference which and how interaction tasks or task DoFs should be separated or combined. While one application might benefit from heavy separation of all input tasks, another might not, yet another could benefit from providing both, separated and combined interaction tasks. Considering the limitations by the hardware available for our evaluation, we need to distinguish between all chosen interaction tasks using touch count, position and id only. To not influence relative comparison of interaction techniques of our three interfaces

during evaluation, the same gesture recognition system should be used for all interfaces.

Recognition Phase One

Our recognition system is implemented as a three phased process. Starting out with stabilising touch recognition, gathering meta data about present touches and consolidating the TUIO[1] protocol messages send by the touch capturing software Community Core Vision[2]. The TUIO messages are received over UDP and then forwarded as *RawMultiTouchTouchEvents* by VIARGO [41], a library supporting several input devices, following the action listener concept. These events are processed in the first stage of our recognition process. The *DataConsolidationPass* class reacts on various different touch *Action* states such as APPEARED, UPDATED or DISAPPEARED. It consolidates the *RawMultiTouchEvent's* data by rearranging multiple touch events from the TUIO protocol into a *TouchPool* data structure offering the position and ID of a variable amount of active touches and a map of their last states. Changes in this structure are emitted through the *ExtendedMultiTouchEvent* for further processing. In figure 5.1 a class diagram of this process can be reviewed.

With the *ExtendedTouchPoolManager* class which organises this data structure, we also introduce a form of "Dead-Touch-Remapping" and touch movement detection via threshold, while also saving a brief history of disappeared touches.

Dead-Touch-Remapping

When dragging or pushing one or more fingers over a surface it can happen that some or all fingers lose contact to that surface for a brief moment of time. Responsible for this are circumstances leading to altered friction during interaction, for example sweaty fingers or moving the fingers in an adverse angle with too much or too less pressure applied to the surface. The gesture recognition process should return the same touch ID for the same finger until it is intentionally removed from the surface. This could increase liability of touch ID tracking which may directly relate to the overall robustness of the complete recognition process.
Figure 5.2 illustrates the life time of a touch including our id remapping process which is handled in the *onAppeared* method in the *ExtendedTouchPoolManager* class. Active touches are ALIVE, while touches which disappeared are set to the state ZOMBIE. If a

[1] http://www.tuio.org/?specification
[2] http://ccv.nuigroup.com/

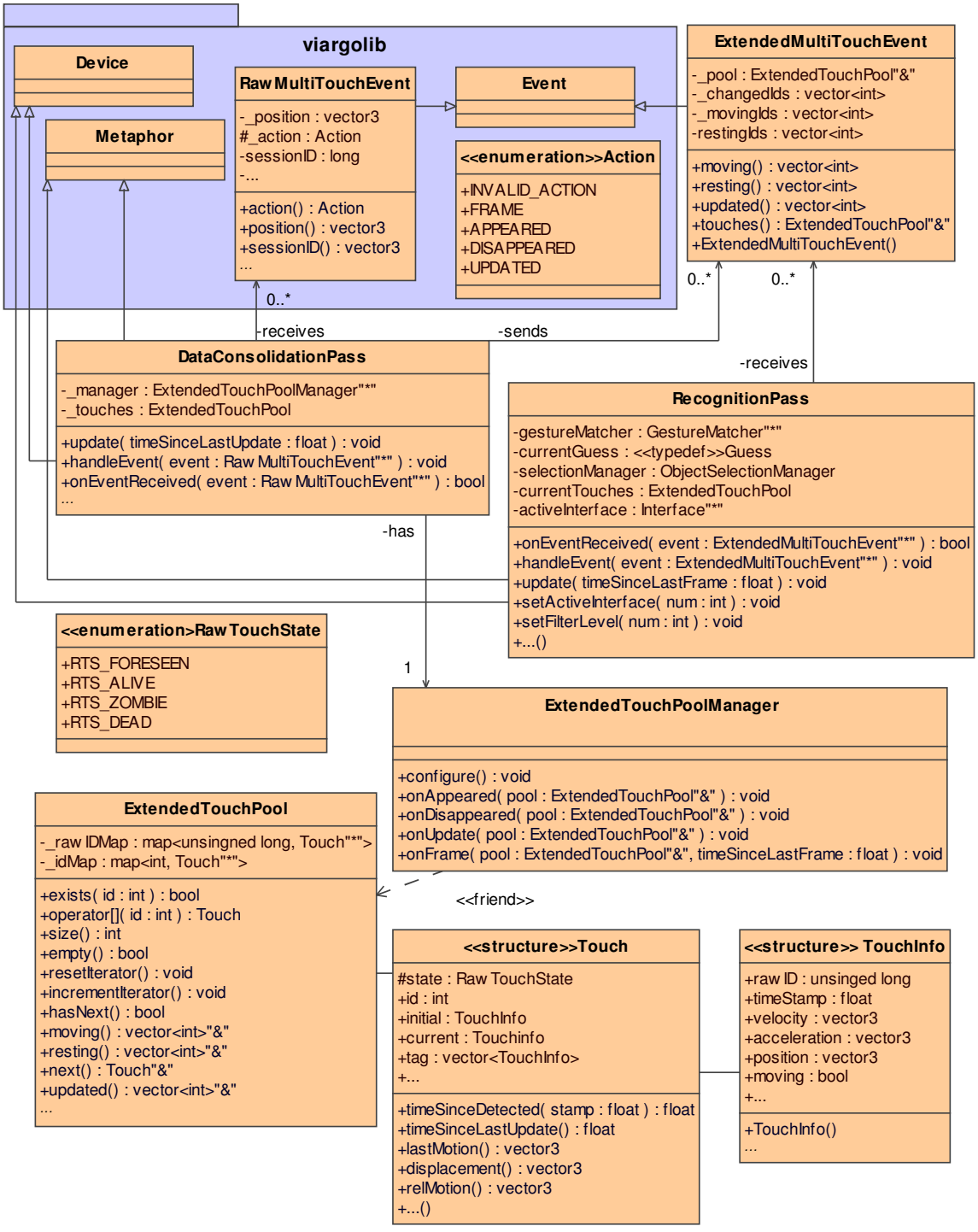

Figure 5.1: Class Diagram of the first recognition phase. This phase consolidates the raw touch data into a more sophisticated structure and gathers some meta data about the touches. Phase two starts with the *RecognitionPass* class which is described further below.

new touch appears it is checked, if a ZOMBIE touch was in proximity to that new touch position - if a match can be found, the ZOMBIE is set to ALIVE taking the *TouchInfo* of the appeared touch as its new current state. If the lifespan of a ZOMBIE is reached before it is resurrected, it becomes a DEAD touch and is removed from the *ExtendedTouchPool*.
We can also keep track of DEAD touches for a certain amount of time, but, if reached the DEAD state they are not eligible for remapping anymore.
Users may not always remove multiple fingers from the surface exactly at the same time when performing interaction techniques which, for example, react on touch disappearance. Thus keeping track of the DEAD touches and their time of disappearance may provide additional data to assist in recognition of such techniques.

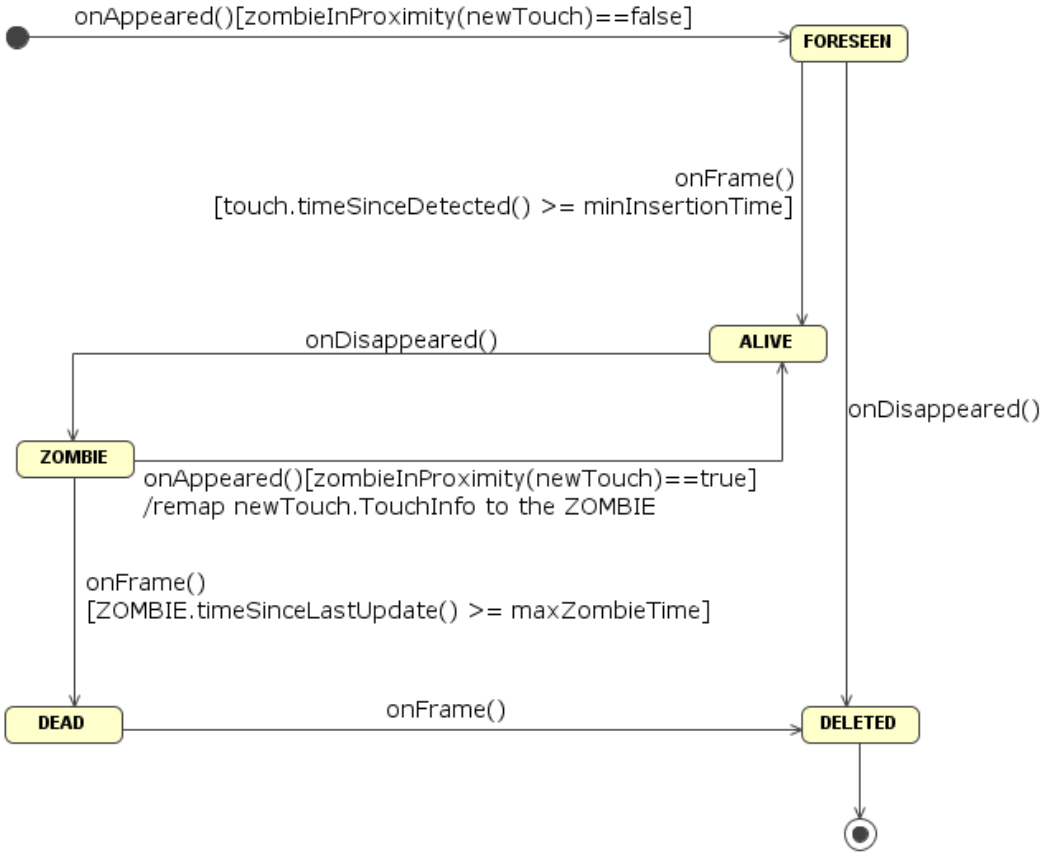

Figure 5.2: State diagram illustrating a touch's states of existence.

Resting and Moving Touches

In contrast to a mouse which sends no motion events when resting, a touch, although not intentionally moved by the user may occasionally emit a position change. This can be due to minor differences in pressure applied from the user to the surface or jitter occurring in computer vision algorithms. This could result in false recognition of techniques which involve usage of one or more resting touch points to provide, for example, a point or axis of reference.

A pseudo implementation of the algorithm to differentiate between RESTING and MOVING touches can be reviewed in listing 5.1. As part of phase one of our gesture recognition process this mechanism is called each time a touch is updating its position *onUpdate()* in the *ExtendedMultiTouchManager*. Differentiating between these touch states provides a way of separating tasks or task DoF, allowing us to use the same amount of touches for different tasks or combinations of tasks. For example a two finger pinch gesture with a resting finger or with two moving fingers could potentially be used to perform two different tasks. On the other hand, we have to carefully decide, if and were we want this distinction to have any effect. Although this distinction increases the amount of interaction tasks we can map to interaction techniques, a mechanism which influences the raw input of the user could increase chances of unintended manipulation. This could be due to additional input constraints which the user has to adapt to, as for example more consistent movement. Thus false detections of touches as RESTING when moving may confuse the user as well as a touch MOVING when resting. In general we are trying to deduce a continuous property (MOVING,RESTING) from discrete input events, thus higher polling rates when capturing touch input potentially increases liability of our algorithm. Modern mice have polling rates up to 1000hz which relates to a delay of one millisecond. Optical touch capturing at 25 fps is forty times slower.
To visualise a touch recognised as RESTING or MOVING we decided for a similar, but less sophisticated version of the "Ripples" system by Wigdor et. al. [45]. Visualisation of our touches can be reviewed in figure 5.3. A red dot hereby indicates that a touch is RESTING.

```
bool checkifTouchisMoving(Touch& touch){
    vec3f center, curpos, initialpos;
    center = touch.restingZoneCenter; //default:(0,0,0)
    initialpos = touch.initial.position;
    curpos = touch.current.position;

    //The touch needs a new reference point (origin) to which its current position can
```

```
    //be compared to.
    if(touch.newOrigin){  //default:true
      //The touch has left the origin position and the secondary check was not yet
      //performed.
      if(touch.leftOrigin == true && touch.secondaryCheck == false){//default:false
        //set the center of the zone determining the RESTING state.
        touch.restingZoneCenter = touch.current.position;
        center = touch.restingZoneCenter;
        //follow up with the secondary check
        touch.secondaryCheck = true;

      //Touch has not left the origin position and a secondary check was not yet
      //performed
      }if(touch.leftOrigin == false && touch.secondaryCheck == false){
        //Touch has not even left the origin similar to its initial position
        //-> it has never moved before
        if(!touch.leftInitialOrigin){//default:false
          //sets the center of the zone determining the RESTING zone to the initial
          //touch position.
          touch.restingZoneCenter = initialpos;
          center = touch.restingZoneCenter;
        }else{
          //touch has moved before -> set center to current touch position
          touch.restingZoneCenter = touch.current.position;
          center = touch.restingZoneCenter;
        }
      }
      //we now have a new resting zone center -> do not renew it for now
      touch.newOrigin = false;
    }
    //touch is not yet classified as moving, but a resting zone is specified and
        secondary
    //check was still not performed.
    if(touch.current.moving == false && touch.leftOrigin == false
                          &&  touch.secondaryCheck == false){
      //check, if the current touch position is in proximity of radius units
      //to the resting zone center specified before.
      if(pointinCircle(curpos, center, radius=0.05)){
        //still not moving
        return false;
      }else{
        //the touch left its resting zone - this will allow update of the resting
        //zone center, if its triggered by "newOrigin"
        touch.leftOrigin = true;
        //especially it left its initial origin position
        touch.leftInitialOrigin = true;
        //and its moving now
        return true;
      }
    //the touch is MOVING now and secondary testing has been ordered
    }else if(touch.current.moving == true && touch.secondaryCheck == true){
      //longer then 500ms in here -> RESTING again
      if(touch.timeInRestingZone < 500){ //default:0.0
```

```
        //if the touch is moving, but is still nearer to the center
        //of the newly specified resting zone than to its outside
        //->increase its timeInRestingZone
        if(curpos.positionCloses(center, 0.001)){
          if(!touch.tag.empty()){
            touch.timeinRestingZone += touch.current.timeStamp -
                                touch.tag[0].timeStamp;
          }
          //touch still MOVING
          return true;
        //if touch is still in the resting zone, but nearer to the outside
        }else {
          //passed secondary Check->deactivate for now
          touch.secondaryCheck = false;
          //reset time in resting zone
          touch.timeinRestingZone = 0.0;
          //still MOVING  (but potential RESTING)
          return true;
        }
      //the Touch has stayed half a second in the resting zone without moving too much
      }else if(touch.timeinRestingZone >= 500){
        //reset everything
        touch.timeinRestingZone = 0.0;
        touch.restingZoneCenter = curpos;
        center = touch.restingZoneCenter;
        touch.newOrigin = true;
        touch.leftOrigin = false;
        touch.secondaryCheck = false;
        //Touch RESTING again
        return false;
      }
    //Touch is still MOVING, but is potentially RESTING in the current resting zone
    }else if(touch.secondaryCheck == false && touch.current.moving == true
          && touch.leftOrigin == true && touch.newOrigin == false) {
      //acceleration dependant check if touch is resting again...
      if(!touch.tag.empty()){
        //touch has little acceleration
        if(touch.current.acceleration <= 0.002){
          //enforce update of resting zone position
          touch.newOrigin = true;
        }
      }
      //touch is fast enough to leave resting zone before next frame
      //->MOVING again
      return true;
    //in case we forgot something
    }else return touch.current.moving;
    //forward state if something is very wrong
    return touch.current.moving;
  }
```

Listing 5.1: Classification of Touches to the RESTING and MOVING states.

Scene Context Dependant Gesture Recognition

Working in free 3D space implies the occurrence of very large or small objects possibly exceeding screen dimensions or being smaller as the finger touching them. If gesture recognition would be dependant on an object being touched or not, especially in unrestricted 3D environments, users could easily encounter situations where they are unable to perform an intended technique because they can't meet their prerequisites. Introducing a freely selectable offset we allow the user to perform object manipulations anywhere on the screen. Hereby we might reduce issues due to the fat finger problem. This however, may reduce the degree of directness of the interactions in such cases, too. Another downside of this design is that information, which could be used in gesture recognition, is not taken into consideration. Scene context dependant recognition enables a single technique to execute several interaction tasks, possibly leading to increased interaction richness. For example touching a wall of a room with one finger translates the view while touching an object will result in its translation.
Because we believe that accuracy and speed are more important properties then ease-of-use when interacting with 3D CAD applications, we choose to provide a freely selectable offset for some manipulations expecting to improve accuracy while sacrificing directness to some degree. Since our gesture recognition is scene context independent we can apply the recognised motion to the corresponding object manipulation task, as if the touch was on the object without needing an additional technique or button to define the off-set.

Recognition Phase Two

Each of our interface prototypes has a configuration file. The *interface definition files (*.idf)* contain a set of parameters used in the recognition process to map the user's input to all tasks available through the interface. Figure 5.4 shows a configuration file of a single interface. In corner-brackets the target and type of a single interaction technique are provided as for example the *[Object \ Translation]* technique. Until a new technique is defined the following lines are of the form *axis=x,x,x,x*. Each line configures one task DoF separately. Since in the example entries for x and y are the same, local object translation will be performed by a single touch, while the z-axis translation will involve usage of two touches. Thus the first entry of the comma separated values provides the number of touches involved to perform the technique, the second entry presents the number of touches which need to be MOVING. The third entry is set to the touch motion type required which is coded to an integer number. The last entry is a special

purpose parameter. For example the *3* in the *"all=6,6,2,3"* line states that instead of six MOVING touches, three MOVING and three RESTING are working too for this technique.
As long as the configuration file entries for each manipulation technique are unique, they can be used as a comparison template to the provided touch input during phase two of our gesture recognition.

Whereas touch amount and number of MOVING touches, as required by the configuration files, can be deduced from the meta information generated in phase one, determining the touch movement type offers many challenges. The prototype implementation can identify two touch movement types: GROUPED or RELATIVE. Figure 5.5 illustrates the results of such a classification process of four touches due to different movement. In the figure group A moves RELATIVE to group B while all touches in group C move GROUPED. But group C itself moves RELATIVE to all other groups. This differentiation for example allows to control object position and scale separately when usage of the same number of touches is intended by the interface design for these tasks as one of the tasks could be assigned to GROUPED and the other to RELATIVE motion. Distinguishing a third movement type as for example ROTATING is much more complex. Using these three movement types would require the recognition system being able to decide for a task by means of at least two parameters. The distance and angle between any two touch points or groups of touches. Thus the gesture recognition needs to decide for an interaction technique based on the magnitude of angle and distance change of the active touches.
However, we refrained from implementing this third motion type instead we allow separation of rotation on a per technique basis. Thus, if we intend to separate the rotational component from the scaling task in a two finger pinch gesture we can still do this after the gesture recognition on the interface level. In such a case the entries in the configuration file for object rotation and object scaling would be identical.
Phase two will return a structure, called guess, containing all the information required for the interface classes to perform the recognised interaction task. In the case of the example provided in 5.4 one touch movement will result in the return of *object/translation/xy* in addition to multiple vectors containing the involved touches' positions and ids. This id vector contains a vector for each group of touches (GROUPED). Therefore the arrangement of ids in this vector codes the movement types of all involved touches. The class diagram shown in figure 5.6 provides an overview of the second and third phase of the recognition process. The *ObjectSelectionManager*, *GestureMatcher* and the *Interface* class are connected to *RecognitionPass* class (see 5.1).The *GestureMatcher* makes

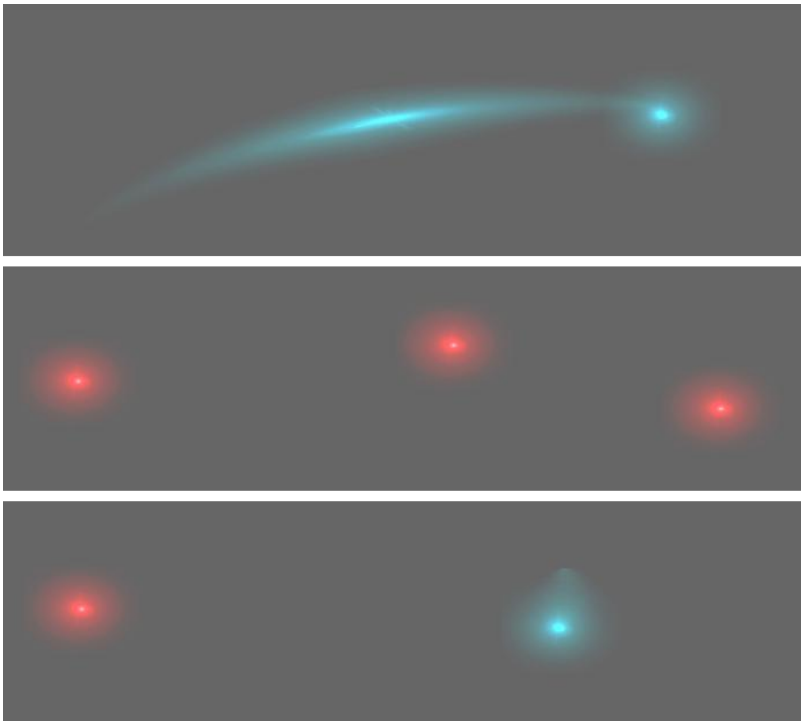

Figure 5.3: Contact visualisation of touch input in our prototype

```
#modifier numberoftouches numberoftouchesmoving movementtype, special
[Object\Translation]
x=1,1,1,0
y=1,1,1,0
z=2,2,1,0
[Object\Rotation]
x=3,3,1,0
y=3,3,1,0
z=3,3,1,0
custom=5,3,2,1
[Object\Scale]
x=2,2,2,0
y=2,2,2,0
z=3,3,2,2
all=6,6,2,3
#[Object\Destruction]
#nomodifier=5,4,2,0 |
[Object\Selection]
nomodifier=3,0,1,D
[Camera\Translation]
x=4,4,1,0
y=4,4,1,0
z=4,4,2,2
[Camera\Orbit]
nomodifier=5,5,1,0
```

Figure 5.4: An example entry of a configuration file of our interfaces. These entries are used in gesture recognition and provide the mapping of user input to all tasks available through the interface.

a *Guess* on the current input by comparing it to the *InterfaceDefinition* of the currently active user interface. Phase three of the recognition process is encapsulated inside the *GestureMatcher* class.

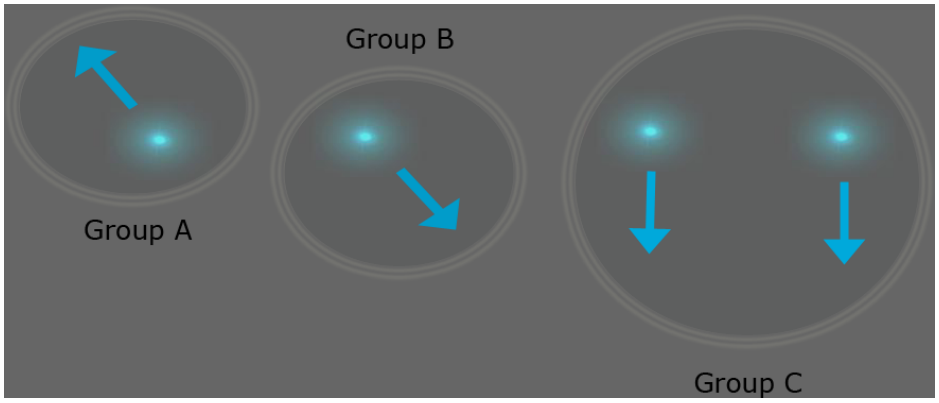

Figure 5.5: Illustration of touch movement type classification to GROUPED or RELATIVE: group A moves RELATIVE to group B and C while touches in group C move GROUPED, but RELATIVE to all other touch groups and their corresponding touches.

Phase Three

In order to keep input lag to a minimum and allow switching of techniques without a complete retouch of the screen, we decided for a frame-to-frame recognition. But since changes in touch position may be very small from frame to frame, we need a mechanism smoothing out false recognitions. This is done in the third phase of the recognition. A filter collects the phase two guesses of the last ten frames and makes a stochastic approach to return the most probable solution. However, switching a technique now requires a certain threshold to be reached resulting in a slight delay of 10 frames or 0.4 seconds for the interface to react on a technique switch. After a gesture has been recognised and remains unchanged, the interface reacts as fast as touch capturing allows.
Figure 5.7 provides a flow chart of this smoothing filter algorithm which can be found in the *GestureMatcher* class.

5.2 User Interface Prototypes

Our gesture recognition system allows us to differentiate between different types of multi-point input not only by its quantity, but also by its type of motion. This section

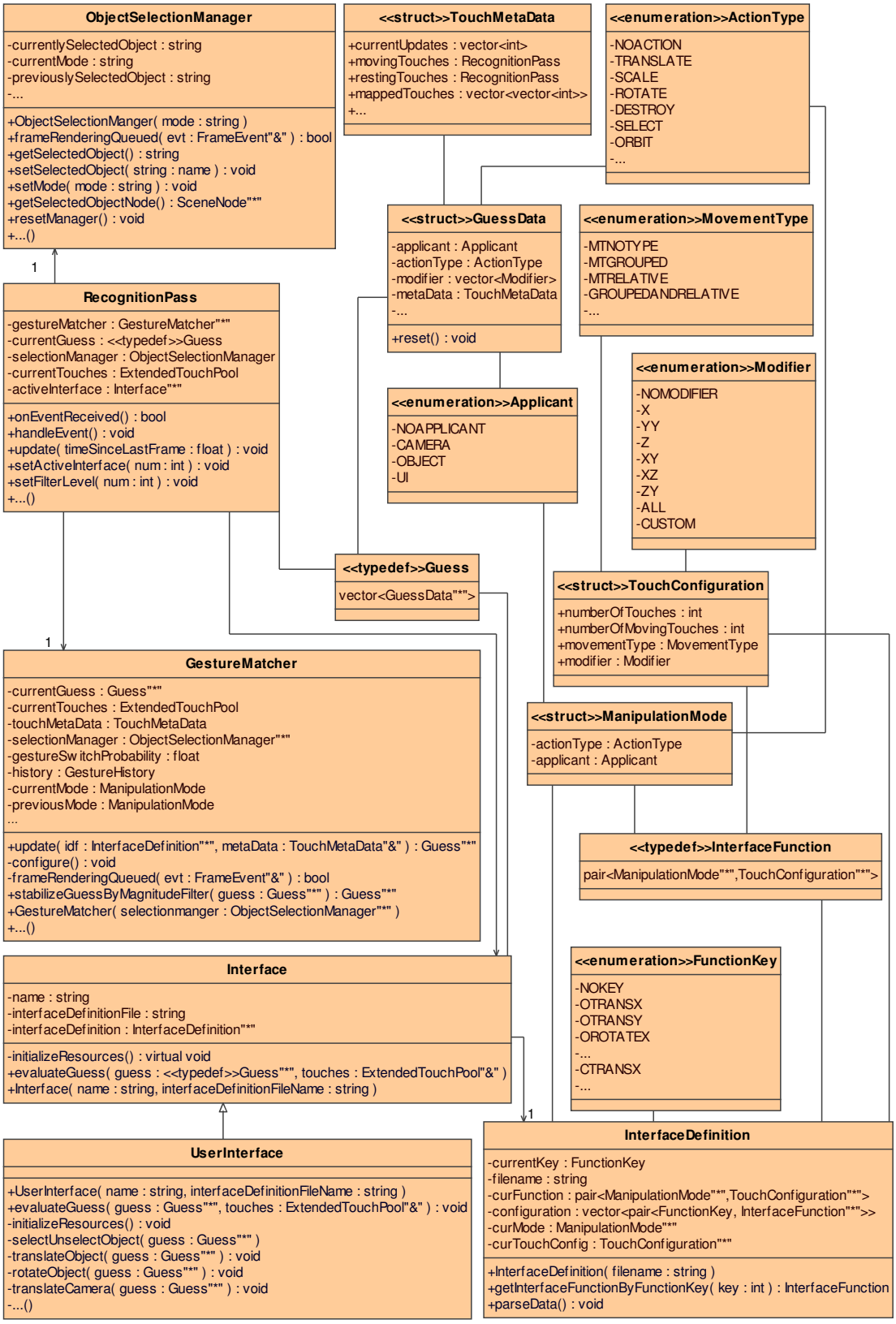

Figure 5.6: Class diagram of the second and third phase of our gesture recognition process.

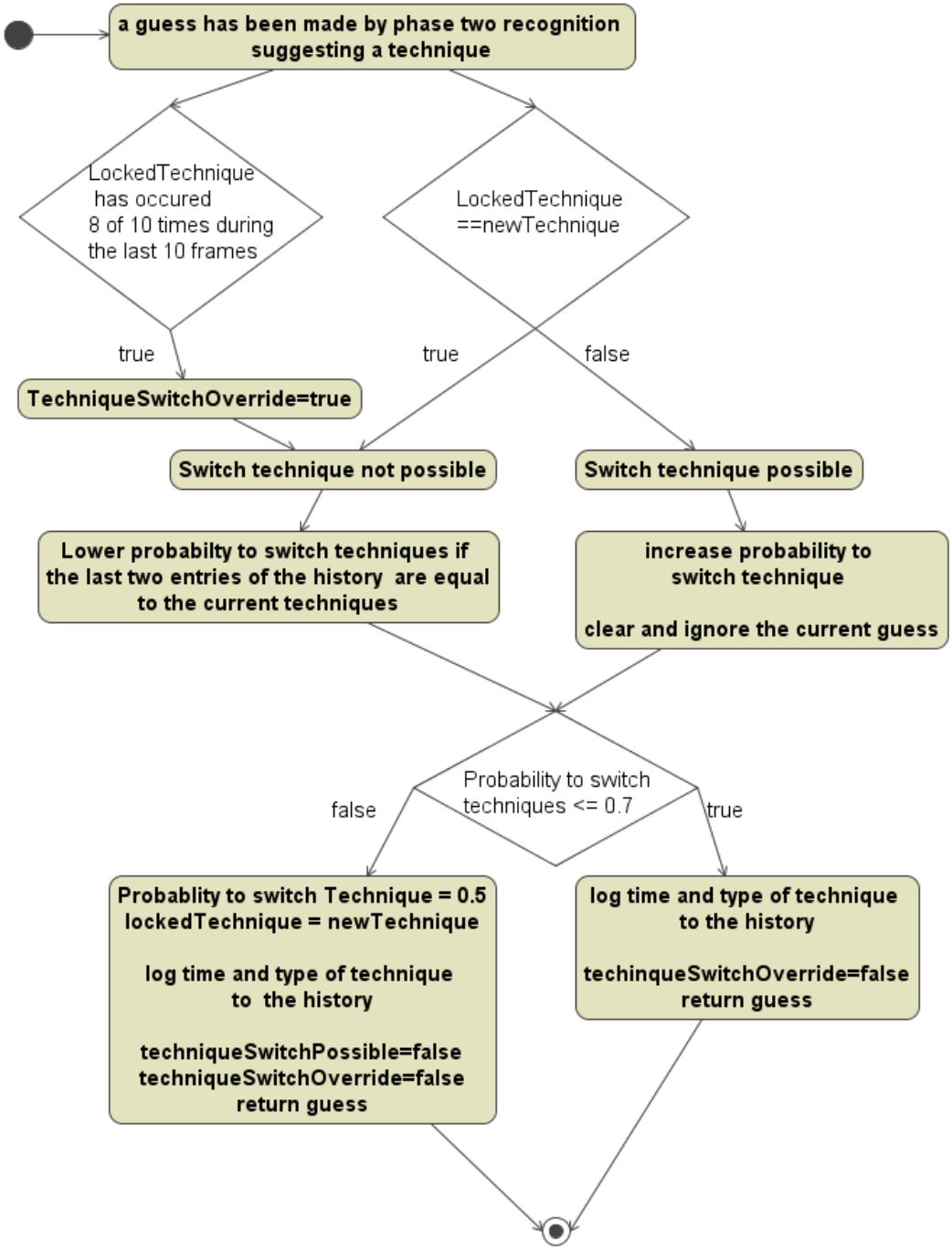

Figure 5.7: Flow chart of the guess smoothing filter.

explains where and why we used this additional information to extend or adapt our initial interface design.

5.2.1 Camera Navigation Techniques

Users of 3D CAD applications need to manipulate the view to be able to watch the scene from different angles and positions, but at the same time they want to focus on manipulating scene contents not the camera. Therefore camera navigation can be seen as a necessity to work within an unrestricted 3D environment and not as an optional manipulation mode which could be present in one interface and absent in another. All prototype interfaces use the same camera navigation technique to allow identifying differences in object manipulation techniques more precisely. This will also establish a base line for all users during evaluation allowing their comparison.
Figure 5.8 illustrates the camera navigation tasks all interfaces are capable of. Similar to the initial design the view can be panned, zoomed and rotated not with two, but with a four touch input, requiring the user to move two touch pairs, which themselves are considered GROUPED. Each group has a centre point between its touches. These two points are taken as the actual input point of the interaction. Their movement is then mapped similar to the way the RTS gesture works, resulting in a four finger bi-manual direct interaction gesture. Camera orbit can be performed using five GROUPED touch points.

To create the impression of direct camera manipulation for the panning, zooming and rolling tasks a plane of reference has to be chosen to which the input motion can be mapped. Determining this is done with a vector to plane projection. A ray is cast from the centre of the screen and the resulting direction vector is projected onto each of the scenes planes (xy,xz,yz). The shortest of the projected vectors decides which plane will be used. We allow direct control for the panning and zooming in regards to that plane, but not for rolling.

Imagine a two touch camera roll technique. To create the impression of direct interaction for this task the camera needs to roll in a way that the scene's points touched stay underneath the fingers during interaction. This however, would imply not only a change to the camera's roll angle, but also its position since the roll task is a rotation around the local camera z-axis and the scene points touched are most likely not located on a plane perpendicular to that axis. As we combine camera roll and translation (including zoom) into a single technique we cannot guarantee directness of all three interactions, because the technique's reference points are located on different planes.

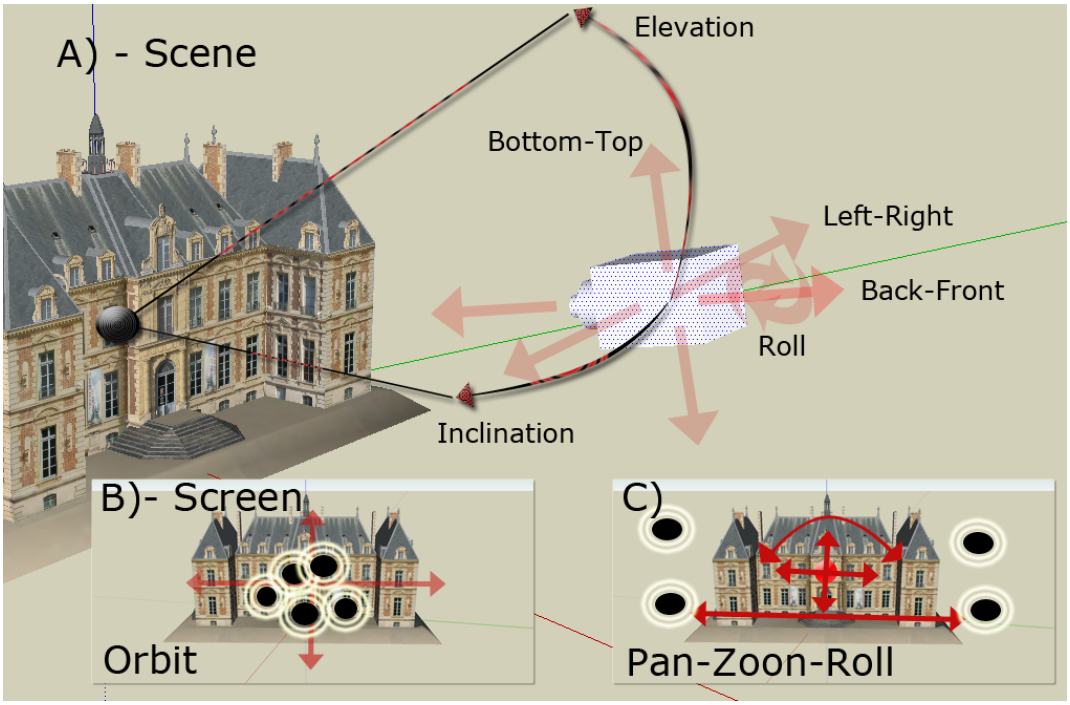

Figure 5.8: The Camera Navigation Metaphor. In A) the camera is shown from a external view point. The arrows indicate the possible translations of the camera. In B) and C) the touch mapping to the available tasks is shown. In both figures black dots represent touches on the screen, while the arrows indicate the required movement. The red dot in C) represents the midpoint of the four touches. The midpoint movement is mapped to the panning tasks, while differences in distance are mapped to zooming.

On the other hand a direct manipulation of all tasks can be achieved, if camera translation and roll were "sticky" to a plane parallel to the view port. But this plane often lies in mid space so the user would touch invisible points so that the overall impression of the interaction might not be a direct one.

Creating the impression of zooming directly in free 3D space creates another problem. When zooming directly we cannot zoom past the point of reference, since the amount of camera translation is related to the distance of the camera to that reference point, preventing it from reaching negative values. To avoid getting "stuck in detail", we set a minimal distance the camera needs to have from the reference point in order to translate in a direct manner. Disabling directness when undershooting a certain distance to the reference point and continue zooming with a constant factor allows to zoom with no limitations into every direction. This enables the user to navigate through the currently "sticky" plane, but at the same time sacrifices direct interaction the moment this minimal distance is undershot. However, directness is "re-enabled"

when, after crossing the reference point, the camera's absolute distance to that point is again larger than the minimal distance required. However, as the user zoomed through the plane of reference the reference points cannot be seen anymore.

Instead of a reference plane, camera orbiting requires a rotational centre or gravity point. The default way of determining the gravity centre in our implementation is to project the screen centre onto the most perpendicular world plane. But, if there is an object near the screen's centre and closer to the camera than the most perpendicular world plane, the objects local coordinate system's most perpendicular plane is taken for determining the gravity centre.
We choose to do this, because users might intend to orbit around an object, too. This might even be the primary purpose of this technique. Early iterations of the orbit technique required the user to specify the gravity centre with one touch and another four touches on the other hand to imply the motion. Another technique would orbit around the currently selected object. But as both techniques resulted in a sudden camera jump to align the camera centre with this focus point we resorted to the behaviour described above.

Implementation

For our pan-roll-zoom technique we describe the change in the camera view point model $V(p_0, d_0, u_0)$ with $u \in \mathbb{R}^3$ being the cameras up-vector, $d \in \mathbb{R}^3$ the look-vector and $p \in \mathbb{R}^3$ the camera position.. We derive three values from the input touches. The number of active touches n, $t_i \in \mathbb{R}^2, i = 1..n$ for the current and $\acute{t}_i \in \mathbb{R}^2, i = 1..n$ for the previous touch positions. Also we define:

$$v : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^+$$

$$v : (t_0, t_1) \to \sqrt{(t_0(x) - t_1(x))^2 + (t_0(y) - t_1(y))^2)}$$

as the distance between any two points. Furthermore

$$\psi : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^2$$

$$\psi : (t_0, t_1) \to ((t_0(x) + t_1(x)) * 0.5, (t_0(y) + t_1(y)) * 0.5)$$

provides the midpoint between two touches.
Thus $\psi : (t_0, t_1) - \psi : (\acute{t}_0, \acute{t}_1)$ returns the motion vector of the midpoint after a motion event, while $\frac{v:(t_0,t_1)}{v:(\acute{t}_0,\acute{t}_1)}$ returns the ratio of the distance change between any two touch

points. The angle in radians between two vectors d and $\acute{d} \in \mathbb{R}^2$ is calculated with:

$$\phi : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow [-\pi, \pi]$$

$$\phi : (d, \acute{d}) \rightarrow arctan2(\acute{d}) - arctan2(d)$$

*Arctan*2 allows to identify the quadrant of the unit circle in which the angle is located, thus we can deduce the roll direction from its sign.

Panning will transform the position p_0 of the camera to p_1 with:

$$p_1 = p_0 + (\Delta(\psi(\psi(t_0, t_1), \psi(t_2, t_3)) - \psi(\psi(\acute{t}_0, \acute{t}_1), \psi(\acute{t}_2, \acute{t}_3))) * \xi$$

with ξ being the distance between the camera's initial position and the reference point on the most perpendicular plane. Δ provides the unprojection of screen motion.
The local z-axis translation of the camera can be retrieved with:

$$p_1 = p_0 + (\partial * (0, 0, \delta z)$$

With ∂ representing the camera's orientation as a quaternion $Q \in \mathbb{H}$ and

$$\delta z = \xi - \frac{\xi}{(\frac{v:(\psi(t_0,t_1),\psi(t_2,t_3))}{v:(\psi(\acute{t}_0,\acute{t}_1),\psi(\acute{t}_2,\acute{t}_3))})}$$

The camera's up-vector u_0 is converted to the new up-vector u_1 with:

$$R(\phi(\psi(t_0, t_1) - \psi(t_2, t_3), \psi(\acute{t}_0, \acute{t}_1) - \psi(\acute{t}_2, \acute{t}_3)), d_0) * u_0$$

with $R(\alpha, \beta)$ being the rotation matrix defined by the rotation angle α and the axis of rotation $\beta \in \mathbb{R}^3$. Simultaneous manipulation of all these properties can be achieved by consecutively applying these transformations to the camera during the same frame.
In listing 5.2 a pseudo implementation of our camera pan-zoom-roll technique can be reviewed while listing 5.3 provides an implementation of our orbit technique (see page 21).

```
void InterfaceName::translateCamera(Guess* guess){

  if(translateIDs!= lastTranslateIDs){
    distanceOld = 0.0;
    ratio = 0;
    cop = getCenterOfProjection(); //get reference point and distance to it
  }
  if(!currentTouches[cameraIDs].tag.empty()){
      Vector2 lastPos1 = currentTouches[cameraTranslateId0].tag[0].position;
      Vector2 curPos1 = currentTouches[cameraTranslateId0].current.position;
      Vector2 lastPos0 =currentTouches[cameraTranslateId1].tag[0].position;
      Vector2 curPos0 =currentTouches[cameraTranslateId1].current.position;
      Vector2 lastd, curd;
      lastd =  lastPos1 - lastPos0;
      curd = curPos1 - curPos0;

      lastd.normalise();
      curd.normalise();
      float angleBetween = atan2(lastd.y,lastd.x) - atan2(curd.y,curd.x);//calc the
          angle

      cam->roll(Radian(angleBetween));
      curmidPointGlobal = getMidpoint(cameraIDs);
      distance = getDistance(cameraIDs);
      //if there are last states
      if(distanceOld != 0) {
        ratio = distance / distanceOld; //scalefactor in %
        screenMotion = curmidPointGlobal - lastmidPointGlobal;
      }
  }
  Vector3 sceneTranslation(0,0,0);
  if(screenMotion !=vec2f::zero){
     sceneTranslation = unprojectVector(Vector2(screenMotion.x, screenMotion.y));
  }
  Real dist = cop.first;
  cam->move(Vector3(-sceneTranslation.x* dist,
              -sceneTranslation.y* dist,
              -sceneTranslation.z* dist));
  }
  // Zoom PART
  if(ratio != 0.0){
    float resize = dist/ratio;
    dist = dist - resize;
    cam->moveRelative(Vector3(0,0,-dist));
  }
  lastTranslateIDs = translateIDs;
  lastmidPointGlobal = curmidPointGlobal;
  distanceOld = distance;
}
```

Listing 5.2: Camera Pan-Zoom-Roll Implementation

```
void InterfaceName::orbitCamera(Guess* guess){
  bool neworbitcenter = false;
  if(orbitIDs!=lastOrbitIDs)
  {
    neworbitcenter = true;
  }
  if(neworbitcenter){
    centerOfProjection = getCenterOfProjection();
    jawAxis = cam->getOrientation().yAxis();
  }
  if(!neworbitcenter){
    float factorx= 0;
    float factory= 0;
    vec3f lastmotion = currentTouches[id1].lastMotion();

    factorx = lastmotion.x*90;
    factory = lastmotion.y*90;

    cam->setPosition(centerOfProjection.second);
    cam->setFixedYawAxis(true,jawAxis);
    cam->yaw(Degree(-factorx));
    cam->pitch(Degree(factory));
    cam->moveRelative(Vector3(0,0,centerOfProjection.first));
  }
  lastOrbitIDs = orbitIDs;
}
```

Listing 5.3: Camera Orbit Implementation

5.2.2 Object Translation Techniques

In contrast to the camera navigation's counterparts in mouse driven applications, object manipulation often relies on the usage of widgets or menus to support manipulations of all object dimensions independently. To create the impression of direct interaction for object translation we need to decide for a reference plane to which the touch motion will be applied. The object can then be translated directly on that plane. However, a user might intend to translate the object in relation to a plane which was not chosen by the system - for example because the intended plane is not the most perpendicular one. Nacenta et al. developed the Z-Technique [29], a combination of indirect and direct interactions to relate to this problem. Using two touches in order of appearance, they mapped the movement of the first touch to a xy-translation of the object and applied positive or negative z-axis translation to the object, if the second touch was moved upwards or downwards.

Independent two touch movement as required by this technique however, somewhat intersects with recognition of other two touch techniques in our design. Thus supporting the Z-technique within our interfaces would require a more complex input model for the gesture recognition process as its motion is of the RELATIVE type which is indented for scaling (see section 5.1). Thus we restrict the user to object translation in the most perpendicular plane with one touch and require a change in view direction to access every other manipulation plane. This technique might be usable to some degree depending on the performance of our camera navigation technique (see fig. 5.9). Two touch GROUPED movement, which is still available, is mapped to the least perpendicular plane allowing the user to translate the object in a direct manner in every direction independently of the cameras viewing direction. However, the perceived result when performing two touch grouped movement may not always coincide with users intentions, because without further aids, for example which side of the object was touched, it is impossible to deduce which of the remaining two planes the users might intend to use.
Moreover, all object translations in our interfaces are global. Further differentiation between the local and global coordinate systems is thinkable though it may require additional techniques or additional means to differentiate the users intentions. Also, instead of choosing a plane to apply the motion to when using the two finger GROUPED technique, we could have restricted this translation to a single axis only. But due to situations similar to figure 3.3 we provide at least two degrees of freedom for translation in the least perpendicular plane aiming to reduce unreasonable movement, since our translation is direct and not indirect. To summarise the design for each object translation technique of our three interface designs table 5.1 provides a comparative view explaining their behaviour and touch mapping.

Interface	Touch Count	Movement Type	Type of Translation
1,2,3	1	RELATIVE	Global object translation parallel to the most perpendicular plane of the global coordinate system.
1	2	GROUPED	Global object translation parallel to the least perpendicular plane of the global coordinate system.

Table 5.1: Translation techniques available in our three interface prototypes

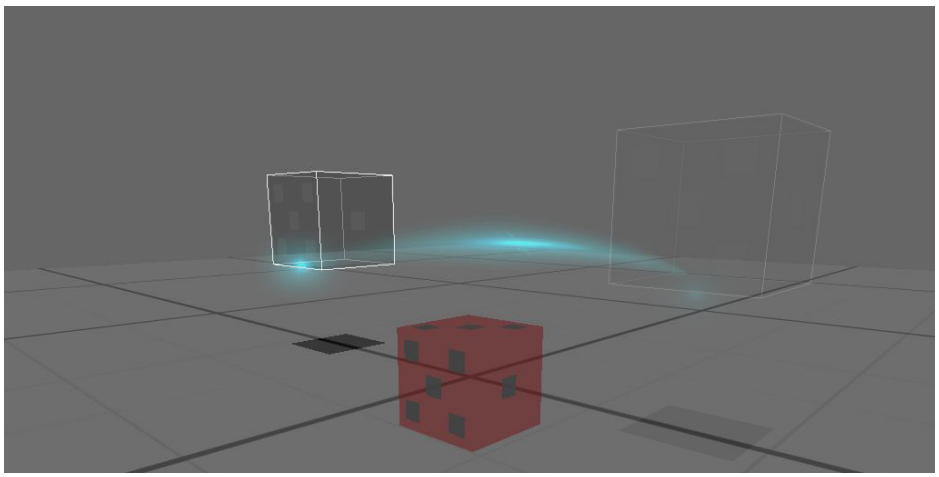

Figure 5.9: Direct global object translation in the most perpendicular plane of the global coordinate system to the camera's viewing direction

Implementation

Listing 5.4 provides a pseudo implementation of our camera dependent one touch object translation technique. The method *getPerpendicularPlane()* evaluates which is the most perpendicular world plane to the current view direction in reference to the object position and the location of the input on the screen. Calling *intersection()* returns the vector between the object's initial position and the projected input point onto the chosen plane. This method provides us with the variable offset mentioned in the initial design considerations.

```
void InterfaceName::translateObject(Guess* guess){

  bool resetOffset = false;
  int id1 = guess->metaData.movingTouches.front();

  vec3f curpos = currentTouches[id1].current.position;
  Ray ray = cam->getCameraToViewportRay(curpos.x, curpos.y);

  if(id1 != previousId){
    //try remap
    if(currentTouches.exists(previousId)){
      id1 = previousId;
      translationStarted = false;
    }else{
      resetOffset = true;
      previousId = id1;
      currentAxis = AXIS_FREE;
    }
  }
```

```
  if(resetOffset == true){
      lastpos = currentlySelectedObject->getPosition();
      plane = getPerpendicularPlane(ray, currentAxis, currentlySelectedObjectName);
      translation = intersection(ray, plane,currentAxis,
                          lastpos,
                          currentlySelectedObjectName);
      resetOffset = false;
  }
  if(translationStarted == false){
      lastpos = currentlySelectedObject->_getPosition();
      plane = getPerpendicularPlane(ray, currentAxis, currentlySelectedObjectName);
      translation = intersection(ray, plane, currentAxis, lastpos,
                          currentlySelectedObjectName);
      translationStarted = true;
      return;
    }
    if(translationStarted == true){
      newPos = intersection(ray, plane, currentAxis,lastpos,
                        currentlySelectedObjectName);
      //calcs translation vector in reference to object position.
      newPos = newPos - translation + lastpos;
      setPosition(currentlySelectedObject,newPos);
    }
  }
}
```

Listing 5.4: View Dependant 3-DoF Object Translation

5.2.3 Object Scaling Techniques

The two finger pinch gesture is often used for global object scaling tasks in 2D applications. However, users might not only intend to change the global size of an object sustaining the objects size ratios, but also alter each of the objects dimensions independently. From a developer's point of view providing all scaling tasks separately with the same input technique requires a widget or other means in order to distinguish between those tasks. For example assigning two touch RELATIVE movement inside an object would scale in one axis depending on the angle between the touches and the objects up-axis on the most perpendicular local plane. On the other hand two touch RELATIVE movement outside the object bounds would scale the object globally. But since too small or too big objects could prevent usage of either global or restricted scaling we decided to provide two different techniques for both object scaling tasks, expecting to reduce situations in which users cannot perform an intended task (see table 5.2). Although RTS suggests to combine scaling with translation - our scaling techniques involve no translation, sacrificing directness and possible performance increase in comparison to a sequential work flow to prevent unintended actions by

maintaining separation of tasks.
Since our interfaces support no undo technique, correcting unintended manipulations could be very frustrating for the users, especially, if precision is a key requirement of the application or when interaction tasks are not separated. On the other hand our camera navigation technique was designed similar to the RTS technique with combined translation and rotation to support a fluent and fast performing camera navigation technique, because we do not expect camera navigation to be as precision demanding as object manipulation.
Interface two and three allow the user to globally scale a previously selected object with two touch RELATIVE movement anywhere on the screen as long as the object is positioned in between these two touches. In addition, a pair of touches and a single touch can be used bi-manually to scale axes in the local object plane most parallel to the view port depending on the angle between touches and object axes in that plane. For the first interface we mapped two touch RELATIVE movement to axis restricted scaling and six touch (3 GROUPED touches on each hand) RELATIVE movement to global scaling. Moreover, the three touch scale technique of interface two and three can be used in interface one to access the remaining axis of the object for scaling, which is the axis most parallel to the camera viewing direction. While the scaling techniques of interface two and three depend on the camera's position to manipulate all scaling DoFs, scaling in interface one is camera independent.
Table 5.2 summarises the design of the scaling techniques for all interfaces while figure 5.10 illustrates the different techniques.

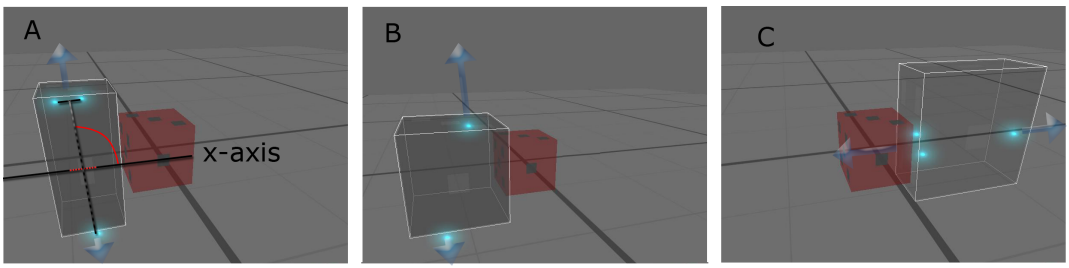

Figure 5.10: Illustration of our scaling techniques: (A) shows how the scaling axis is chosen. The angle between touch points and the local object x-axis derives the scaling axis. If the angle is smaller than 45 degrees, the other axis is chosen (C). In (B) the global scaling technique is shown.

Interface	Touch Count	Movement Type	Type of Translation
1	2	RELATIVE	Scales a single object axis in the most perpendicular object plane to the camera view direction.
2,3	2+1	(GROUPED)× RELATIVE × (GROUPED)	
1	2+1	(GROUPED)× RELATIVE × (GROUPED)	Scales along the object axis most parallel to the camera view direction.
1	3+3	(GROUPED)× RELATIVE × (GROUPED)	Global scale
2,3	2	RELATIVE	

Table 5.2: Scaling techniques available in our three interface prototypes

Implementation

Listing 5.5 provides a pseudo implementation of our scaling technique taken from interface one, which is the only interface to not incorporate a rotational component within this technique (see section 5.2.4).

```
void InterfaceName::scaleObject(Guess* guess){
  globalScale = false;
  TranslationAxis currentAxis = extractAxisFromGuesss();

  //scaleIDs consists of up to 6 touch ids
  mapGuessIDsToInternalIDs(guess,scaleIDs);
  if(currentAxis= AXIS_FREE){
    globalScale=true;
  }
  if(globalScale){//make sure retapping works as intended
    if(internalIDhaveChanged("1-6"){
    lastDistance = 0.0;
    }
  }else{
    if(internalIDhaveChanged("1-2")){
      lastDistance = 0.0;
      cameraRelScaleAxis = getPerpendicularPlane(currentlySelectedObject, cam);
      if(internalIDhaveChanges("3")){
        cameraRelScaleAxis = getParallelPlane(currentlySelectedObject, cam,);
      }
    }
  }
  if(currentAxis == AXIS_XY){
      //calc Distance between 1 & 2
      curDistance = distanceBetween(id1, id2);
    }
  if(currentAxis == AXIS_Z){
      curDistance = distanceBetween(id1, id3);
```

```
  }
  if(currentAxis == AXIS_FREE){
      curDistance = distanceBetween(id1, id6);
  }
  if(lastDistance != 0.0){
    ratio = curDistance / lastDistance;
  }
  if(currentAxis == AXIS_XY){
    scaleObjectPerpendicular(id1, id2, cameraRelScaleAxis);
  }else if(currentAxis == AXIS_Z){
    scaleObjectParallel(id1, id2, cameraRelScaleAxis);
  }else if(currentAxis == AXIS_FREE){
    currentlySelectedObject->scale(Vector3(ratio, ratio, ratio));
  }
    lastDistance = curDistance;
    lastScaleIDs = scaleIDs;
  }
}
void InterfaceName::scaleObjectCameraDependant(int id1, int id2, axis){
  //TranslationAxis can represent a plane or an axis
  TranslationAxis curAxis = (TranslationAxis) cAxis;

  if(id1 != lastScaleId1 || id2 != lastScaleId2){
    lastCameraDistance = 0.0;
  }
  if(curAxis == AXIS_YZ || curAxis == AXIS_XY || curAxis == AXIS_ZX){
    curDistance= distanceBetween(id1,id2);
  }
  if(lastCameraDistance != 0.0){
    ratio = curDistance / lastCameraDistance;

    if(curAxis == AXIS_XY){
      //Check if facing front or back of the object:
      Vector3 camZAxis = cam->getOrientation().zAxis();
      Vector3 objZAxis = currentlySelectedObject->_getDerivedOrientation().zAxis();
      float lookAngle = camZAxis.angleBetween(objZAxis).valueDegrees();
      bool recalcAngles = false;
      if(lookAngle > 90){
        recalcAngles = true;
      }
      //determine object roll on active plane in relation to camera up vector
      Quaternion  objori = currentlySelectedObject->_getDerivedOrientation();
      Vector3 objYAxis = objori.yAxis();
      Vector3 camUp = cam->getDerivedUp();
      objYAxis.normalise();
      camUp.normalise();
      Vector3 vp = objYAxis.crossProduct(camUp);
      vp = vp.crossProduct(objYAxis);

      Vector3 objXAxis = objori.xAxis();
      float tmp = objXAxis.angleBetween(camUp).valueDegrees();
      objroll = -sign(camUp.dotProduct(vp))*
                        atan(objYAxis.dotProduct(camUp));
```

```
      tmp = tmp* sign(objroll);
      objroll = tmp;
      if(recalcAngles){
        objroll = objroll*-1;
      }
      if(objroll <0){
        objroll +=360;
      }

      //angle between touches and screen-up in 360 degrees
      vec3f touchdir = currentTouches[id1].current.position-
                  currentTouches[id2].current.position;
      Vector2 screenTouchDir = Vector2(touchdir.x, touchdir.y);
      screenTouchDir.normalise();

      float angleScreenUpTouches = atan2(screenTouchDir.x,screenTouchDir.y);
      angleScreenUpTouches = Radian(angleScreenUpTouches).valueDegrees()*-1;
      angleScreenUpTouches += -90;
      if(angleScreenUpTouches <0){
        angleScreenUpTouches = 360+angleScreenUpTouches;      }

      float touchToGlobalObjectRollAngle = abs(angleScreenUpTouches-objroll);

      if((touchToGlobalObjectRollAngle > 45 && touchToGlobalObjectRollAngle < 135) ||
         (touchToGlobalObjectRollAngle > 225 && touchToGlobalObjectRollAngle < 310)){

        curAxis = AXIS_X;
      }else {
        curAxis = AXIS_Y;
      }
    }else if(curAxis == AXIS_YZ){
      //ANALOGUE
    }else if(curAxis == AXIS_ZX){
      //ANALOGUE
    }
  }
  if(curAxis == AXIS_X){
    currentlySelectedObject->scale(Vector3(ratio,1,1));

  }else if(curAxis == AXIS_Y){
    currentlySelectedObject->scale(Vector3(1,ratio,1));

  }else if(curAxis == AXIS_Z){
    currentlySelectedObject->scale(Vector3(1,1,ratio));
  }

  lastCameraDistance = curDistance;
  lastScaleId1 = id1;
  lastScaleId2 = id2;
}
```

Listing 5.5: Pseudo Implementation of Object Scaling Technique.

5.2.4 Object Rotation Techniques

All interfaces offer at least one techniques for object rotation. With three touches moving GROUPED users can execute an ArcBall [23] rotation technique to rotate the object around its centre in each interface. Although only one input point would be needed for the ArcBall technique, we could not use one touch movement for the ArcBall rotation because it is mapped to object translation. Using three instead of one touch though might reduce usability because the user has to derive the actual point on the ArcBall from the three touches he used to perform the technique with. The point on the ArcBall is derived from the gravity centre of the triangle the three touches form. This way the user is not forced to remember a certain finger arrangement to perform the technique.
Although it is possible to control rotational DoFs separately with the ArcBall by touching outside the object, this concept might not be obvious to the users [23]. Thus interface two and three provide one additional rotation technique allowing rotation around the object axis which is most parallel to the camera's viewing direction with two touches. However, we mapped two touch movement of the RELATIVE type to scaling in each interface and two touch GROUPED movement to translation in interface one. Thus it is impossible to provide another separated two-touch technique for rotation by means of our gesture recognition system in interface one.

Therefore we adapt the two touch scaling techniques of interface two and three to

Interface	Touch Count	Movement Type	Type of Translation
1,2,3	3	GROUPED	ArcBall rotation.
2,3	2	RELATIVE	Rotation around the object axis most parallel to the cameras viewing direction
3	1+1	(MOVING)× RELATIVE × (RESTING)	Simultaneous translation and rotation around the point specified by the RESTING touch.

Table 5.3: Rotation techniques available in the three interface prototypes

allow scaling and rotation, while interface one will only support ArcBall rotation.
Moreover, the two finger rotate+scale technique performs differently in interface two and three. By using two MOVING touches anywhere on the screen, but with the objects centre in between, users can scale globally and at the same time rotate around the object axis most parallel to the camera view direction in interface three. Interface

two, however, separates these two tasks. Scaling of the object is denied, when it has been rotated a few degrees already. This way the user can choose to either rotate or scale with two touches, but needs to retouch the screen to switch tasks. This allows a brief separation of both manipulation modes without requiring another interaction technique.
Furthermore, if using one MOVING touch and a second RESTING touch somewhere on the screen the user may translate and rotate the object around the point specified by the resting touch in interface three. That point lies above or below the most perpendicular plane to the cameras viewing direction to account for displacement of the object along the normal of that plane. See figure 5.11, 5.12 and 5.13 for an illustration of the three object rotation techniques and table 5.3 for a summary of the input mapping to rotation tasks.

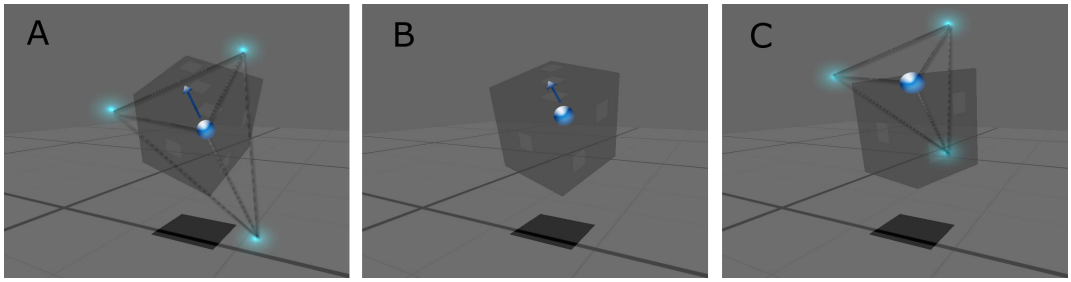

Figure 5.11: Arcball rotation: The gravity point of the triangle formed by the three touch points act as the ArcBall's active input point. Through motion of the touches this points translates to its destined position

Implementation

The rotational components of the rotate+scale technique can be reviewed in listing 5.6 including the separation of both tasks. A reference for the implementation of the ArcBall can be found in [37].

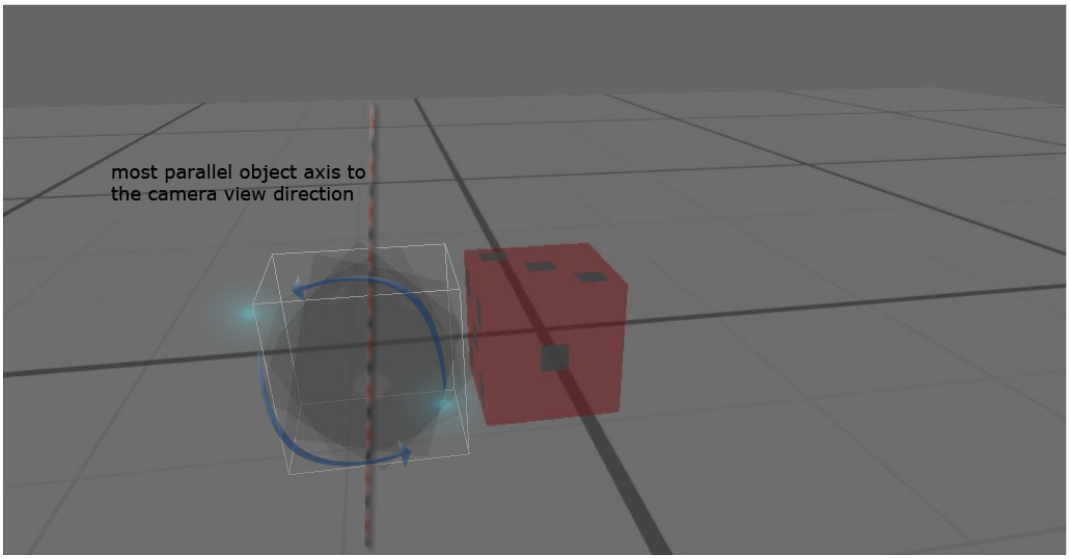

Figure 5.12: Axis restricted rotation rotates the object around the most parallel local axis in relation to the camera's viewing direction

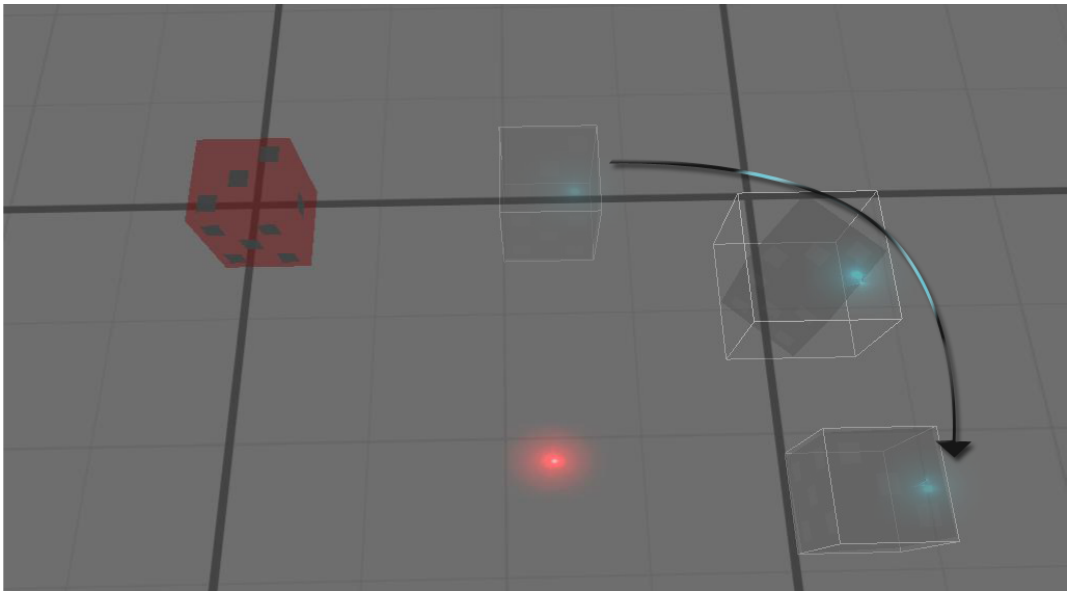

Figure 5.13: Rotation and Translation around a point specified by the resting touch (red).

```
void InterfaceName::rotateScaleObject(Guess* guess){

    if(guess->metaData.currentUpdates.size() == 2){
      rotateScaleId1 = guess->metaData.currentUpdates.front();
      rotateScaleId2 = guess->metaData.currentUpdates.back();

      if(rotateScaleIDs != oldRotateScaleIDs ){
```

```
        Vector2 touchpos =currentTouches[rotateScaleId1].current.position;

        withoutScale = false;
        rotatethottle = 15;
        currentAxis = getPerpendicularPlane(currentlySelectedObject,
                                cam,scalerotateRel);
        //curObjectPos is global and modified by this method
        getScreenspaceCoords(currentlySelectedObject, cam ,curObjectpos);
        }else{
        if(!currentTouches[rotateScaleIDs].tag.empty()){
          Vector2 lastPos = currentTouches[rotateScaleId1].tag[0].position;
          Vector2 curPos = currentTouches[rotateScaleId1].current.position;
          Vector2 lastd, curd;
          lastd =  curObjectpos - lastPos;
          curd = curObjectpos - curPos;
          lastd.normalise();
          curd.normalise();
          float angleBetween = atan2(lastd.y,lastd.x) - atan2(curd.y,curd.x);

          vec3f lastTouchDir = vec3f::zero;
          if(currentTouches[rotateScaleIDs].tag.size() >0){
            lastTouchDir = currentTouches[rotateScaleId1].tag[0].position -
                       currentTouches[rotateScaleId2].tag[0].position;
          }
          float lastAngleScreenUpTouches = atan2(lastTouchDir.x, lastTouchDir.y);
          lastAngleScreenUpTouches =
                    Radian(lastAngleScreenUpTouches).valueDegrees()*-1;
          lastAngleScreenUpTouches += -90;
          if(lastAngleScreenUpTouches <0){
            lastAngleScreenUpTouches = 360+lastAngleScreenUpTouches;
          }
          vec3f currentTouchDir = currentTouches[rotateScaleId1].current.position -
                         currentTouches[rotateScaleId2].current.position;
          float currentAngleScreenUpTouches =
                         atan2(currentTouchDir.x,currentTouchDir.y);
          currentAngleScreenUpTouches =
                    Radian(currentAngleScreenUpTouches).valueDegrees()*-1;
          currentAngleScreenUpTouches += -90;
          if(currentAngleScreenUpTouches <0){
            currentAngleScreenUpTouches = 360+currentAngleScreenUpTouches;
          }
          //diff current & last Angle to thottle
          rotatethottle -= lastAngleScreenUpTouches- currentAngleScreenUpTouches;
          //withoutScale is global and default false
          if(withoutScale == false){
            if(rotatethottle < 0 || rotatethottle > 30){
              withoutScale = true;
            }
          }
          if(TranslationAxis(currentAxis) == AXIS_XY){
            if(!withoutScale){
              // @par translationAxis = 0 = GlobalScale
              scaleObject(guess, 0 );
```

```
            }else{
              currentlySelectedObject->
              rotate(Vector3::UNIT_Z,Radian(scalerotateRel*(angleBetween)));
            }
          }else if(TranslationAxis(currentAxis) == AXIS_YZ){
            if(!withoutScale){
              scaleObject(guess, 0);
            }else{
              currentlySelectedObject->
              rotate(Vector3::UNIT_X,Radian(scalerotateRel*(angleBetween)));
            }
          }else if(TranslationAxis(currentAxis) == AXIS_ZX){
            if(!withoutScale){
              scaleObject(guess, 0);
            }else{
              currentlySelectedObject->
              rotate(Vector3::UNIT_Y,Radian(scalerotateRel*(angleBetween)));
            }
          }
          lastScaleRotateAngle = angleBetween;
        }
      }
      oldRotateScaleIDs = rotateScaleIDs;
    }
}
```

Listing 5.6: Implementation of our Rotate+Scale technique

5.2.5 Selection Techniques

Although touch mapping for all interfaces supports recognition of all gestures regardless of scene content, we still require scene context information for some tasks to perform properly or allow separation of its DoF. Thus we require an object to be selected, before it can be manipulated. As soon as there are multiple objects in the scene, it is not possible to decide to which object the action should be applied without detecting whether a certain object was touched or not. Moreover, complex 3D models consist of child-components - a robot, his arm, and his hand for example. All components should be modifiable without changing properties of parental components. Without a selection technique the user would need to perform more camera navigation tasks to get the object into manipulation range. With a selection technique however, the user could select a small child-component and perform the task without needing to navigate the camera in order to increase size of that object.
A common selection technique is the double click or tap with a touch. For selecting multiple objects often the rubber-band technique is used. Instead of using these well known techniques we allow the user to select a single or multiple objects with the

same technique. But since two touch movement is already mapped to object scaling we cannot use a two-touch rubber-band selection technique. By putting three touches on the screen and not removing them until they reach the RESTING state users can select all objects inside the triangle formed by those touches.
However, since the evaluating experiment will involve manipulation of a single object only, multiple selection functionality is not fully implemented, yet. Moreover, using four points instead of three would double the selection surface area available to the user. Figure 5.14 illustrates our selection technique. Since this technique may be performed bi- manually it is likely that the touch points' disappearance is recognised with a slight delay. This is the main reason why we keep track of these touches (see. section 5.1).

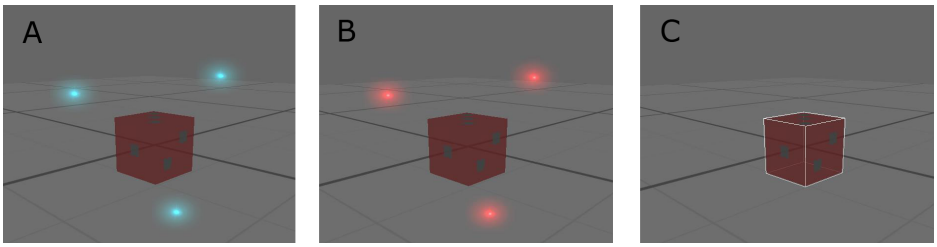

Figure 5.14: Object Selection: (A) Three touches appear forming a triangle. (B) the user waits until the touches reach the RESTING state. (C) The fingers are removed from the surface and the object inside the triangle is selected.

6 Usability Evaluation

To gain insights on the usability of the proposed user interfaces an evaluation was conducted. In order to compare usability of the interfaces it would be reasonable to let each participant perform a task trial with two or even three interfaces, so that subjective user preferences could be compared directly.
Considering the amount of possible manipulations to be tested, the need for repeated measurements to gain reliable results, and the necessary time investment from the participants, we decided to provide only one interface per participant. This however, requires us to carefully assign the participants to our interfaces in order to create a uniform distribution of participants abilities over all interfaces, so that statistical comparisons may be reliable.

6.1 Experiment Design

The experiment was set up as shown in figure 6.1. The screen projected onto a wall did not extend to the projectors maximum capabilities. To simulate a desktop sized screen the extend of the projected area was limited to 85 cm diagonally. The participants could choose to sit or stand in front of the screen to account for height differences. The screens bottom edge was 95cm above ground level. A camera with an infrared lens delivered touch input capture at 25 fps at a resolution of 1024*786 pixels resulting in a pixel density of about 15 pixel per centimetre. Since the application's resolution was higher than the touch captures resolution (1920*1080) pixel accurate input is not possible in our setup due to rounding input coordinates. Our prototype as well as the capturing software were executed from the same computer - an Intel Core2Quad Q6600 @ 2.4 Ghz with 4GB of RAM and an Nvidia GeForce GTX285. To receive best possible results the touch capturing software community core vision[1] was reconfigured for every participant to account for different hand sizes and forces applied to the screen during interaction.

[1] http://ccv.nuigroup.com/

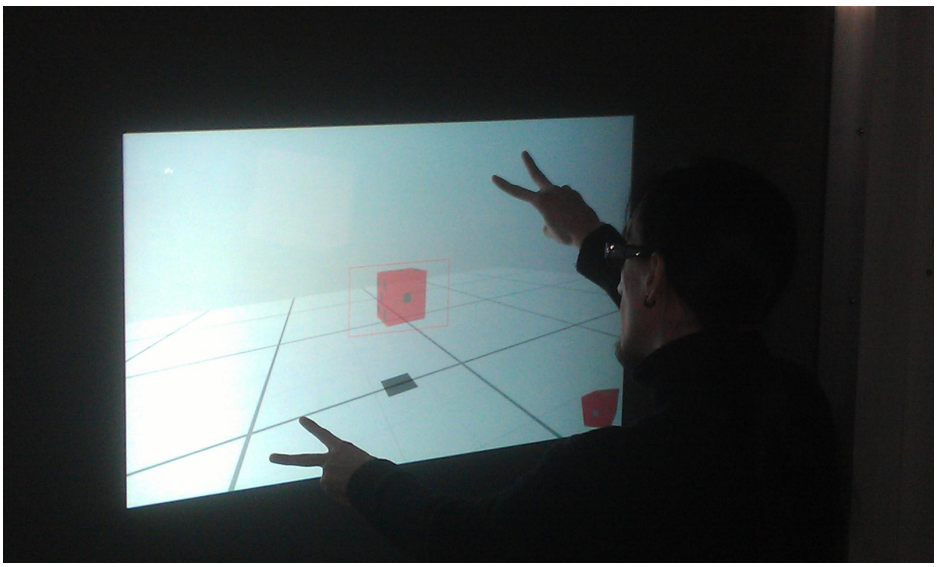

Figure 6.1: Experiment Setting

As an initial step participants had to fill out the first part of a questionnaire to identify their general computer skills and experience with multi-touch technology, so that he/she could be assigned to an interface group.
Each participant was asked to perform the same task-trial in the experiment. The trial was divided into two blocks. In the first block the subjects had to navigate the camera in a way that the only object in the scene, a six-eyed-dice, would be positioned in the middle of the screen with a designated side facing the camera. Additionally the participant should align the cameras up vector with the objects local y-axis. Figure 6.2 shows the initial and final states of a camera navigation task. The reference view on the lower right side of the screen provides assistance with rotations, if the subject has problems identifying eye-count of the hidden dice faces. Before the trial started participants were introduced to the camera navigation, but not the object manipulation techniques moreover, they did not try the techniques for themselves. After the demonstration of the techniques the first block of the trial was initiated. This camera navigation block is tiered into a training and a main section. The initial training consists of nine easy to medium difficult tasks. The main section follows with 29 tasks with gradually increasing difficulty in respect to the magnitude of navigation required to solve the task.

Since perception of the criterion: "bring the object to the middle of the screen" in the camera trial tend to be very subjective, a red 2D rectangle is drawn on the view port - not in the scene, to give the participants a reference to align the object to (see fig. 6.2). However, each task was finished by the participant manually, if he/she

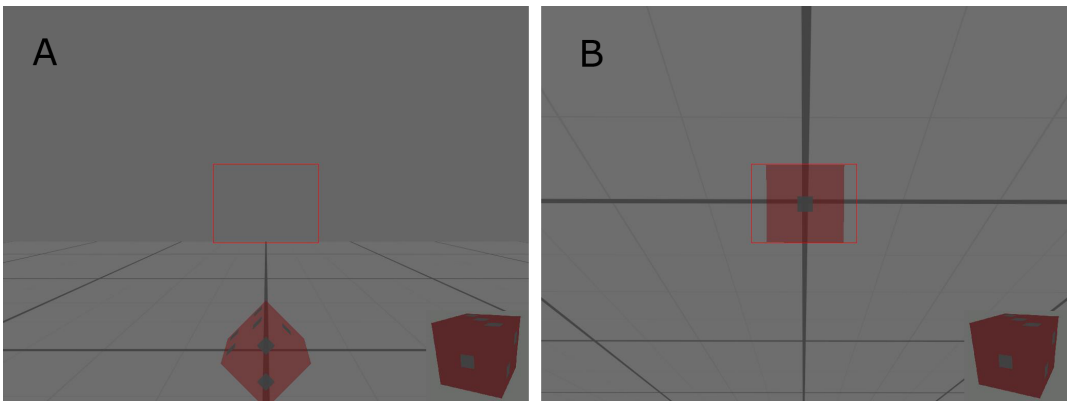

Figure 6.2: Camera Navigation Task: In (A) the initial state of a task is shown. To be able to solve the task the user has to e.g. orbit around the object in order to identify the correct alignment with the object. Having identified what to do, the user performs navigation tasks to reach the just formulated goal as shown in (B).

found the current state acceptable. Moreover, each task was given a maximum of two minutes to be completed, independent of its difficulty. If the time limit was reached, the application did not accept any further input thus the subject had to skip to the next task by pressing a button.
After completing the camera navigation tasks, the participants were introduced to the remaining object manipulation techniques. The second block of the trial again begins with a training section consisting of 9 trials and a main phase with 29 trials. The participants had to carry out a docking task in this block of the trial.
In this phase two objects are shown in the scene. One immovable and not selectable object representing the target condition and another selectable object which the subject has to adapt to the target condition in regard to size, position and orientation. Figure 6.3 shows an exemplary docking task's initial and near final state. Similar to the first block participants were allowed to finish a task, if they were satisfied with the current state. After finishing the object manipulation trial, the participants filled out the second part of the questionnaire and were asked for their personal opinion on the topic of multi-touch 3D interaction or to comment their experience of the trial. Participants were allowed to take a break anytime during the trial.

6.2 Participants

15 participants, one of them female with average age of 27,93 years (23-36, SD: 3,71) where assigned to three groups of five people. Skill and experience of participants in

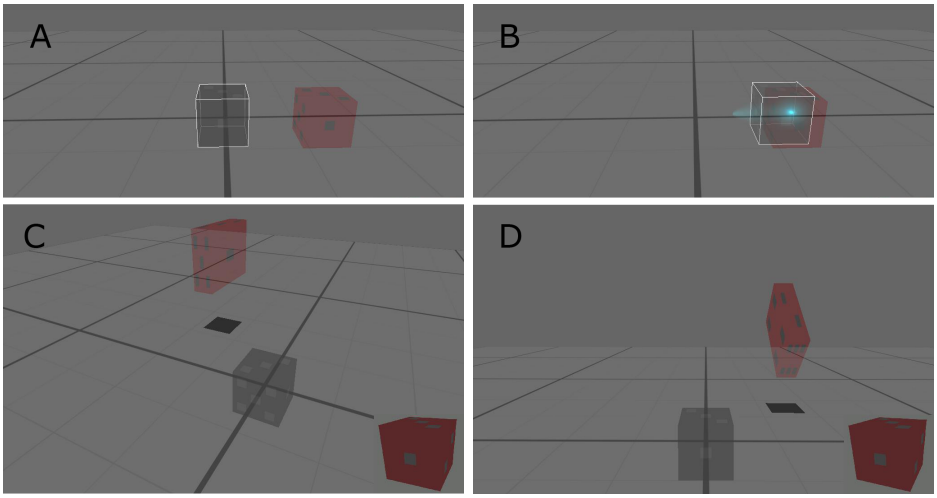

Figure 6.3: Object Docking Task: (A) shows the initial and (B) the near final state of an "easy" difficulty task. Here only translation along the y-axis is required to solve the task. In (C) object scaling and translation is needed to solve the task while in (D) the manipulation of all object properties is required.

each test group regarding touch screens and 3D applications vary greatly. The participants are distributed over the test groups according to their self-reported abilities, so that a uniform distributed set can be assumed. Means of the self-reported skills are shown in table 6.1 for each group in 5 point Likert scale (1 = Low Experience - 5 = High Experience). The second and third column however, show the portion of participants who could answer the referring question with "yes". Thus for each group three of five participants have taken part in an experiment involving touch-screens before.

Interface Group	*1	*2	Ex. w. 3D App.	Ex. w. 3D CAD	Ex. w. 3D Games
1	3	5	3.8(MD:4,SD:1.64)	3(MD:4,SD:1.87)	4(MD:5,SD:1.73)
2	3	4	3.4(MD:4,SD:1.34)	2.4(MD:2,SD:1.14)	4(MD:4,SD:1)
3	3	5	3.8(MD:4,SD:1.30)	2.8(MD:2,SD:1.30)	4(MD:4,SD:1)

Table 6.1: Distribution of user reported abilities for each interface group.
(Ex. w. = Experience with)
*1) Number of participants which have taken part in an experiment involving touch screens before.
*2) Number of participants which have worked with a touch screen before.

Subjects in each group have overall advanced skills with 3D games and 3D applications (median of 4) and slightly less experience with 3D CAD/Design applications. Since skills data from all participants was not available before the first trial, the participants were assigned to their groups consecutively, putting strong candidates in the weakest group and vice verse. However, there were more experienced 3D CAD users

in the second half of the trial than in the beginning resulting in a slightly unbalanced value for test group one in the "Experience with 3D CAD" column.

6.3 Materials & Methods

To evaluate which of our interface prototypes delivers the best results in terms of usability we have taken the following properties into consideration:

Task completion time Is there a relation between task completion time and interface

Precision Are there differences in error rates between the different techniques / interfaces

Ease-of-use Although participants are not able to compare their experiences during the trial with another interface, their subjective comments could provide an impression of what went well and what didn't.

Before any statistical approaches are made on the data, tasks which did not finish before the 120 second threshold or which were not solved correctly were filtered from the trials' output data. Overall there were 870 main and 270 training tasks completed by the 15 participants. That is 435 main tasks and 135 training tasks per block. After identifying faulty cases we reduced the task count to 109(-19,3%) camera training tasks and 101(-25,2%) object training tasks. The main task count was reduced to 417(-4,1%) cases for the camera block and 298(-31,5%) for the object block.

6.4 Trial Block 1 Results - Camera Navigation

6.4.1 Task Completion Time

Assuming that subject's skills are uniformly distributed between the groups we expect no significant differences in camera task completion times between all three groups, because they all use the same navigation techniques.
Exploratory data analysis (EDA) (see 6.4) reveals a mean completion time of 29.04 sec. per task(SE:1.42, SD:16.89, V:285.316) for the first group which was assigned to interface one. The second interface group's data is 28.23 sec. (SE:1.37, SD:16.4, V:269.12) and 27.31 sec. (SE:1.66, SD:19, V:360.96) for the third group[2].

[2] SE:Standard error, SD: Standard deviation, V: Variance squared

A one-way ANOVA was performed on the mean task completion time of every participant for each interface. The result does not allow us to reject the main hypothesis of the mean completion time being equal, because no significant differences between any of the groups could be found, (F(2:12)=0.041; P=0,960) while the homogeneity of variances criterion was met(P=0.31).

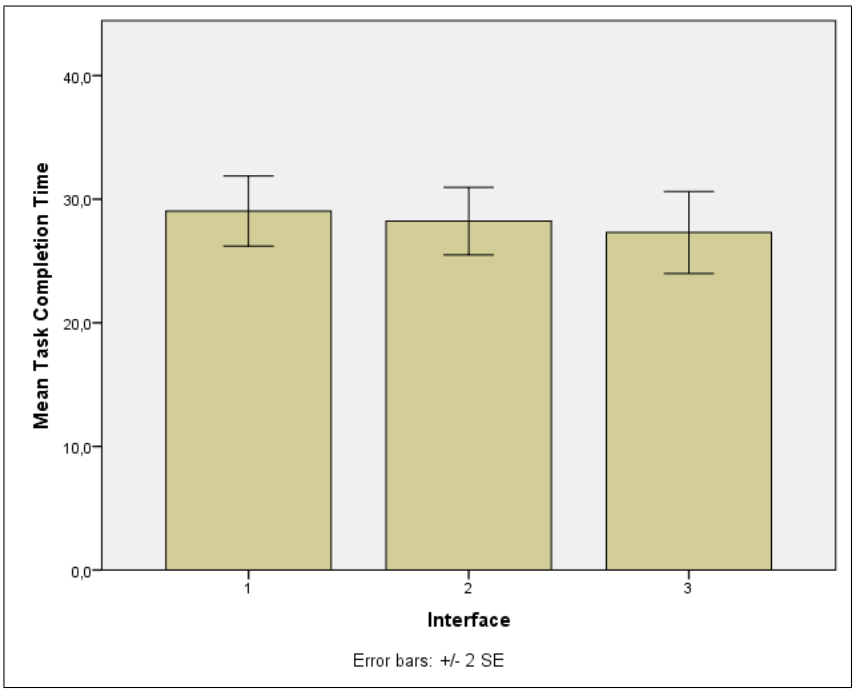

Figure 6.4: Similar completion times for the camera navigation tasks in the main block.

6.4.2 Precision

Only translation and rotation errors need to be taken into consideration when analysing accuracy of the subjects since the camera-zoom error is a translation error. The translation error was measured in scene units. Besides the translation error for each DoF we build a combined score of all three errors with $\sqrt{errorX^2 + errorY^2 + errorZ^2}$. The mean values for the x-axis translation error is 15.08(SE:1.16, SD:13.77, V:189.74) for interface one, 11.81(SE:0.97, SD:11.63, V:135.29) for interface two and 11.73(SE:1.11, SD:12.76, V:162.86) for interface three. For the y-axis error EDA returned a mean value of 10.7(SE:0.99, SD:11.74, V:137.82) for interface one, 11.8(SE:1.09, SD:13.02, V: 169,7) for interface two and 8.96(SE:1.05, SD:12.07, V: 145,58) for interface three, while for the z-axis error means of 23.76(SE:1.27, SD: 15.19, V: 230.63), 16.24(SE:1.04, SD:12.46, V: 155.24) and 19.04(SE:1.34, SD: 15.29, V: 233.75) were given for interface one to three.

The combined metric has a mean error of 34.46(SE:1.39, SD:16.57, V: 274.6) for interface one, 27.88(SE:1.25, SD:14.97, V: 224.17) for interface two and 28.45(SE:1.54, SD: 11.64, V: 311.18) for interface three.
For the rotational errors we found a mean of 5.22 degrees (SE:0.36, SD:4.33, V:18.78) for interface one in the combined score, while the second interface has 6.38(SE:0.44, SD: 5.32, V: 28.3) and the third 4.71(SE:0.41, SD:4.64, V:21.49). The mean errors for each rotational DoF are 2.68(SE:0.21, SD:2.52, V:6.34), 3.2(SE:0.26, SD:3.08, V:9.47) and 2.22(SE:0.22, SD:2.5, V:6.25) for interface one to three in the x-axis condition. Respective 2.42(SE:0.18, SD: 2.18, V:4.77), 2,98(SE:0.23, SD:2.76, V: 7.62) and 2.28(SE:0.22, SD:2.47, V:6.09) for the y-axis condition and 3.32(SE:0.28, SD:3.31 V:10.97), 4.1(SE:0.33, SD:4, V: 16,02), 3.04(SE:0.3, SD:3.46, V:11.98) for the z-axis condition.
A graphical representation of all different translation errors can be reviewed in figure 6.5 while the rotation errors can be found at figure 6.6.

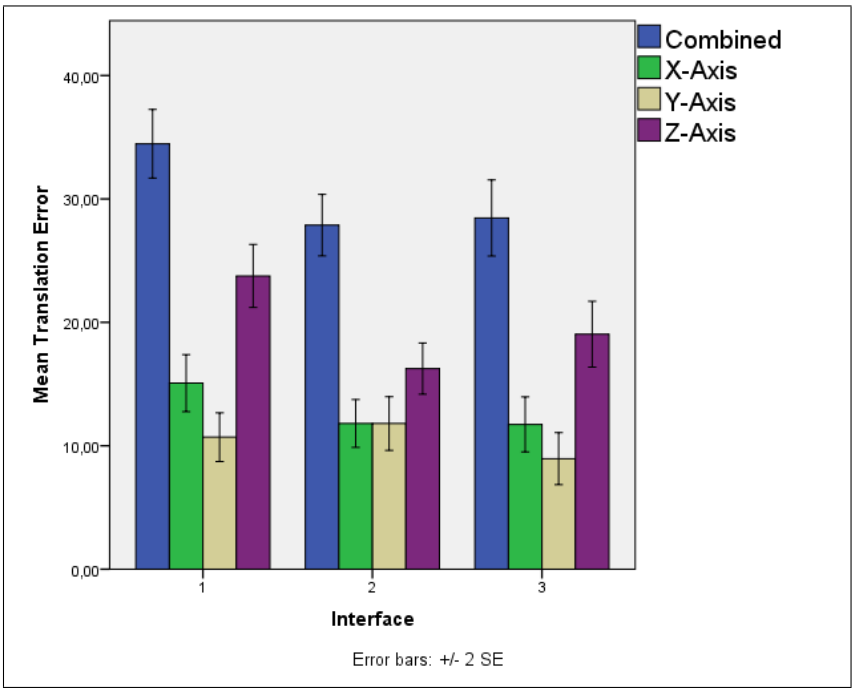

Figure 6.5: Different translation errors and its combined score.

We expect the interfaces to have no significant differences in mean errors for any property. However, the z-axis error is in general higher, because the camera has to move more scene units to the front or back in reference to the other dimensions, so that a difference between the alignment frame (see fig.6.2) and the objects extends of at least one pixel can be spotted.
ANOVA was performed between the interface groups on the combined mean trans-

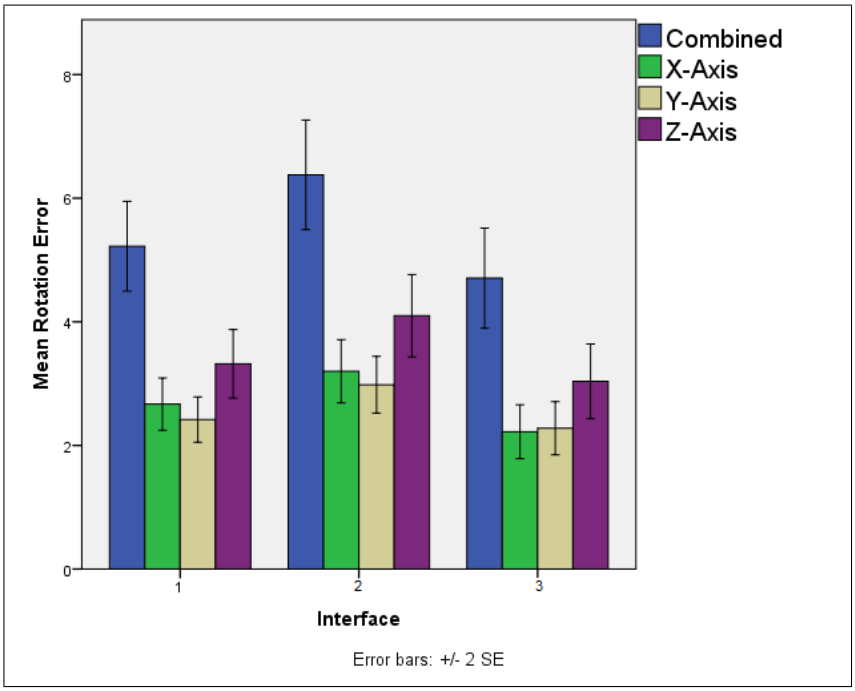

Figure 6.6: Rotational errors and its combined score.(errors in degree).

lation error and revealed no significant differences (F(2:12)=0.65; P=0.539). Also for the rotational errors no significant differences were found by a one-way ANOVA test on the combined score (F(2:12)=2.764; P=0.103). Homogeneity of variances was met in both cases (P=0.776, P=0.282).

6.5 Trial Block 2 Results - Object Manipulation

6.5.1 Task Completion Time

Unfortunately there were no statistically significant differences for the task completion times between groups in the second block of the trial as determined by one-way ANOVA (F(2;12)=0,32 P= 0,732). EDA revealed average completion times of 51,74(SE:2.91, SD:30.53, V:932,31) for the first, 56,82(SE:3.17, SD:31.34, V:982.05) for the second and 52.54(SE:3.34, SD: 31.7, V:1005,1) for the third group (see fig.6.7).

6.5.2 Precision

Although completion times of the groups had no significant difference this does not necessarily apply to the error rates, too. Figure 6.8 summarises the results of EDA

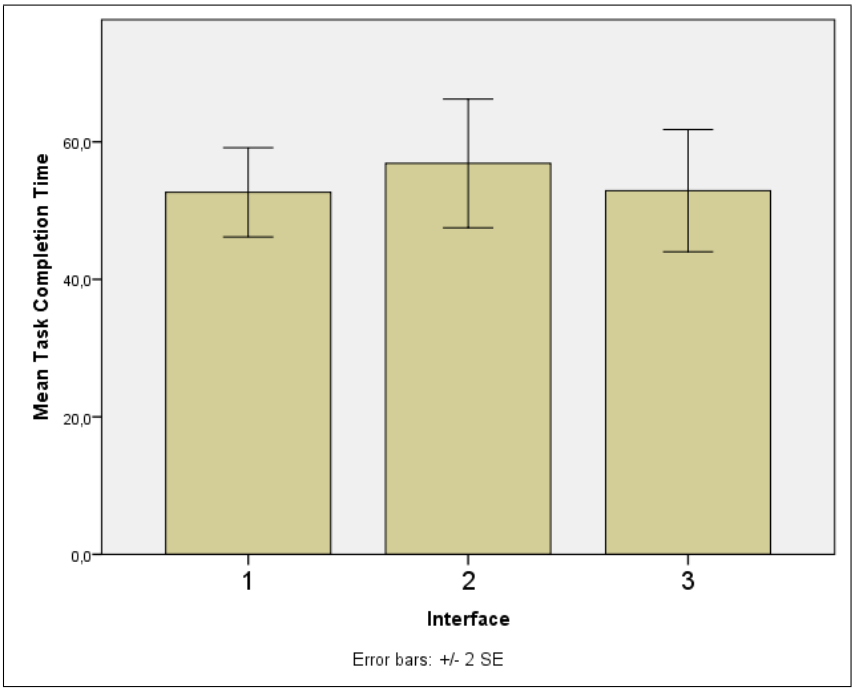

Figure 6.7: Illustration of mean completion times in the object manipulation block.

on all translation errors and its combined score. For the combined condition mean translation errors of 4.11(SE:0.52, SD:5.41, V:29,29) for the first, 4.28(SE:0.75, SD: 7.46, V:55.72) for the second and 5.35(SE:0.7, SD:6.62, V:43.86) for the third are given. This relates to x-axis mean translation errors of 1.06(SE:0.11, SD:1.17, V:1.38), 1.76(SE:0.28, SD:2.74, V:7.51) and 2.86(SE:0.51, SD:4.79, V:22.99) for interface one to three. The mean translation errors for the y- and z-axis are 2.3(SE:0.39, SD:4.05, V:16.43) and 2.3(SE:0.39, SD:4.1, V:16.81) for group one. Y-axis error for group two is 2.82(SE:0.7, SD:6.95, V:48.34) and 2.22(SE:0.29, SD:2.79, V:7.78) for group three. The z-axis translation errors for group two and three are 1.33(SE:0.24, SD:2.34, V:5.49) and 2.59(SE:0.49, SD:4.69, V:22.02).

We performed multiple one-way ANOVA tests between all translation error conditions, however, no significant differences between groups could be found.
The combined, the y- and z-axis values meet the homogeneity of variances criterion (P=0.154, P=0.167, P=0.112) while the x-axis value does not($P=0.028 < 0.05$). Thus we have to be very careful interpreting the results of the ANOVA tests. The main results of the test state no significant differences between groups for all error condition [Combined: ($F(2:12)=0.154$; $P=0.859$), X-Axis: ($F(2:12)=2.715$; $P=0.106$), Y-Axis: ($F(2:12)=0.176$; $P=0.841$), Z-Axis: ($F(2:12)=1.704$; $P=0.223$)]. This correlates with the results of the Welch and Brown-Forsythe tests, which do not suggest to reject the null

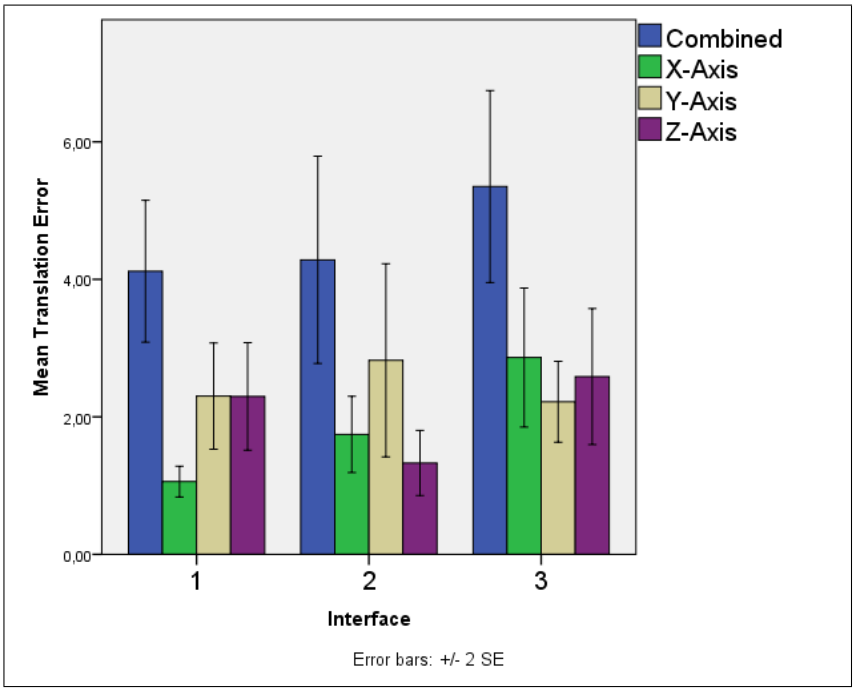

Figure 6.8: Mean object translation errors of all groups / interfaces.

hypothesis either.

Figure 6.9 represents the results of the EDA on the different rotational errors and its combined score. For the combined error we found a mean value of 3.93(SE:0.46, SD:4.87, V:23.75) for interface one, while interface two and three show a mean error of 3(SE:0.39, SD:3.9, V:15.2) and 4.65(SE:0.67, SD:6.46, V:41.79). The means for the x-axis rotational error for interface one to three are 2.2(SE:0.29, SD:3, V:9.04), 1.54(SE:0.21, SD:2.09, V:4,37) and 2.59(SE:0.39, SD:3.7, V:13.66). For the y-axis we find 2.02(SE:2.51, SD:2.64, V:6.98), 1.76(SE:0.25, SD:2.45, V:6) and 2.84(SE:0.46, SD:4.32, V:18.67). Z-axis means for interfaces one to three further are 2.23(SE:0.29, SD:3.05, V:9.33), 1.62(SE:2.42, SD:2.39, V:5.72) and 2.17(SE:0.36, SD:3.4, V:11.54).

Again we performed multiple one-way ANOVA tests between all error conditions. This time the homogeneity of variances criterion cannot be met for any of our test variables. (Combined: P=0.02, X:P=0.001, Y:P=0.003, Z:P=0.019). However, the results of the ANOVA tests indicates no significant differences between any group [Combined: (F(2:12)=1.237; P=0.325), X-Axis: (F(2:12)=1.688; P=0.226), Y-Axis: (F(2:12)=1.222; P=0.329), Z-Axis: (F(2:12)=1.278; P=0.314)]. Moreover, we did not find any significant differences with the Welch and Brown-Forsythe tests. Thus we deduce no significant

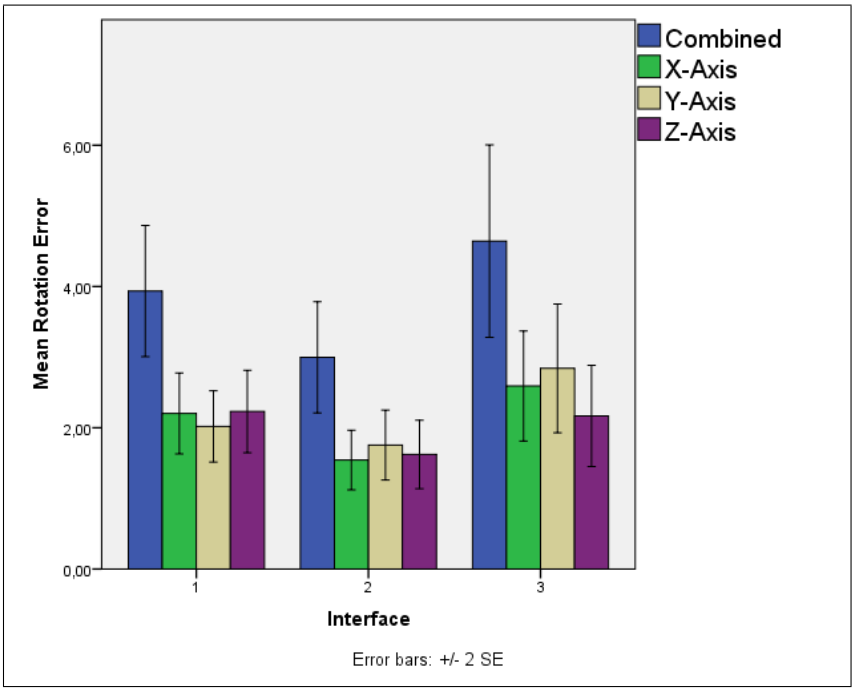

Figure 6.9: Mean object rotation errors for all groups / interfaces

differences between groups for the two error types.

The scale error will be calculated as the relative difference to the optimal object size in percent. Thus a scale task where the object is half the size of the target object in a certain dimension will have an error of 50%. For this combined score this relates to a mean error of 7.38(SE:0.72, SD:7.52, V:56.61) for interface one, 6.9(SE:0.76, SD:7.48, V:55.94) for interface two and 7.45(SE:0.88, SD: 8.34, V:70.03) for interface three. Mean errors for the x-axis condition of interfaces one to three are 2.25(SE:0.39, SD:4.11, V:16.86), 3.17(SE:0.43, SD:4.3, V:18.45) and 3.53(SE:0.53, SD:5.02, V: 25.19), while for the y-axis means of 4.2(SE:0.64, SD:6.75, V:45.58), 3.61(SE:0.46, SD:4.54, V:20.62) and 4.29(SE:0.58, SD:5.54, V:30.69) were evaluated. The remaining z-axis mean errors are 3.26(SE:0.38, SD:3.95, V:15.59), 3.29(SE:0.56, SD:5.54, V:30.67) and 3.55(SE:0.54, SD:5.13, V:26.32). These values are summarised in figure 6.10.

Again ANOVA revealed no significant differences between the three groups for any of the error parameters. In each case the homogeneity of variances criterion was met, so the results of the ANOVA are quite reliable [Combined:(F(2:12)=0.55; P=0.946), X:(F(2:12)=2.322; P=0.14), Y:(F:(2:12)=0.308; P=0.741), Z:(F:(2:12)=0.191; P=0.829)].

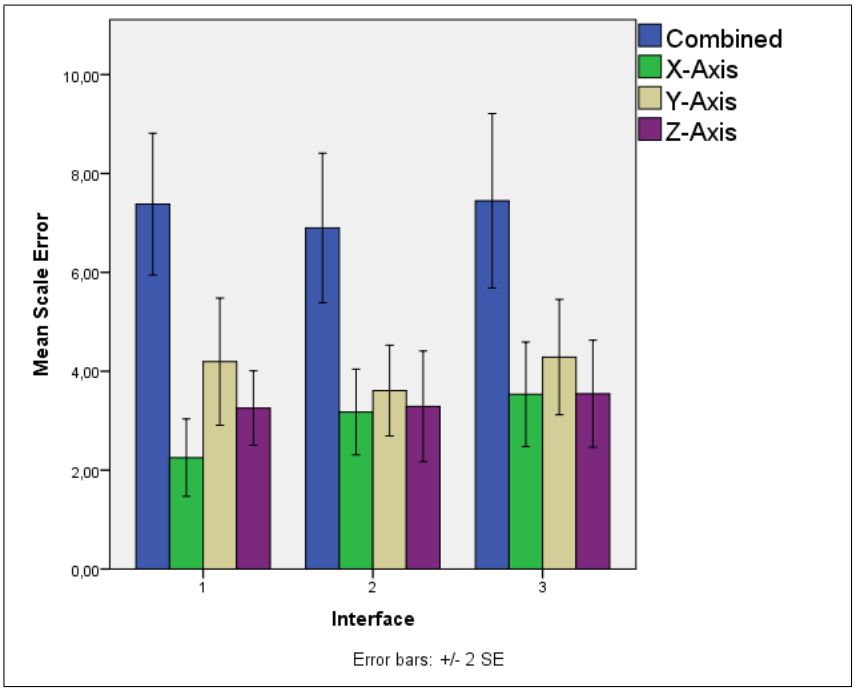

Figure 6.10: Mean object scale errors for all groups / interfaces

6.6 Discussion

As expected no significance in average task completion time in the first block was shown by the ANOVA test. This indicates that participant abilities were well balanced between the groups.
Because subjects were free to finish a task, if they felt its current state being precise enough, the difference between the mean x and y-translation errors in the camera navigation block can be taken as a hint that camera navigation does not need to be as precise as object manipulation in this field of application - because one condition was more restricting than the other. Thus DoF separation of camera navigation tasks, as a mean to increase precision, might not be as desirable as separation of object manipulation tasks for the user.

To be able to tell how easy a certain technique is to use, a second technique to which a relative comparison can be made to, is needed. Therefore we cannot deduce an "ease-of-use" rating from the subjects relative estimations. Comparing the completion times of the training section and the main section of the camera navigation block might give some impression, if subjects adapted to the new techniques.

Figure 6.11 shows a comparison of the average completion times of the training and main section of the camera navigation block. In contrast to the results of the main section completion times in the training section are more wide spread and in general higher despite the lower difficulty. Because the homogeneity of variances criterion could not be met (P=0.13<0.05), we refrained from performing an independent T-Test and used an ANOVA test between two groups. According to the ANOVA (F:(1:28)=19.759, P=0) as well as the Welch (P=0) and Brown-Forsythe tests (P=0) we can state that there is a significant difference between completion times during the training and the main phase. We can make this assumption independently from the assigned groups, because we evaluated that there are no significant differences between interfaces for the camera navigation tasks.
These results support our observations during the trial that participants adapt quickly to the camera navigation techniques and at some point of the trial reach a state where he/she is not hesitating to make his/her next action. Additionally many participants stated in their concluding interview that they were overall pleased with the camera navigation technique, but still pointed out some difficulties. Some participants stated that they often reversed the direction of the camera panning, expecting it to translate to the left, when actually translating to the right.

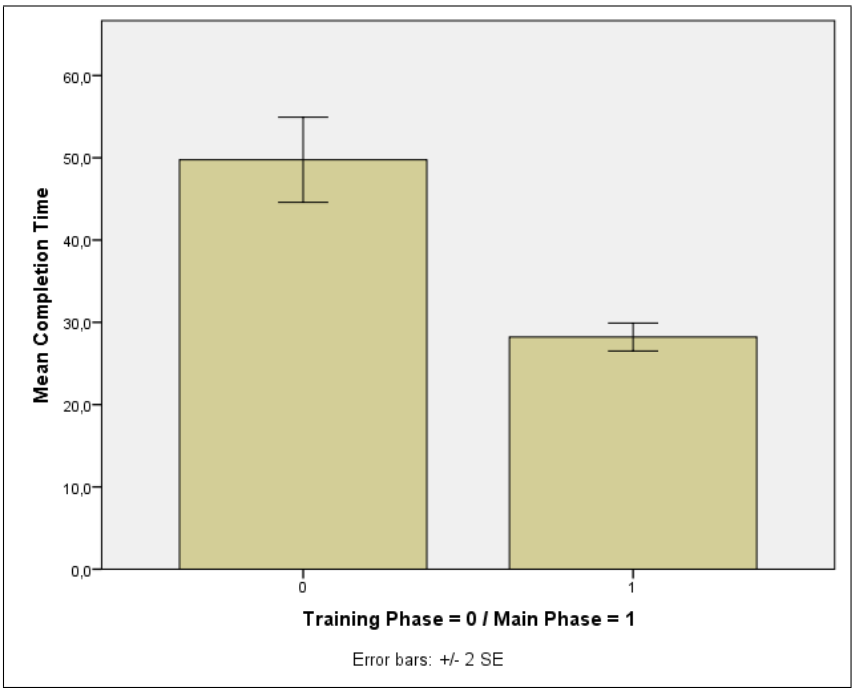

Figure 6.11: Mean completion time comparison between the training and main phase of block one

Although differently designed, no significant differences could be found between the three interfaces in the second block, either. Reasons for this could be inappropriate recognition of input techniques or task difficulty being too high. It could also be possible that participants struggled to utilise the interaction techniques itself. On the other hand the high amount of tasks with low completion times somewhat counters the assumption of the participants struggling to handle a technique appropriately, especially for the easy and medium tasks. This might indicate that not a certain technique caused problems, but consecutive usage of different techniques as required for the harder tasks, which often have a high completion time. There is also the possibility that the interfaces were designed too similar, thus effects through different behaviour might be too low.

At the end of the trial participants were asked whether the problems encountered during the trial were due to unexpected reactions of the application or the task being too complex for any reasons including struggling how to use the available techniques to reach the goal. Table 6.2 provides an overview about the given answers, which were taken in 5-point Likert chart. This data indicates that participants were substantially more concerned with unexpected reactions of the system than with the difficulty of the tasks. Observations during the trial also suggest that subjects were more often forced to repeat or correct their input because of unreasonable reactions of the application than due to difficulties using the interfaces techniques as intended by the design or struggling with the task itself. Subjective comments at the end of the trial support the assumption that the touch recognition system should be overhauled with priority in future elaborations. Related to this issue we observed that participants often chose to use a technique with a RESTING touch to reduce chances of a false recognition. For example when scaling with two fingers in interface one participants used a RESTING and a MOVING touch instead of two MOVING touches because this might result in unintended translations because of the two touch translation technique.

Although results of the precision condition cannot be used for any statistical meaningful statements we can make out substantial differences which may indicate where subjects struggled most often. It seems that the combination of an ArcBall technique and a two touch single axis rotation provide better accuracy than the ArcBall technique alone. Additionally the subjects of group three could have had problems with the two finger translate-rotate technique interfering with the rotation condition while scaling with two fingers. Since this group had the same rotation techniques as the subjects of group two, the different error extends can only result from recognition of the two-finger

Question	Mean	SD	Median.	Min	Max
Did you have a hard time solving the tasks?	2.6	1.18	3	1	4
Did your problems occur because of unexpected reactions on your input?	3.33	1.29	4	1	5
Did your problems occur because you found the task to difficult?	1.6	0.99	1	1	4
Did you find the experiment too long?	2.07	1.22	2	1	4

Table 6.2: Results of the interview concluding the experiment which were reported in 5-point Likert chart.

rotation-translation technique during two-finger scaling. This is supported as subjects in group three stated that "they had no idea why the object sometimes was rotating and moving". We also observed that only one subject of group three ever used the two finger translate-rotate technique intentionally others did not use this technique, either because they did not find any use for it, or they did not fully understood the techniques properties.

Since there were no significant differences in task completion time and error margins between all interfaces in the object or camera block, we cannot make any meaningful assumptions about relative learnability. Though subjective comments and observations suggest that usage of at least the camera navigation, object translation and for some subjects the scale and rotate techniques, too, became less and less difficult to use over time. "After I performed all techniques a few times I had no trouble using them". Some inexperienced subjects though noticed the "huge cognitive effort" required to be able to solve the tasks at first.

Most subjects pointed out serious problems with the rotation techniques. Many of them stated having problems understanding the three point ArcBall technique. In addition subjects of group three suggested to remove the two finger rotate-translate technique or at least not use the same finger count as for the scale + rotate technique. Only one person stated that he did not like the heavy usage of the camera navigation to allow restriction of manipulation axes. Some participants though had serious problems with the orbit technique resulting in overall worse trial results. For the orbit technique five distinct touches were necessary. If the person had small hands or had trouble spreading his fingers, even optimizing parameters of the recognition software did not produce any satisfying results. These subjects had to either press very hard on the

screen or spread their fingers with help of a handkerchief put between their fingers. However, none of the participants stated that they had serious problems using the orbit technique, if it was recognised properly. Participants with high experience in using CAD applications gave some more detailed feedback as they wished for features like custom axis rotation and snap-to-grid functionality. One of them rejected the idea of multi-touch for 3D design applications, but rather would like it for the view port concept. In addition a subject of group one desired an axis restricted rotation, which was absent in his group's interface, but included in every other group's interface, emphasising the positive impression of this technique during the trial. Moreover, we made an interesting observation during the trial. Since the screen was vertically aligned many subjects mentioned difficulties raising their arms to the top of the screen because it got more and more exhausting over time. As a result participants relied on a limited area of the screen for touch input which would not be possible, if we enforced direct manipulation in our interfaces. We therefore suggest that touch input with any offset could be provided in vertically aligned display systems to reduce fat-finger related problems and fatigue. Some overall inexperienced participants had trouble thinking of a field of application in the 3D domain for multi-touch input except design applications and games, but despite the difficulties and drawbacks encountered, overall feedback was positive and many of the subjects were eager to test future improved versions.
Our results suggest that the idea to control degrees of freedom depending on camera position/orientation could become a, to some degree usable, interaction concept for 3D design applications. At least it seems promising enough to justify future investigations in this direction.

7 Conclusion and Future Work

In this thesis we provided an overview about past and recent research on multi-touch technology and its usability for various interactions. We contributed by exploring the possible usage of multi-touch technology in 3D CAD and design applications, such as Maya and AutoCad. After the explanation of important interface design paradigms and discussion of its implications on 3D manipulation and navigation tasks we formulated an initial design.

Proceeding the adaption of our initial design by suggestions of a user survey, we implemented three different interface prototypes. By cross comparing the interfaces in an user evaluation, we expected to find some of our techniques performing better than others in terms of speed and/or accuracy, gaining a relative estimate on the usability of the interfaces. Unfortunately we did not find any significant differences between the prototypes. However, participants were able to finish most of the tasks and their subjective comments after the trial were overall positive. Thus we may state that usability was achieved to some degree in each interface, but it is highly probable that it might be negatively influenced by problems with the gesture and touch recognition process. Tasks which required participants to change all object properties were often nearly finished, but as participants tried to make small changes to finalise their task, techniques were falsely recognised and lead to object states which were not correctable in the remaining time. As an example participants were heavily distracted by unintended execution of the two finger rotate+translate technique in interface three, which caused many problems changing not one, but two object properties on a single false recognition.

Future elaborations have to explore, if a distinction of gestures by motion type can be used to separate interaction tasks or task DoFs without putting too many constraints on the input of the user since this may decrease ease-of-use and predictability of the interface. Our attempt to distinguish GROUPED and RELATIVE touch movement as well as RESTING and MOVING already required the participants to adapt their input motion in several ways. On the one hand, participants could not move touches as fast as they desired, but were forced to stay below a certain motion speed or the touch(es) would not be captured consistently enough. On the other hand, moving to slow led to

classification of touches to the RESTED state resulting either in none or false behaviour. Moreover, participants often struggled to move a group of touches without affecting relative location of the touches within the boundaries of the recognition processes constraints. The three touch scale technique for example was often recognised as the ArcBall technique and vice verse.
Thus it may be more viable to map only a single technique to a certain amount of touches although it might be required to use widgets or other mode switches to allow the same functionality we provided in our prototype. As long as some of the techniques separated in this way provide a useful and predictable combination of multiple tasks, we still could utilise the strength of multi-touch.

Since touch recognition and interaction techniques depend on each other, interaction techniques have to evolve inside the hardware and gesture recognition systems boundaries. A system which can distinguish users and fingers, recognise point size, shape or fiducials with high accuracy could allow more diverse and unique input techniques than a touch screen with mere point detection. The mightier the recognition procedure is, the more interaction richness might be achieved before needing to resort to using mode switches. Considering the interaction techniques which are not separated by a mode switch provide access to a combination of tasks in a useful, predictable and easy-to-use, hence, usable way.
Besides improvements to the gesture recognition system we have some ideas for immediate adaptions to our interfaces based on the observations and results of the evaluation. First of all the current translation technique takes the input point's projection as a reference point for choosing the translation plane. Therefore it can happen that a translation cannot be undone with the reversed motion. We don't think this is something an interface can benefit from. But it is possible to take advantage of this behaviour by performing translations in a certain order with different off-sets. This way three DoF translation with one finger without changing the view point is possible. We can also imagine a designated area on the screen to switch modes by gestural input. Such as drawing a circle activates another input-to-task-mapping allowing diverse rotation tasks, while drawing a line will switch back to the translation mode.

Moreover, subjects of our trial often stated "that a vertical aligned display is not optimal for this kind of application", pointing out that providing a fitting work environment is part of the user interface, too.
On this issue tablet PCs may provide a more comfortable approach for long term touch interaction and moreover, provide very precise and fast touch recognition, hence they

often have capacitive touchscreens. Furthermore they are often accompanied with a pen which could increase precision even further, also the build-in camera may be extended to a stereoscopic camera so that gestures above the tablets surface may assist the user.

We envision the user being able to use applications with an input concept which utilises the advantages of several input devices into one user interface. Combinations of multi-touch, pen, mouse and keyboard input could provide speed up prototyping with touch and pen without sacrificing pixel accurate input with keyboard, mouse or pen.
This however, requires us to identify and understand why, where and when the various devices perform best within an application to be able to design new interfaces, which are more accessible and motivating to use, but at the same time still are as reliable and precise as previous user interfaces.

Bibliography

[1] Abowd, G., and Beale, R. Users, systems and interfaces: A unifying framework for interaction. In *Proceedings of HCI '91 People and Computers VI* (1991), D. Diaper and N. Hammond, Eds., pp. 73–87. 5, 6

[2] Astheimer, P., Böhm, K., Felger, W., Göbel, M., and Müller, S. Die virtuelle umgebung - eine neue epoche in der mensch-maschine-kommunikation. In *Informatik Spektrum* (1994), p. 17(6). 6

[3] Benko, H., A., W., and Balakrishnan, R. Sphere: Multi-touch interactions on a spherical display. In *Proceedings of ACM User Interface Software and Technology* (2008), pp. 77–86. UIST '08. 10

[4] Benko, H., and Feiner, S. Balloon selection: A multi-finger technique for accurate low-fatigue 3d selection. In *Symposium on 3D User Interfaces* (2007). 3DUI '07. 11, 15

[5] Benko, H., Morris, M. R., Brush, A. J., and Wilseon, A. D. Insights on interactive tabletops: A survey of researchers and developers. Technical Report MSR-TR-2009-22, 2009. Microsoft Research One Microsoft Way Redmond, WA 98052, USA. 1

[6] Benko, H., Wilson, A. D., and Baudisch, P. Precise selection techniques for multi-touch screens. In *Proceedings of the SIGCHI conference on Human Factors in computing systems* (2006). CHI '06. 2, 9, 10, 11

[7] Blitzer, D. L., and D., S. PLATO IV - an economically viable large scale computer-bases education system. National Electronics Conference, Dec. 1968. University of Illinois. 10

[8] Brandl, P., Forlines, C., Wigdor, D., Haller, M., and Shen, C. Combining and measuring the benefits of bimanual pen and direct-touch interaction on horizontal interfaces. In *Proceedings of the working conference on Advanced visual interfaces* (2008). AVI '08. 10

[9] Buxton, B. Multi-touch systems that i have known and loved. online article `http://www.billbuxton.com/multitouch0verview.html/`. Date 08.02.2012, (17:56). 8, 9, 13, 42

[10] Buxton, W., and Myers, B. A study in two-handed input. In *Proceedings of CHI'86* (1986), pp. 321–326. 2

[11] Christie, A. Does facebook prove that bad usability and sparse documentation don't matter? http://www.itauthor.com/2011/10/14/does-facebook-prove-that-bad-usability-and-sparse-documentation-dont-matter/. Blog entry of ITauthor.com, Date 08.02.2012, (18:11). 8

[12] Corp., M. Windows developer preview - windows 8 guide. `http://windows.microsoft.com/en-US/windows-8/preview`. Date 08.02.2012, (19:04). 9

[13] Dietz, P., and Leigh, D. Diamondtouch: A multi-user touch technology. In *Proceedings of the 14th annual ACM symposium on User interface software and technology* (2001), pp. 219–226. 10

[14] Esenther, A., and Ryall, K. Fluid DTMouse: Better mouse support for touch-based interactions. In *Proceedings of the working conference on Advanced Visual Interfaces* (2006), pp. 112–115. Venice, Italy. 1

[15] Forlines, C., Shen, D., and Balakrishnan, R. Direct-touch vs mouse input for tabletop displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2007), pp. 647–656. San Jose, California, USA. 1

[16] Grossman, T., and Wigdor, D. Going deeper: a taxonomy of 3d on the tabletop. In *Second Anual IEEE International Workshop on Horizontal Interactive Human-Computer Systems* (2007), pp. 137–144. 10

[17] Han, J. Y. Low-cost multi-touch sensing through frustrated total internal reflection. In *Proc. 18th Annual ACM Symposium on User Interface Software and Technology* (2005), pp. 115–118. UIST '05. 1

[18] Hancock, M., Carpendale, S., and Cockburn, A. Shallow-depth 3d interaction: design and evaluation of one-, two- and three-touch techniques. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (2007). CHI '07. 11

[19] Hart, J., Ridley, C., Taher, F., Sas, C., and Dix, A. Exploring the facebook experience: a new approach to usablity. In *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges* (2008). NordicCHI '08. 8

[20] Herot, C., and Weinzapfel, G. One-point touch input of vector information from computer displays. In *Computer Graphics* (1978), pp. 210–216. 2, 10

[21] Hinckley, K., Pahud, M., and Buxton, B. Direct display interaction via simultaneous pen + multi-touch input. In *Society for Information Display (SID) Symposium Digest of Technical Papers, May 2010, Volume 41(1), Session 38* (2010), pp. 537–540. 10

[22] Hinckley, K., Pahud, M., Coddington, N., Rodenhouse, J., Wilson, A., Benko, H., and Buxton, B. Pen + touch = new tools. In *Proceedings of ACM UIST '10* (2010), pp. 27–36. 10

[23] Hinckley, K., Tullio, J., Pausch, R., Proffitt, D., and Kassell, N. Usability analysis of 3d rotation techniques. In *Proceedings of the 10th annual ACM symposium on User Interface software and technology* (1997). UIST '97. 28, 45, 73

[24] Holz, C., and Baudisch, P. The generalized perceived input point model and how to double touch accuracy by extracting fingerprints, 2010. CHI 2010, Atlanta, Georgia, USA. 2, 11

[25] Ishii, H. Tangible bits: beyond pixels. In *Proceedings of the 2nd international conference on Tangible and embedded interaction* (2008). TEI '08. 9

[26] Ishii, H., and Underkoffler, J. Illuminating light: An optical design tool with a luminous-tangible interface. In *Proceedings of CHI '98* (1998), pp. 542–549. 9

[27] Ishii, H., and Underkoffler, J. Urp: a luminous-tangible workbench for urban planning and design. In *Proceedings of CHI '99* (1999), pp. 386–393. 9

[28] Kable, R. G. Electrographic apparatus. United States Patent 4,600,807, July 1986. assigned to Scriptel Corporation, Columbia, Ohio. 1

[29] Martinet, A., Casiez, G., and Grisoni, L. The design and evaluation of 3d positioning techniques for multi-touch displays. In *3D User Interfaces (3DUI), 2010 IEEE Symposium* (2010), pp. 115–118. 8, 12, 31, 40, 65

[30] McAvinney, P. The sensor frame - a gesture-based device for manipulation of graphical objects, 1986. Carnegie-Mellon University. 2

[31] Mueller-Tomfelde, C., Schöning, J., Hook, J., Bartindale, T., Schmidt, D., Oliver, P., Echtler, F., Motamedi, N., Brandl, P., and Zadow, U. Building interactive multi-touch surfaces. In *Tabletops - Horizontal Interactive Displays* (2010), C. Mueller-Tomfelde, Ed., Human-Computer Interaction Series, Springer London, pp. 27–49. 1

[32] Nacenta, M. A., Baudisch, P., Benko, H., and Wilson, A. Separability of spatial manipulations in multi-touch interfaces. In *Proceedings of Graphics Interface 2009* (2009). GI '09. 11, 12, 16, 47

[33] Nakatani, L. H., and Rohrlich, J. A. Soft machines: A philosophy of user-computer interface design. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (1983). CHI '83. 10

[34] Norman, D. A. Emotion & design: attractive things work better. In *Magazine Interactions* (2002), pp. 37–42. Volume 9 Issue 4. 8

[35] Reisman, J. L., Davidson, P. L., and Han, J. Y. A screen-space formulation for 2d and 3d direct manipulation. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology* (2009). UIST'09, Victoria, British Columbia, Canada. 2, 12

[36] Rekimoto, J. Smartskin: An infrastructure for freehand manipulation on interactive surfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves* (2002). CHI '02. 2

[37] Shoemake, K. Arcball: A user interface for specifying three-dimensional orientation using a mouse. In *Graphics Interface* (1992), pp. 151–156. 74

[38] Spolsky, J. User interface design for programmers. online article `http://www.joelonsoftware.com/uibook/fog0000000249.html`. Date 30.01.2012, (16:49). 7

[39] Strothoff, S., Valkov, D., and Hinrichs, K. H. Triangle cursor: Interactions with objects above the tabletop. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces* (2011), pp. 111–119. 11

[40] Valkov, D. Navigationstechniken für interskopische multi-touch oberflächen (imuts based navigation techniques), 2009. Diploma Thesis. 5, 6

[41] Valkov, D., Bruder, G., Bolte, B., and Steinicke, F. Viargo: A generic vr-based interaction library, 2011. SEARIS Workshop in konjunction with IEEE VR. 48

[42] Valkov, D., Steinicke, F., Bruder, G., and K., H. 2d touching of 3d stereoscopic objects. In *Proceedings of the 2001 annual conference on Human factors in computing systems* (2011). CHI '11. 12

[43] Vogel, D., and Baudisch, P. Shift: A technique for operating pen-based interfaces using touch. In *Proceedings of CHI '07* (2007). 11

[44] Weiser, M. The computer for the 21st century. In *Scientific American* (September 1991), pp. 94–104. 1

[45] Wigdor, D., Williams, S., Cronin, M., Levy, R., White, K., Mazeev, M., and Benko, H. Ripples: utilizing per-contact visualisations to improve user interaction with touch displays. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology* (2009). UIST '09. 2, 11, 51

[46] Wilson, A. D., Izadi, S., Hilliges, O., Garcia-Mendoza, A., and Kirk, D. Bringing physics to the surface, 2008. UIST '08, Monterey, California, USA. 2, 10

[47] Yu, L., and Isenberg, T. Exploring one- and two-touch interaction for 3d scientific visualisation spaces. University of Groningen, the Netherlands. 12