Edge-Based Multi-Touch Graph Exploration Techniques

Project work thesis

Faculty of Computer Science

Otto von Guericke University of Magdeburg

by: Sebastian Schmidt

Course: Computervisualistik

Student number: 168650

Supervisors: Jun.-Prof. Raimund Dachselt
Otto von Guericke University of Magdeburg

Dr. Miguel A. Nacenta
University of Calgary

Prof. Dr. Sheelagh Carpendale
University of Calgary

Abstract

Node-links diagrams are a very common way to visualize datasets of different graphs like friendships in social communities, flight maps or communication networks. In these datasets nodes are often connected by a huge number of edegs, which leads to edge crossings and edge congested areas, occluding nodes and further information underlying the diagram. To allow exploring the graph despite those problems, several graph interaction techniques approaches have been developed.

This thesis explains the adaption of such techniques for the use on a multi-touch tables. Additionally I propose a new technique, developed using the new possibilities provided by touch devices. All proposed techniques are designed for simultaneous use in one application. A gesture set is suggested, allowing concurrent interaction on all techniques without changing modes.

The implementation of an example application regrading different challenges in edge drawing and the new field of gesture interaction is described. Significance and limitation of the system are named and some outlines for future work into this research field are given.

Zusammenfaßung

Netzwerk Diagramme bilden die Grundlager für die Darstellung von Graphen und können Bekanntschaften in sozialen Gruppen, Karten von Flugrouten oder verschiedene Kommunikationsnetzwerke darstellen. Knoten in diesen Diagrammen haben oft eine hohe Anzahl an Verbindungen untereinander. Dies führt zu Anhäufung und zunehmender überschneidung von Kanten. Die Kanten überdecken dabei Knoten und weitere unter der Visualisierung liegende Informationen. Um trotzdem eine Analyse dieser Daten zu ermöglichen wurden verschiedene Interaktion-Techniken entwickelt.

Diese Studienarbeit beschreibt die Adaption dieser Techniken für die Nutzung an Multi-Touch Tischen. Zusätzlich wurde eine weitere Interaktions-Technik, unter Berücksichtigung der neuen Möglichkeiten der neuen Eingabemedien, entwickelt. Alle gezeigten Techniken können gleichzeitige und ohne Wechsel in verschiedene Modi in einer Applikation genutzt werden.

Die Implementierung einer Beispiel Anwendung wird beschrieben und Problemstellungen im Bereich Kantendarstellung und dem neuen Forschungsfeld der Gesten-Interaktion werden erklärt. Bedeutung und Einschränkungen des Systems werden zusammenfaßend genannt und Richtungen für eine weitere Entwicklung in der Forschung angedeutet.

Contents

1 Introduction

Graphs have a very long history in mathematics, computer science and other fields. They can represent a big variety of data and provide a good overview on the information data, when shown in visualizations. The typical visual representation of a graph is the known node link diagram, where the nodes represent an element and links or connections between these element, are shown as edges.

The amount of data, behind these diagrams, tend to grow which causes the visualization to get more complex and less understandable. Especially a growing number of connections, leads to typical problems in overlapping of several graphical elements. These problems complicate exploration of the data and makes it even impossible sometimes. Several techniques to minimize these issues and to allow an easy exploration have been developed. However these techniques have been developed only for typical desktop environments using mouse and keyboard input.

This thesis presents the adaption of such techniques for the use on touch devices, focused on Multi-Touch-Screen (MTS). The developed application allows interaction with several exploration techniques and multiple elements at the same time.

1.1 Definitions

In this work I make use of several terms, which are not well defined in literature or used in a wide spread of meanings; sometimes the commons use differs from research terms. The given definitions are made in consideration of this work and the usage of these terms might change in different contexts.

Gesture Considering Kurtenbach and Hulteen [11] "A gesture is a motion of the body that contains information. Waving goodbye is a gesture. Pressing a key on a keyboard is not a gesture because the motion of a finger on its way to hitting a key is neither observed nor significant. All that matters is which key was pressed."

In this work a gesture is a specific movement of touch by one or multiple fingers on a surface. The complete gesture is recognized by a touch enabled device and can be either obvious like moving an object with touching and dragging it or rather abstract, as in tapping the objects to remove it.

Interaction Technique Interaction techniques described in this work are always the composition of a gesture and action in the graph visualization according to this action.

MTS A Multi-Touch-Screen (MTS) is a touch device, where the touch enabled surface is a display screen.

Bridge In a mathematical graph a bridge is an edge, where removing this edge disconnects the graph.

Articulation Point An articulation point in a mathematical graph is a node, where the graph is disconnected when removing this node.

1.2 Motivation

In our world, the amount of data grows continuously and for data analyses and exploration visualizing this data in diagrams is vital. On kind of data is related data representing friendships in social communities, flights from one airport to another or event people related to a product they bought once in the collection of selling companies. For visualizing this relating data, graph diagrams have proven to provide good results [6, 2]. However even these diagrams show their limitations, when a huge amount of data has to be visualized. In heavily connected graph diagrams, we figure the problem of edge congestion, which often made it hard, sometimes even impossible, to explore the data in a fast and comfortable way. To solve such problems some tools and techniques for graph exploration have been provided by the research community.

All of these tools and techniques have been designed for common desktop systems using mouse and keyboard input. Looking into the last years touch enabled devices become more and more common in usage and affordable for people. The goal of this work is to combine the known techniques from desktop environments with touch technology and to show new techniques which can hardly be done on desktop systems, or at least seem more natural on touch enabled devices.

1.3 Graphs and the goals of graph exploration

Graphs play an important role in the field of information visualization for relationship based datasets. Often graphs are shown as simple node-link diagrams. These can be maps of streets in a navigation system or of bus and train routes in public transit. Another field where graphs get more and more common are connections in social communities, where an edge can represent a friendship or business relations. Some of these visualizations are shown in 1.1.

Considering the wide field of domains, where graphs are used there is a huge variety of tasks as well. Lee et.al. [19] proposed a Task Taxonomy on graph visualizations for designing and evaluating new systems in this area; I want to present just a few in this section and give real examples, some tasks are named again in the technique section 3 in context of usage for the technique.

A very obvious task in graph visualization is revealing of two nodes are connected, which can either be detecting if two people know each other in a social map or if I can go from one town to another by plane with a single flight.

Sometimes it might be interesting to count the number of adjacent nodes to a specific one, when looking for the number of relatives in a family tree regarding heritages or looking for the number of flowers eaten by cows, which can be read out of a food chain.

People might not only want to observe direct neighborhoods in a graph, but go further into the seconds third and more iterations, looking which friends do they friends have and if they might know them as well. This applies to the flight map, when looking for alternative routes and for UML diagrams, looking from how many classes the current inherits. So far, these tasks have in common that they sound pretty simple and easy to perform on most graph visualizations, like shown in figure 1.1 (a); but looking if there is a flight between two specific places in figure 1.1 (b) is already much more difficult and finding a friend of my friends in figure 1.1 (c) seems already impossible.

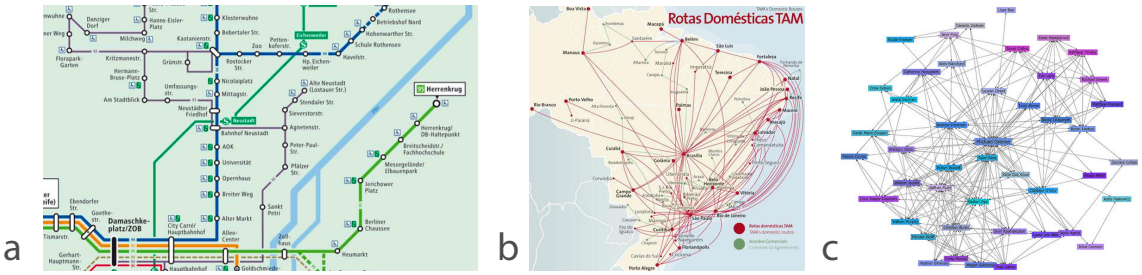

Figure 1.1: Three different type of data visualized using different layouts. Perceiving specific information becomes more difficult going from image (a) to (c). Sources (by 18. July 2010):
(a) http://www.mvbnet.de/downloads/pdf/MVB-Liniennetz_Anschlussverkehr_141209.pdf
(b) http://www.bigtravelweb.com/images/tam-l.jpg
(c) http://farm1.static.flickr.com/

Looking into these figure shows the problems which are to solve in graph visualizations, while tasks can get even more complex, looking into a communication network, the observer often need to know, where are bridges and articulation points to avoid disconnection of parts of the network.
For social research a community network can be real joy, for them it is often interesting where people tend to group, which shows as a cluster in the graph visualization.
Especially when thinking in economic ways it is often interesting to find the shortest path, either if this mean getting from one Town to another without as few changes of flights as possible, or finding the shortest path on the routes to another place or simply the fastest way a data package should go through a TCP/IP network structure.

1.4 Touch interfaces as new input devices

Nowadays probably nobody would name the first multi-touch device as such. As in general understanding, it is a simple keyboard where pressing two keys at the same time, forces a reaction. Although these first beginnings of multi-touch interaction can not be retrieved [3], the field got a new push in the last years, when the touch devices transformed to touch-screens. Touch screens allow the changing of the direct interaction surface; something not imaginable for a normal keyboard or mouse. This gets even more

useful, when the touch-screens get to multi-touch-screens (MTS), where not only one touch can be recognized but all touched areas.

There are several approaches to enable screens for multi-touch recognition. The first distinction has to be made in optical and capacitive devices. Optical devices use a camera, which is mostly beneath the surface to record the hand interaction. It is necessary to illuminate the surface for these cameras. To avoid problems with the visual scene on the surfaces, the light and the camera works mostly with infrared light, which is not visible for human eyes. However there are still problems, when bright infrared light, emitted by the sun for example, hits the surface and is recorded by the camera. Capacitive touch devices do not have to figure such problems, but are more expensive and more difficult to produce, when used on big screens.

1.5 Scope

This work focuses on graph exploration techniques and not on creating or editing them. This is caused by some research work which has been done on MTS and graph editing and creation while I did my work and by the idea of just exploring data, while the information itself is generated by algorithms out of existing environments.

Further the application deals mainly with edge manipulation, which is less explored compared to pure node interaction and manipulation in terms of graph layout algorithmic. The implemented techniques are designed to solve the less complex tasks presented in section 1.3, because the more complex ones are often a composition of these.

The application is designed to show usage of multiple techniques at the same time to show the possibilities multi-touch systems provides us; as well as it is designed to allow multiple-user interaction. However these approaches where not the main goal of my work, that is why I do not provide a fully new way of solving problems in this area.

1.6 Overview

The following chapters of this thesis are structured as follows.

Chapter 2 gives a short introduction into the field of graph exploration, with focus on edge interaction and the research in multi-touch and gesture interaction.

Chapter 3 explains all techniques I designed and implemented into my example application. The gestures used for each interaction technique are described in detail.

Chapter 4 provides a detailed explanation of the implementation of all techniques. Technical problems occurred in the design process and their solution are discussed.

Chapter 5 gives a summarization of this thesis and names some limitations in the developed approach. This chapters finishes with a prospectus of further advancements which might be included into this project in future development.

2 Related Work

Graph visualization is not a new area of research. With the beginning of Graph theory people have made node links diagrams. Even before this the first graphs have been drawn, for example in Family trees. With the upcoming of computers and graphical interfaces these visualizations were ported to this new medium. Computers do not only allow to visualize given data, but also allow further interaction and editing. Because of the growing data and the limitations of display size, research provides several further visualization techniques.

2.1 General graph exploring techniques

2.1.1 Graph layout algorithms

As mentioned before one of the first approaches in graph visualization have been the family trees in the middle ages. Several layout algorithms were developed [6, 2] and evaluated in [5]. Some random layouts are illustrated in figure 2.1. Trees for example represent a graph layouts easily to understand. They are showing a hierarchy of the data starting a root and ending in leafs. To draw graphs in a tree layout the data has to be analyzed in forehand and equivalence relations have to be found.

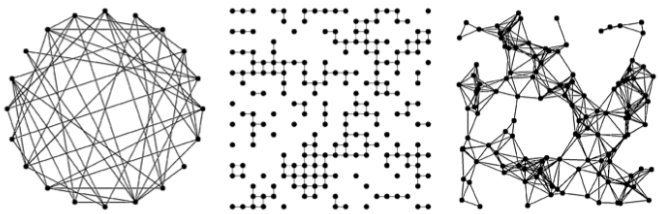

Figure 2.1: Different graph layouts, shown in [5].

Another common way to layout a graph is the orientation on the longest path. Therefore the path has to be found and is drawn in one line. All other nodes and edges are placed regarding their relation to this path. This approach is often used for graphs that simulate processes.

For datasets where to nodes can not be ordered by a hierarchy or other information, which is useful to layout them in a specific way, other algorithms have been developed. Most based on force-feedback, the Fruchtermann-Reingold [9] layout for example.

2.2 Distortion oriented techniques and EdgeLens

To offer a better overview on graph data without moving nodes Sakar et.al. proposed the fisheye view in graph application [23] based on the original fisheye from Furnas [10]. In this approach observing people can define a region of interest, in which nodes are highlighted by increasing the size and decreasing the size of nodes outside this region. Sakar proposed special algorithms for the fisheye on graphs where the nodes have a geographical meaning.

Basing on distortion oriented techniques like the fisheye, Wong et.al. introduced Edge-Lens [31], to avoid edge congestion. Like in the fisheye a region of interested can be defined. But instead of changing the nodes, the Edge-Lens manipulations the edges in this specific region of interest. The edges are bend away depending on the distance to the center of the lens and the two nodes they are connected with, illustrated in figure 2.2 (2). This gives a better view on the data the user specifies as important, without destroying the observers mental map of the graph in changing the size of the nodes or, even worse, move their position.

2.3 Edge manipulation techniques

The Edge-Lens gives the potential to manipulate all edges in a specific region, but without the possibility to manipulate one single edge. To give the user more control for interaction Wong et.al. extended the Edge-Lens and proposed Edge-Plucking for graph exploring [30]. It allows the plucking of a single edge like a strum of a guitar, by clicking and dragging with a mouse. People gain more control on which edges are bend, but still are able to clear a region with plucking all edges in there with a single gesture. Plucked edges can be pinned at a position and stay out of the current region of interest. One example is shown in figure 2.2 (1)

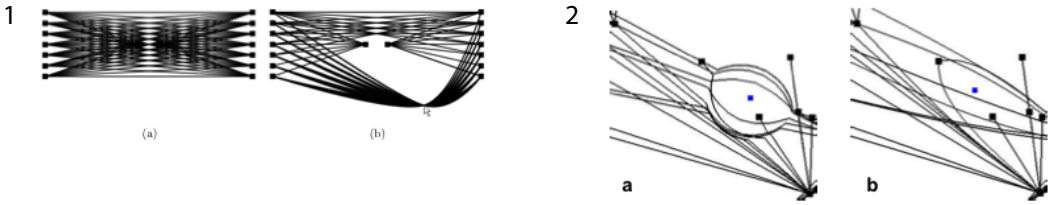

Figure 2.2: EdgePlucking (1) for mouse interaction as proposed in [30] and (2) a comparison of the bubble (a) and spline (b) approach of the EdgeLens [31].

Edge-Plucking lead to a bundling of edges, when a lot of edges are plucked and pinned together, they behave like a bundle. An automatic approach for bundling of edges was introduced by Danny Holten et.al. in [15]. Holten revealed, people using the Edge-Bundling in an example application where very exited to use it in a real graph exploring environment. An example application is shown in figure 2.3.

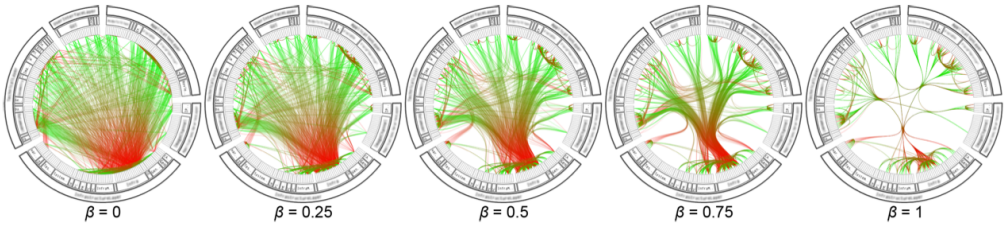

Figure 2.3: Edgebundling using different bundling strength ß, proposed in [15]

2.4 Touch techniques for interaction

Multi-Touch interfaces become common more and more, even in all day fields, for example the Microsoft Surface [28] or the SMART table [26]. As Buxton et.al. showed that two handed input allows the simultaneously interaction on different parameters [4]

Moscovich [20] spotted the advantages of multi-touch interaction and later Msocovich et. al. [22] proved that interaction touching with two hands is more suitable for change in multiple parameters and two touches from one hand are good for changing scale and orientation. Moscovich et.al. [21] showed Multi-Touch enables more degrees of freedom for direct interaction which can be used for scaling or rotating of objects on touch surfaces.

Fundamental research in usage of gesture interaction when using a touch enabled surfaces was done by Wobbrock et.al. [29]. He revealed out that people do prefer multi-touch gesture where the count of touches does not matter. When interacting directly with touches on a surface, people might expect a more realistic behavior. One was shown by Agarawala et. al. with BumpTop [1] as a virtual desktop environment. They as well propose small gestures for tossing objects.

In [7] Frisch et.al. presented a study to propose different multi- and single-touch techniques for general graph editing. Although we decided to concentrate on manipulation of edges, instead of graph editing, we use some of the proposed gestures. In the presented analysis people used similar, sometimes even the same, gesture for different operations. We think, these gesture have a common meaning on different objects, like the pinch gesture for resizing an object. Later a set of interaction techniques for graph editing and manipulation has been proposed [8].

3 Interaction techniques in Cutting Edge

In this chapter every interaction techniques is explained in detail. First the effects on the graph visualization are explained, which is followed by a listing of all gesture available for this technique. When available the composing of two techniques is mentioned and the gesture for interaction are explained. For every technique one example is usage according to the tasks listed in section 1.3 is expounded.

The presented techniques and gesture are presented as a set, already implemented in an example application. They are designed to work in simultaneous action. This needed to limitations in usage of technique or gesture, which are named as well.

3.1 Interaction with nodes

Although I focus on edge interaction in my work, interaction on nodes are a fundamental feature for graph exploration and should not be neglected; simple touch techniques on nodes are even necessary to enable specific features of edge interaction.

3.1.1 Moving nodes

Nodes can be moved by starting a touch on them and dragging them around. The nodes are moved according to the movement of the finger, while all edges stay connected and change their shape if they are applied to an edge interaction technique. With moving one node, no other nodes are affected; however multiple nodes can be moved at the same time.

Although the moving of nodes is not suitable, if there is specific information underlying the position of nodes, it is a very powerful tool for graph exploration. Nodes which might belong to a group can be located near to each other, so this approximates to the mental idea of the observer and this information can be easier shown to other people.

3.1.2 Tapping nodes

It can be necessary to revert all edges of one node into their original shape. This can be achieved by staying with a touching finger on a node for at least 1.5 seconds. All edges adjacent to this node are highlighted and set straight not matter which edge interaction techniques are applied. This state is held as long as the touch stays on the node and is not dragged; once the finger drags the nodes this state is lost and the usual movement of the node is enabled.

Setting the edges into a linear shape and highlighting them, can reveal connections to other nodes, especially if there are techniques applied to the edges, which bend them

in a bundle with other edges. While the edges are highlighted no further interaction is possible, which can be used to select edges not adjacent to this node and move them out of the focus of view; but is the starting point for another interaction technique directly used on edges and described in 3.2.1.

A short flicking gesture on a node causes another reaction of the adjacent edges, which is highly related to an edge interaction technique and is described in section 3.4.1

3.2 TouchPlucking

Plucking an edge was first introduced by Wong et.al. in terms of EdgePLucking[30] but was only performed in desktop interaction with mouse and keyboard. The technique is very powerful and the basis technique for several further approaches described in this work. Using TouchPlucking, people can "take" an edge and bend it according to the position of the finger and the adjacent nodes, as shown in figure 3.1. The new shape of the edge is a curve which starts and ends in the nodes and goes directly through the finger of the touching person.

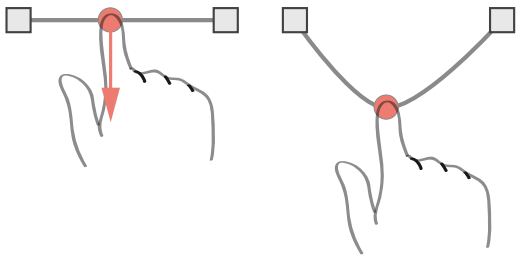

Figure 3.1: bending s single edge using TouchPlucking.

Plucking and edge can be started in different ways, each one has its advantages in different purposes. Starting a touch directly on one edge, selects just this one and enables it for plucking. The plucking is performed by dragging the finger away from this position. This can be achieved with multiple edges as well, when starting a touch in an edge congested area, or on a location where edges are crossing.

Another way to select multiple edges is, starting a touch on an empty space of the visualization, neither on an edge or any other element. Dragging this touch through edges collects them and bend them as in plucking a single edge. This technique select only edges, which are not affected by any other technique so far, and other objects are not affected in anyway. Although less precise, this indirect plucking enables selection of a larger amount of edges. The difference of direct and indirect plucking are shown in figure 3.2.

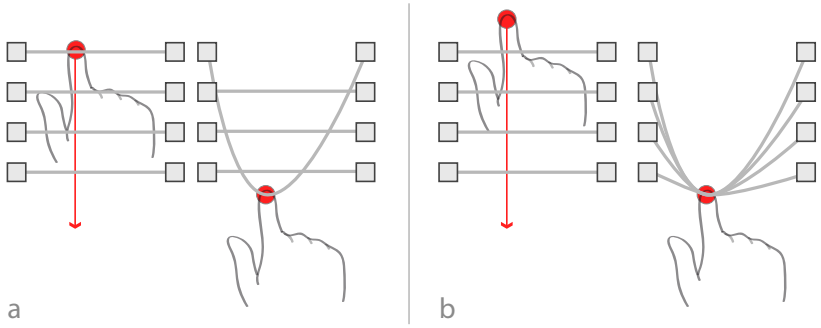

Figure 3.2: Two different ways of selecting and plucking edges. In (a) one edge is selected and plucked, which ignores other edges. Starting the touch offside any element and dragging it, selects all crossed edges as in image (b).

3.2.1 Single-Edge-Selection

In Touch applications, we often face the so called "fat finger" problem, which is selecting an interface element with a finger, where the element is much smaller than the touch area. We faced this problem with a multi touch technique; where one touch starts on a node and stays there for at least 1.5 seconds, which causes all connected edges to return into their original shape, no matter if they are plucked or pinned. While keeping this state, the edges cannot be plucked right away; but tracing along one edge selects it as in direct plucking and further interaction is possible, shown in figure 3.3. As this technique is for selection of a single edge, it is not possible to select more then one edge in one run.

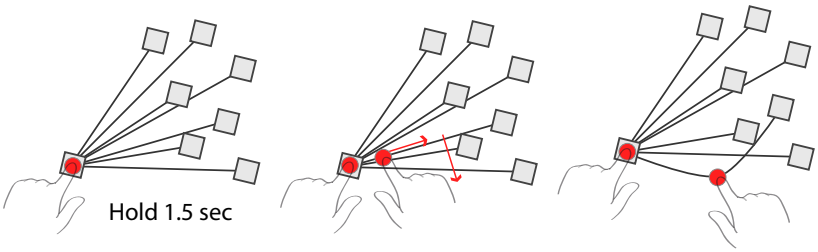

Figure 3.3: Plucking an edge out of a congested area. One finger touches a node for 1.5 seconds and a second one moves along the edge to select it.

Edges in a TouchPluck can be simply released with lifting the touch from the surface, which leads the edges into their original position. In this way, TouchPluck provides a fast way to explore the graph without changing the visualization permanently. However there is a difference in edge behavior, when the TouchPluck is released near to another touch, a pin or an edge bundle (3.3, 3.5). In that case the edges are assigned to the second pluck, which is indicated by a visual highlighting at the forehand.

3.2.2 TouchPlucking in usage

TouchPlucking allows the observer to reveal connections of nodes and reduces cluttering in areas of high edge congestion. Often it is not clear, if an edge goes through a node or this node is connected by two different edges. With EdgePlucking this problem is solved by plucking the edge, as shown in figure 3.4.

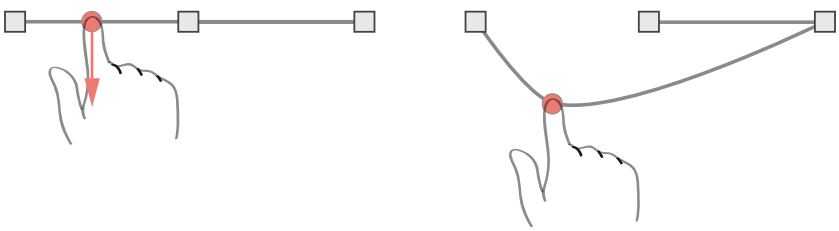

Figure 3.4: Revealing the adjacent of two edges using TouchPlucking.

3.3 TouchPinning

Although one of the main advantages of TouchPlucking is the not permanent change of the graph, it is sometimes necessary to change the visualization for a longer period of time, when further investigation is needed. For that the application provides a technique called *TouchPinning*; which is highly related to TouchPlucking and is fixing plucked and bended edges in their current position.

To gain this status a plucking touch stays at one location for at least 1.5 seconds and when this touch is released afterwards the edges are already fixed and stable. The change of state is highlighted with a small push pin as illustrated in figure 3.5.

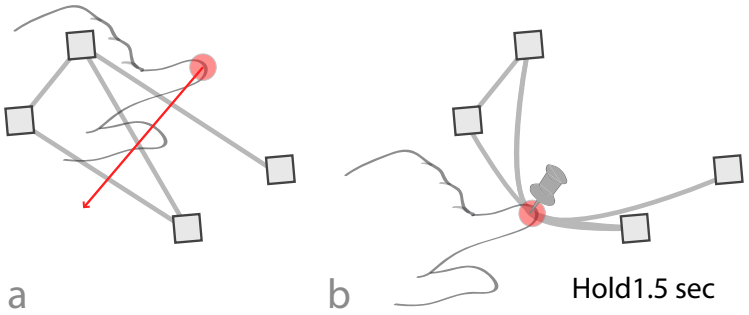

Figure 3.5: Fixing the shape of plucked edges using TouchPinning. The new state is illustrated with a push pin.

Once a collection of edges is pinned, more edges can be assigned to this pin, with plucking them near to it and releasing the finger, this works in the same way as the combining two plucks, explained in TouchPlucking 3.2.1.

Further interaction with once pinned edges can be achieved with starting a new touch directly on the pin. The edges then behave as if they were plucked directly, these edges then can be released or pinned again. If a pin is not needed anymore, a small click deletes it, with releasing all edges into their original position. I enabled the possibility to pluck a single edges out of the pin as well, however at this point, we are facing the "fat finger" problem again, so that the edge has to be plucked where it does not interfere with another edge, which might be impossible.

TouchPinning allows a permanent change of the graph visualization, which allows constancy for a longer exploration session, where some edges might be uninteresting for a longer term; but instead of the case, when making them invisible, the connections between nodes can still be revealed and with the option to pluck edges out of the bundle, they are accessible at every time.

The TouchPinning is very useful for multi-user systems, it allows to show a selection of edges in a bended state, without touching it every time, which would not only be unstable, when people could not place their finger in the same position for a longer period of time, but the arm of the touching finger, would occlude a lot of the visualization as well. This applied especially in top-projected systems, where the arm of a static touching finger would always reduce the visualization space.

3.4 TouchStrumming

TouchStrumming is another technique, related to TouchPlucking and takes advantage of the saliency of human motion-perception to offer an alternative way to visualize the connection of one or several edges. The metaphor behind the idea is a guitar string, which is strummed with a finger. Instead of dragging and plucking edges it uses a short “flick”-gesture. As expected the edges selected by this gesture are released immediately, but do not move back into their original position slowly, they are going to vibrate for some time, as shown in figure 3.6, where the amplitude and duration of the vibration depends on the length of the flicking gesture.

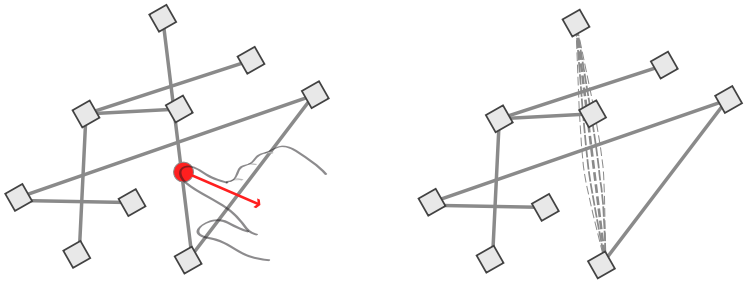

Figure 3.6: Applying a strumming motion to an edge, using a short “flick”-gesture.

Edges can be strummed when they can be plucked, in fact TouchPlucking is the beginning of a TouchStrumming action, just differentiated by the time and length of the gesture. If an edge is strummed out of a TouchPinning or a TouchBundle it is not assigned back into this again, but goes into its straight position.

3.4.1 Strumming on nodes

Strumming cannot only be assigned on edges, but on node as well. The nodes then is moved back into its original position and all adjacent edges are strummed. The idea is shown in figure 3.7. Despite strumming on edges, the maximum amplitude is not assigned to the perpendicular line regarding the ending position of the touchpoint, but goes along the edge. This simulates a movement of the vibration from the strummed node, to its neighbors.

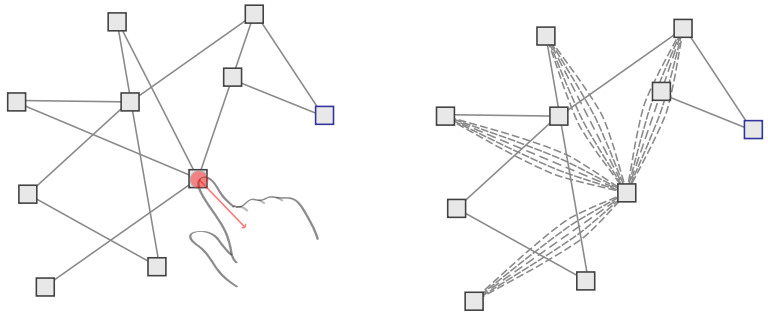

Figure 3.7: Using the same "flick"-gesture as for strumming edges on nodes, causes all edges to strumm.

3.4.2 Advantages of Strumming

Strumming can be used to reveal nodes adjacent to a specific edge, like the example in plucking where an edge goes through a node, strumming can reveal if these are two adjacent edges or it is a single edge, which is not connected. Unlike in plucking, the touching finger can be removed immediately after the gesture is performed, while the strumming is still applied; which reduces occlusion of other elements of the visualization. If a greater number of edges is bend already, strumming is easier to perceive by multiple then a further plucked one.

When strumming on two nodes at the same time, other nodes, which are connected to both of them, are easier to see and even subgraphs, which contain this nodes can be perceive and shown in a fast way. While some of these tasks can also be solved by a highlighting, strumming has further advantages. It can easily be applied to exiting applications, without concerning existing coloring or highlighting of edges, which often illustrates a specific meaning already. Despite color highlighting, strumming can be perceived by color-blind people as well.

3.5 TouchBundling

Although TouchPlucking and its derivates are very powerful tools for interaction with single and multiple edges, it can be necessary to combine edges, either to reduce cluttering or to highlight the connections between groups of nodes, which was the original idea of edge bundling, when it was proposed by Holten et.al. [15].

The gesture for TouchBundling is a two-touch interaction, where both touches move into one direction and select edges, like a funnel, which are roughly perpendicular to the line between both touches. One way to assign edges to a bundle is using this gesture on a bunch of edges, where all edges which are directed similar as the movement, with a range of $\pm 30°$. The idea of bundling is shown in figure 3.8 (a). Another way using the TouchBundle gesture, is bundling all edges of one or multiple nodes. This is achieved by crossing the node with the described gesture. All adjacent edges are added immediately to the bundle. Using the same graph as in figure 3.8 (a) , bundling on the nodes selects on edge more, illustrated in (b) of the same figure.

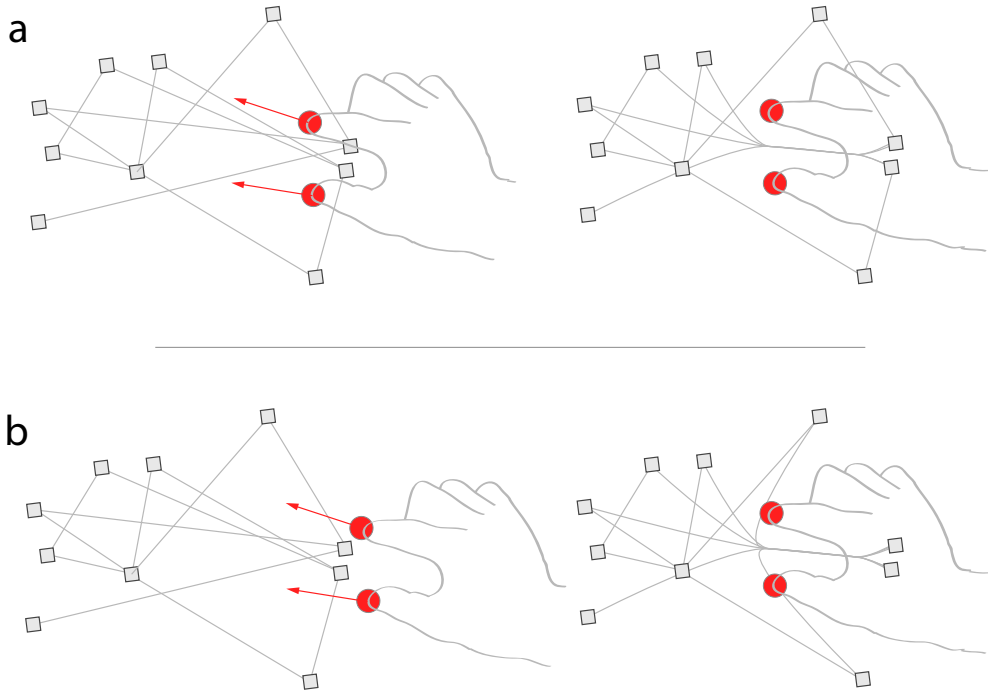

Figure 3.8: Bundling edges using a two-touch gesture. The gesture is started on edges in (a) and first crosses nodes, selecting all adjacent edges, in (b).

3.5.1 Bundling Interaction

Once the gesture is finished, the bundle is stable, but can be changed in position, by dragging the endings of the bundling segment, which are the points, where all edges run into each other. The shape can be altered as well, by using the TouchPlucking where

the bundle behaves similar to a single edge. Plucking a bundle is only possible in direct plucking, but not with indirect plucking.

Adding further edges to a bundle is possible by plucking them near to an ending of the bundling segment. Edges can be removed from a bundle, by simply plucking them at a place, where the bundle is not started yet or by tapping on the adjacent node, which releases all edges from this node and was earlier describes in section 3.1.2. The complete bundle can be deleted, by using a short tapping gesture on the bundle itself.

Bundling can be used to reduce cluttering of edges in a specific region of the graph visualization. Especially regarding subgraphs and other groups of nodes, the bundling can reveal the number of connections. In my example application this is done by highlighting the bundling depending of the number on edges it contains.

Like TouchPinning the TouchBundling technique is stable over a period of time, which allows further exploration without occluding other elements of the visualization. Despite the Pinning technique, a bundle does not spread the edges and looks more accurate. This reduces the cluttering even more. TouchBundles even allow further interaction with plucking and changing their position.

TouchBundling provides a good way perceiving bridges and articulation points in a graph visualization. When multiple people working in a visualization, bundling is suitable revealing these instances to others.

3.6 PushLens

The PushLens is a virtual circular object, which leads the edges to a behavior as if it would be physical real in the graph visualization, the idea is derived from the EdgeLens by Wong et.al. [31]. Edges intersecting the lens, but without an adjacent node inside the lens are bent along the shape of the lens, which is shown in figure 3.9; edges connected to at least one node inside of the lens are not affected by the lens. Plucked and pinned edges are affected, but not when the pin or the plucking finger is inside the lens, this is due to the fact, that the edges should stay connected to pin and touch, to keep the mental connection.

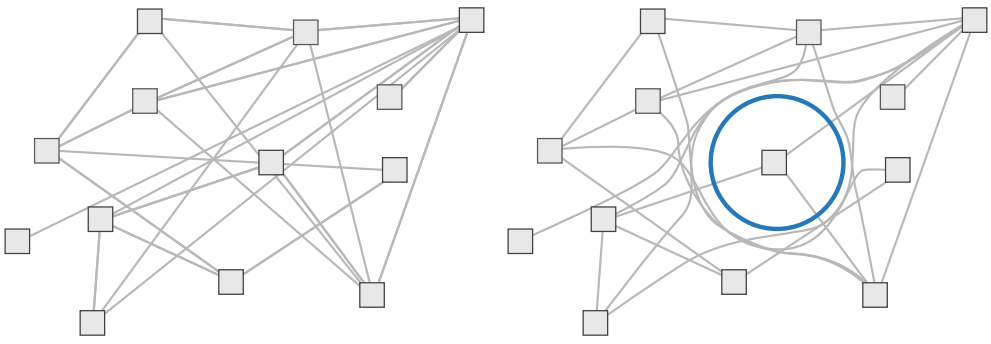

Figure 3.9: A PushLens applied to a node in an edge congested area, showing the connections of this node.

3.6.1 Interaction with PushLens

To avoid confusion in interaction with bundling a PushLens is created by starting a three finger touch technique anywhere on the device. The PushLens is created immediately and visualized by a circle around the three touches. Dragging these touches the lens is moved around and with pinching them the lens can be resized.

Once the creating gesture for the PushLens is released, interaction using less touches is possible. First the lens can be resized by another multi-touch gesture using at least two fingers and the typical pinching gesture; dragging these fingers allows to move the lens as well. The lens can be moved by a single touching finger as well, by starting the touch on the lens border and dragging it.

This touch has to start on the border, because we allow the usage of other techniques inside of the lens; in that way the observer can use plucking for example inside the lens, when cluttering is reduced already. When a lens is not necessary anymore it can be deleted by a short tapping gesture on the border, again this can not be done inside the lens, to provide tapping on nodes inside the lens.

3.6.2 Relevance of the PushLens

The PushLens provides a powerful tool for more detailed graph exploration, when making a PushLens containing a subgraph the subgraph can be analyzed without being disturbed by edges not adjacent to this subgraph.

When analyzing nodes in an area congested by edges, the PushLens can be used to focus a node to reveal the number of edges, which are adjacent. Further the lens allows showing other visual information beneath an edge congested area, which can be geographical information in a route map.

3.6.3 The PushLens as general tool and lens idea

As the PushLens is derived from the EdgeLens [31] approach, there are further lenses on graph imaginable; research has already been done in this field [27] and should be considered, when implementing a real life application for every day usage.

Gestures used for the PushLens should not only be seen as interaction with the lens, but as proposal for general tool interaction. This excludes the creating gesture, if implementing more than one tool. The differentiation in general interaction with graph elements and tools could be the distinction of multi and single touch gestures.

4 Development environment and Implementation

All techniques described in section 3 have been implemented in an example application. This chapter explains the development process and the application design. After describing the development environment and used Frameworks, the way of gesture recognition and interaction is explained. Upcoming problems in this areas and are named. Because most techniques are based on edge manipulation, the drawing of edges is a major issue and explained in detail.

Next to usual Desktop PC running Microsoft Windows™ for programming, I had access to two touch enabled tabletops. A Microsoft Surface ™[28] and a SMART™ table [26] where provided for development and evaluation of the application. Both tables are quiped with a camera sensing infrared light, for touch tracking. As most of these systems, the tables have the same limitations, recognizing false touches and losing the tracking of a fast gesture.

The example dataset for the application was provided in the graphml file format [12]. A XML based file format for graph data. It is supported by several applications and provided with an OpenSource License.

4.1 Frameworks

4.1.1 Visualization Toolkit

There are several visualization toolkits available. As basis for the example application prefuse [14] was chosen. It allows a fast development of a performant node-link visualization applications and smooth animations are easy to implement. Because prefuse is designed for graph visualizations, it provides several libraries for reading graph data, including the graphml file format.

Unfortunately development on prefuse stopped in 2008 and was ported to the flare project [18] based on Flash. Still prefuse is a stable and easy to use framework, while flare is just provided as an alpha version. Like most visualization toolkits today, prefuse does not support touch interaction in its default configuration.

4.1.2 Touch toolkits

Touch devices are a quite young device group; until today there are just few uniform ways to access the touch events like mouse events provided by different drivers. One approach shows the TUIO library, which is a network protocol for sending touch events

based on a network protocol for sound. It is develop by Martin Kaltenbrunner in Austria and is already used in some project concerning touch devices [16, 13]. Unfortunately this protocol is not natively supported by Microsoft Surface™ or SMART™ Table.

To gain access to touch events from both tables, I had access to an unpublished Java™ library, which provides access to them, as well as to events emitted by the TUIO protocol. The accessibility of TUIO events allowed testing the application using the desktop computer and the TUIO Simulator, presented in [17].

The named library provides a listener interface, which can be accessed to receive all touch events emitted from the clients. The touch events provide all data given from the device. This differs for the devices and protocols, but always contains a location and size of the touch. The library provides minor distinction for touch events, as for example detecting a tapping touch or when a touch was dragged, but does not distinct more complex gestures.

4.2 Application architecture

Providing an understandable and easy to extend programming structure the application architecture is approximated to the MVC (Model-View-Controller) software design. The MVC approach is not specified completely and is hard to apply when using new touch recognition frameworks.

Received touches are analyzed in a TouchCalculator, which distinguishes multi from single-touch events and converts all touch events into gesture events. All gesture events are forwarded into a data structure read by an event-controller. This controller has access to the model data and calculates all interaction of events with the underlying data. TouchCalculator and Controller work as a Controller in terms of MVC.

Model and View consist of derivates from prefuse classes, which are extended and adapted to accept gesture events bye the controller. Despite the idea of MVC, the View does not provides the input events, because user interaction works completely on touch library. As described in some MVC approaches the View reads most of the data to visualize out of the model structure. An overview of the application structure is given in figure 4.1.

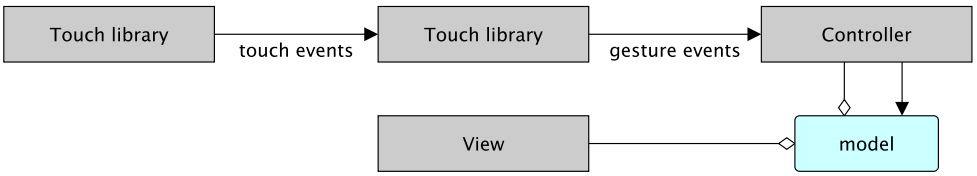

Figure 4.1: The application structure showing most important parts. Light gray shows threads and turquoise the model.

4.3 Gesture Recognition and interaction

One of the goals for application design was using simple and easy to understand gestures. Because gesture recognition of such easy to perform gestures is less complex and do not need a complex framework in background, I go without such a framework. This also saves performance, because not every touch has to by analyzed by complex algorithms.

However it was sill necessary to distinguish multi-touch and single touch events, which was detected using time and location difference of touches. If two or more touches are started near by each other and at nearly the same time they are combined to a multi-touch event, which then was evaluated for itself. This approach has its limitations, when multiple people are working in the same area, but in most cases the system behaves as expected. This is still a major problem in touch recognition; but like the named general problems on camera based touch devices of false and losing touches not in the scope of this work.

4.3.1 Single-Touch Gestures

For interaction just four single touch gesture have to be distinct:

- tapping
- permanent touch
- dragging
- flicking

Tapping is a short gesture, which is not moved and removed in the first 1.5 seconds. A touch staying at one position for longer then the 1.5 seconds is named permanent touch. When the touch is moved in the first 1.5 seconds and released before a long dragging is performed it is a flicking gesture. Longer dragging is labelled as dragging. These gesture states are for general understanding, while the system distinguished more states of a gesture:

- *touched*, a touch just started
- *pressed*, a touch not moved for 1.5 seconds after starting it
- *dragged*, a touch moved in the first 1.5 seconds
- *moved*, a touch moved after being pressed or pushed
- *clicked*, a touch released without moving
- *pushed*, a touch staying at the same location for at least 1.5 seconds after being moved
- *flipped*, a touch moved in the first 1.5 seconds, but not moved a long distance before released

- *released*, a touch which is released from the device surface

An overview in relationship of these states gives figure 4.2 The distinction of pressed and pushed touches is necessary, because of the single-selection, which can only be started by a pressed touch on a node, not by a pushed.

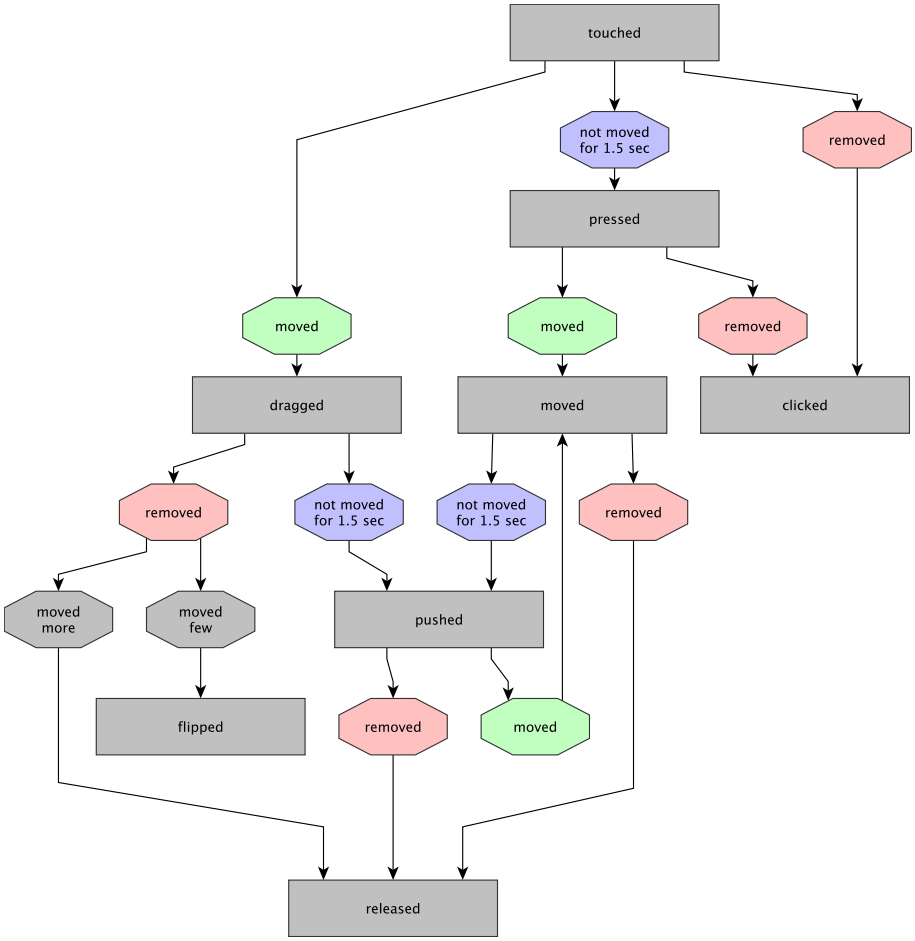

Figure 4.2: Diagram showing the different states assigned to a single-touch gesture. Every gesture starts as "touched" and ends either as "clicked", "released", or "flipped".

4.3.2 Multi touch gestures

A Multi-Touch gesture in my application is defined by three different values. As named in the gesture recognition, the number of touches is important. Further I assign a position to the gesture, which is the evaluated center of all touches. To define a size of the multi-touch gesture, I calculated the distance from this point to every single touch, this I named the radius of the gesture. Position or center and the radius of a multi-touch gesture describe a circle including all touches of this gesture.

Detecting a pinching gesture

Once a collection of touches is assigned as a multi-touch gesture, their radius is constantly controlled and compared to the last known value. If this value differs more then 5% of the original value it is set as a pinching gesture.

Detecting a TouchBundle gesture

Multi touch gestures are only used for interaction with lenses and bundles. Interaction with bundles is performed with a very specific multi finger gesture and only in specific conditions it is applicable for a bundle. That is why a multi touch gesture is tested for the conditions in the first way:

1. the gesture has just two touches
2. the center of both touches is not inside a lens
3. no pinching gesture is recognized so far
4. after first movement of both touches, they move into the same direction, with a variation of less than $\pm 10°$.

If all three conditions are fulfilled the gesture is assigned as a bundling gesture, if one of the tests fails, the gesture is set as lens interaction gesture. All further gesture events, which are pinching and moving events, are send directly to the lens.

4.4 Gesture Interaction

4.4.1 Single-touch interaction

Once a touch gesture is started on an object, the gesture is assigned to the object and the object to the gesture. One gesture can influence multiple objects, while an object can not be influenced directly by multiple gestures. This is due to avoid an unexpected behavior of the application for observing people. Edges however can be influenced by different techniques, which is implicit by multiple gestures, for example using and TouchPlucking and PushLens at the same time.

When starting a single touch gesture, it looks for available elements in the following order:

1. PushBundles
2. PushLenses
3. PushPins
4. Nodes
5. Edges

Ordering the first three in this way is due to the size of intersection space they provide in general, going from less to more. Regarding the "fat finger problem" it appears often that a touch starting on a node intersects with edges as well, otherwise edges usually can be selected easily outside the node area; that is why nodes are privileged before edges.

4.4.2 TouchBundle gesture interaction

Once the gesture is recognized as a TouchBundle interaction edges crossing the line between both touches are tested, if they are along the movement direction of both touches, with a variation of $\pm 30°$, they are added to the bundle. Nodes are tested for intersection with the line between both touches.

While the end of a bundle section is always set to the current center of the gesture, positioning the starting point is more difficult and has to be distinct into two cases. As long as no node has been crossed by the bundle it is set to the center, in the moment the first edge is added to the bundle and can not be changed until the gesture is completely finished.

Once one or multiple nodes are crossed by the bundling gesture, the center of all node positions is calculated. The beginning of the bundle section is set on the line between the center of both touches and this evaluated point, with a maximum distance to the center of the nodes of 30 pixels. This value is set to a specific pixel length, which worked quite well on both devices, because no equation working in all cases was found.

4.4.3 PushLens gestures

A multi touch gesture for the PushLens has to be tested if a new PushLens should be created or changes are made to an existing one. An existing lens is assigned to this gesture if the gesture position is located in an existing lens. The lens must not be assigned to another gesture. If no existing lens is assigned, a new one is created for this gesture. When a new lens is created the radius and position are set to the analog values for the gesture.

Changes applied to an exiting edge are not set directly to radius and position of the gesture, but relative to the last change of the gesture. This is, when the radius of a gesture is doubled the radius of the lens is doubled as well, according to the radius the lens hat before.

4.5 Drawing Curved Edges

As the application is focused on edge interaction and all techniques changes the shape of edges in some way, it is necessary to have a proper basis for drawing shapes. Although prefuse supports bended curves in by using bezier curves, it does not support multiple curves.

I worked on two approaches, one supported by subdivision splines and the second by bezier curves. The bezier curves worked much more sufficient in several cases. This let

me drop the subdivision approach, nevertheless I explain both approaches in this section and name advantages and disadvantages. Starting with subdivision splines, I explain all algorithms used for drawing edges in the application.

4.5.1 Subdivision splines

The idea of subdivision splines is to subdivide the lines of a curve into shorter lines and illustrated in figure 4.3. The number of subdivisions can be set variable, which allows a control of performance in comparison with quality of the visualization.

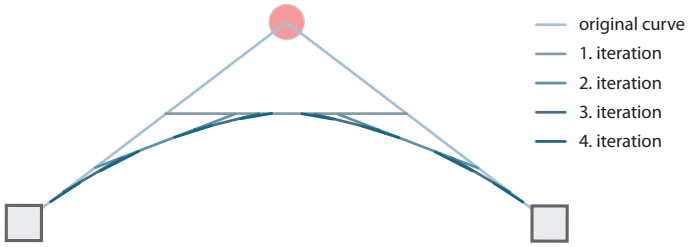

Figure 4.3: The idea of a subdivision spline applied to an edge plucked by a touch. The original curve is shown brightly and the subdivision iterations are drawn darker. The final curve goes not through the touching finger.

As can be seen in figure 4.3, the original algorithm does not let the line go through the touching finger, because of the first dividing iteration, which flattens the curve. I solved this problem, with adding a segment which has the position of the touch, is slightly longer and parallel to the line between both nodes. Because this segment is just made smaller but never completely removed ore repositioned, as shown in figure 4.4 the curve of the edge always goes through the position of the touch.

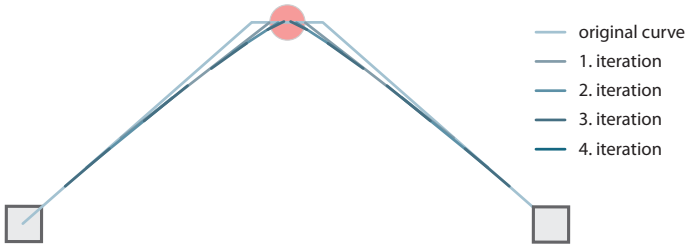

Figure 4.4: Improved subdivision curve for TouchPlucking. The curve follows the touch, achieved by a small line segment at the touch position.

This approach worked quite well, but in several tests subdivision splines showed another problems Calculating the curves with more iteration, to gain better quality, only a few edges could be plucked otherwise the number of frames per second decreased very fast. This problem got even worse, when I started to animate the bending of edges for TouchStrumming.

4.5.2 Bezier Curves

When facing the problems in performance using subdivision splines, I figured a native implemented shape would solve this in a easy way. In Java such a given native shape are the quadratic and cubic bezier curve. The bezier curve has another advantage. It is a real bend curve and not approximated as the subdivision spline.

The idea of a bezier curve is based on is a parametric curve, where the cubic curve is described with two ending points and two control points, as shown in figure 4.5. The curve starts from the endings always going into the direction of the control points.

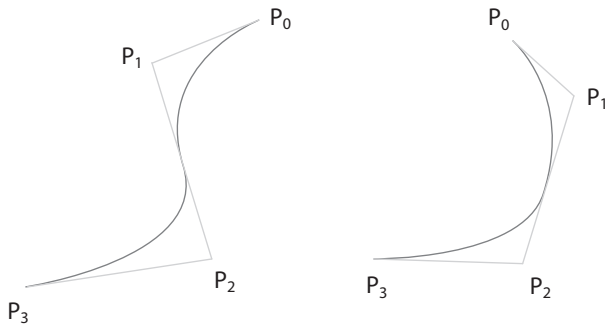

Figure 4.5: Two bezier curves, illustrating the possibilities of drawing S- and C-shapes. Endings of the Bezier curve are named P_0 and P_3; control points are P_1 and P_2.

After some testing with bezier curve, the performance showed much better, but some other problems had to be solved. The original shape class in Java has a strange intersection function, which returns true not only, when the intersection is in the curve, but always if there is an intersection with the convex hull of the control and ending points. Another problem is the description of the bezier curves, which is based on the two control points, but not necessarily goes through them. Solving these problems is explained in the following sections.

Intersection with other elements

Despite to subdivision splines I my application did not calculated the shape for the parametric for itself, leaving this to Java. For intersection testing I use an approximation of the bezier curve, calculating points on the curve and testing the lines between those points for intersection with the other object.

Plucked Edge

Using bezier splines and setting the touch position for the control points, I faced the same problem as for the original subdivision splines. The spline does not followed the touch, illustrated in figure 4.6. To correct this error I hat to go into the bezier formula and calculate the current control points out of it.

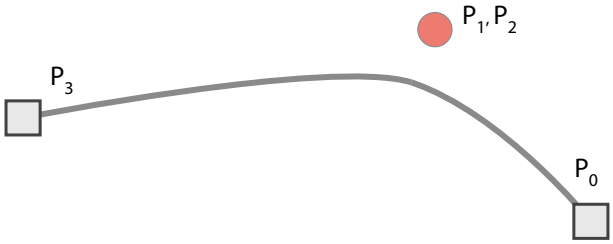

Figure 4.6: When using the touch position for the control points, the curve of the edge does not goes through the touch.

The original formula for the cubic bezier curve is

$$B(t) = (1-t)^3 * P_0 + (1-t)^2 t * P_1 + (1-t)t^2 * P_2 + t^3 * P_3$$

With P_0 and P_3 as ending positions, P_1 and P_2 as control points and $B(t)$ as point on the curve according to the parameter t, with $0 < t < 1$. To solve the given problem I took this formula and calculated the position for the control points, in a way that the curve goes through the touch position P_T. Setting both control points to the same position, $P_1 = P_2$, and the touch position as the result of the equation, $P_T = B(t)$, the formula can be reorder to:

$$P_1 = \frac{P_T - (1-t)^3 * P_0 - t^3 * P_3}{3 * (t - t^2)}$$

To gain a Point for the control point P_1 the parameter t has to be set to a specific value. My first implementation used with $t = 0.5$ as a static value; and worked quite well. The result is shown in figure 4.7.

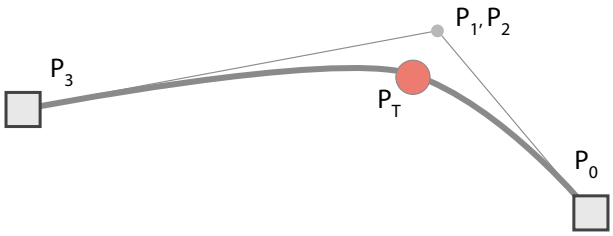

Figure 4.7: Drawing a bended edge using the bezier curve approach. The edge goes through the touch position after calculating the control points.

Later I developed an equation for t, according to the distance to both endings of the curve:

$$t = 0.5 + 0.4(\frac{|P_T\vec{P_0}|}{|P_T\vec{P_0}| + |P_T\vec{P_3}|}) - 0.2$$

Which calculates the value of t in the borders $0.3 < t < 0.7$ and provides a more natural shape of the curve near to the endings of the curve, as shown in figure 4.8.

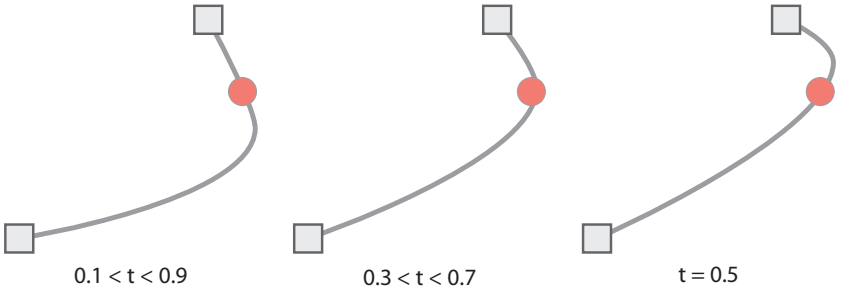

Figure 4.8: The bending of a curve, when the plucking touch is near an adjacent node. Different values for t, showing different shapes of the curve. The best results where achieved using the example in the middle.

Animation of the strumming

The curve for the strumming are drawn in the same way as for the plucking, only the control point is not calculated out of the position of a plucking touch, but a point which position is calculated as a continuous animation.

For the direct and indirect strumming, I took a maximum amplitude a, defined by the length of the strumming vector $\vec{sv}$ which is calculated as the doubled vector between P_T, the last position of the touch and P_P, the point on the straight edge curve, with proportional distances to the endings of the edge as P_T:

$$P_P = P_0 + \vec{P_0P_3} * \frac{|\vec{P_TP_0}|}{|\vec{P_TP_0}| + |\vec{P_TP_3}|}$$

With P_0 and P_3 are the endings of the edge. The strumming vector is $\vec{sv} = 2 * P_P - P_T$ and the amplitude a is $a = |\vec{sv}|$. This amplitude is minimized linear based on the left time before the strumming ends, to provide the sense of a real string, where the strumming decreases over time as well; the duration of the strumming is set depending of the initial value of the amplitude a.

The actual position of the strumming point P_S is computed with:

$$P_S = P_T\vec{P_P} * ca * cos(step * \pi * 1.5 * wav)$$

Where ca is amplitude decreasing over time and $step$ is an over time linear increasing value in the interval $0 < step < 1$. This lead to $ca = a * (1 - step)$.

When nodes are strummed the strumming of edges follows a curve itself, going from the strummed node to the adjacent one. This movement is achieved by adding a movement along the edge to the movement of P_S explained above; in this case the amplitude is not decreased. The strumming does not moves in the direction of the vector from P_T to the node, but orthogonal to the edge. With P_{SN} as the source node, which had been strummed and P_{TN} as target node, the strumming moves along the vector $\vec{eo} \perp P_{SN}\vec{P_{TN}}$. P_S is calculated by:

$$P_S = (P_{SN} + P_{SN}\vec{P_{TN}} * step) + \vec{eo} * a * cos(step * \pi * 1.5 * wav)$$

The curve the strumming point P_S describes a cosines function along the edge, where the edge is the abscissa. wav defines the number of wave it describes and a the amplitude, which is the greatest distant to the edge.

Intersection and interaction with a PushLens

Once an EdgeLens is created I start the intersection control for edges. First I use the general java shape function and if there is an intersection, I test if any node of the edge is in the lens or maybe the touch or pin, bending the edge currently. Because I can not directly test for intersection of the bezier curve with the circular shape of the EdgeLens I first approximate the cubic curve again and test the single points for being inside the shape of the lens, if at least one point is inside and one outside, I assign the lens to the edge. The idea of drawing an edge around a lens is shown in figure 4.9, where important points are named. The algorithmic approach is explained int the following paragraph.

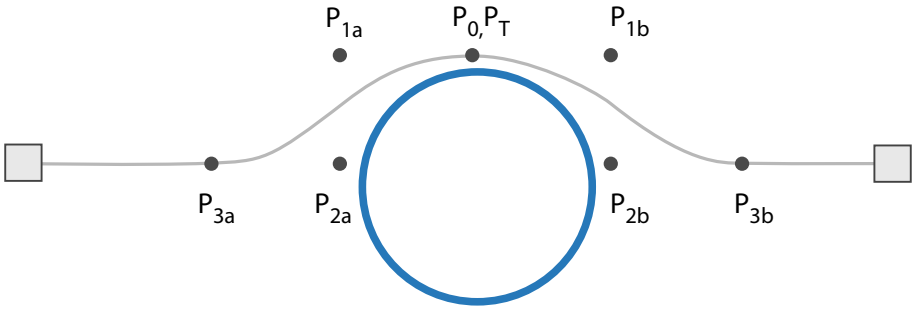

Figure 4.9: The bending of an edge evoked by a PushLens and using bezier. P_{3a}, P_{3b} and P_0, P_T are endings of the bezier curves. The control points are named P_{1a}, P_{2a}, P_{1b}, P_{2b}.

When a lens intersects with an edge, even if this edge is bend or not, I calculate the two intersection point, named P_{2a} and P_{2b}, with a slightly addition to the radius of the lens. Between P_{2a} and P_{2b}, the edge is not drawn with the original shape, but with two bezier splines, which approximate the shape of the lens. Both splines are drawn out of four points. The first control points for each spline, named P_{3a} and P_{3b}, are located on the line defined by P_{2a} and P_{2b} but going further away from the lens, one to each side. The next point for a spline is computed from a vector $\vec{a}$ which is orthogonal to $\vec{P_{2a}P_{2b}}$. As the circle of the edge lens is divided into two segments of different size by $\overline{P_{2a}P_{2b}}$, $\vec{a}$ points away from the $\overline{P_{2a}P_{2g}}$ in the same direction as the smaller segment of the circle. The length of $\vec{a}$ is defined by the height of the smaller segment with addition of a smaller value, so the edge does not touches the lens directly. Further points for the splines named P_{1a}, P_{1b} and P_0 are calculated out of the given points and $\vec{a}$:

$$P_{1a} = \vec{a} + P_{2a}$$
$$P_{1b} = \vec{a} + P_{2b}$$

$$P_0 = \frac{P_{1a} + P_{1b}}{2}$$

With these points computed, the bezier splines can be drawn using the point $P_0, P_{1a}, P_{2a}, P_{3a}$ and $P_0, P_{1b}, P_{2b}, P_{3b}$ as shown in figure 4.9.

Computing the curve of an TouchBundle

The bundle itself is first drawn as straight line going from one bundle end to the other. For the calculation of edge drawing these bundle ends are named P_{0a} and P_{0b}. From these bundle endings every edge is drawn as a bezier spline to the connected node.

The spline from the bundle end to the nodes starts either in P_{0a} or P_{0b}, for the calculation is the same on each side we take P_0 as one end of the spline. The second ending is the position of the node and named P_3. Control point P_2 is the middle of P_0 and P_3: $P_2 = \frac{P_0+P_3}{2}$.

For calculation of the second control point we need a vector $\vec{ab}$ from one bundle end to the other, $\vec{ab} = P_{0a}\vec{P}_{0b}$. This vector is shortened and added to the bundle ends for computing the control points P_1.

$$P_{1a} = P_{0a} - \vec{ab}$$

and

$$P_{1b} = P_{0b} + \vec{ab}$$

The control points are outside the bundle itself, but on the same line. This way the edge seems as going out of the bundle line. The curve of one edge from a bundle to a node is illustrated in figure 4.10. Drawing from the other end of the bundle to the other nodes works the same.

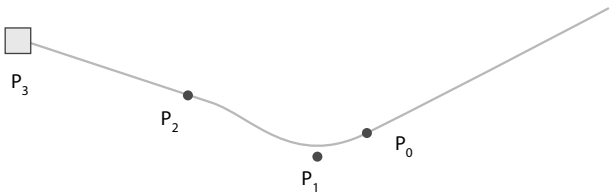

Figure 4.10: Showing the bezier curve for a single edge from the end of a bundle (P_0) to the node.

As TouchBundles as well are enabled for plucking, although I had to change the way of illustration. When an edge is plucked it ends into two nodes and it does not matter in which slope the curve goes into the nodes. The curve of a plucked bundling ends in P_{0a} and P_{0b} and has to have the same slope at this points as the original straight line. Otherwise the visual curve of the edges would break at this point.

To solve this problem I used a similar approach as for the curves of an edge affected by an EdgeLens. The curve of a plucked bundle consists of two separate curve, where the position of the touch is one end both splines. On control point P_2 calculated

$$P_{2a} = P_T - \vec{ab} * 0.2$$

for the one curve and

$$P_{2b} = P_T - \vec{ab} * 0.2$$

for the other. Where P_T is the position of the touch plucking the bundle and on the same position as P_3. $\vec{ab}$ is the vector from P_{0a} to P_{0b} as explained before. The third point is calculated by

$$P_{1a} = P_{0a} + \vec{ab} * 0.2$$

for the first and

$$P_{1b} = P_{0b} - \vec{ab} * 0.2$$

for the second curve. The fourth point is either P_{0a} or P_{0b}. This calculation is illustrated in figure 4.11

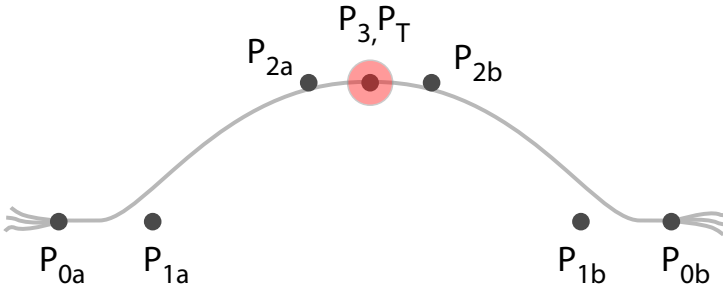

Figure 4.11: The plucking of bundle, using two bezier splines. P_{0a} and P_{0b} are the endings of the bundle.

5 Conclusion and Future Work

5.1 Conclusion

The presented work provides a set of new multi-touch interactions on node-link diagrams exploration and analysis. Not all techniques are completely new, but transferred from common desktop interfaces with single-point devices to multi-touch interactions. None of the known techniques has been simultaneously used with another.

This concurrent usage of techniques gives greater power to the techniques themselves. This shows the advantages of multi-touch devices, where people are able to perform different gesture for different techniques at the same time; usage of different techniques at the same time, is hardly imaginable on todays mouse based single point systems.

All techniques are focused on edge manipulation, which has been slightly ignored in research so far. However the application shows the capabilities for edge manipulation, which is located especially in strong interconnected graphs; an upcoming field, regarding the increasing number of social and communication networks.

Still, the application shows that node interaction can not be ignored and even regarding edge only interaction, interaction with nodes is often a basis for these technique, compare the single-edge selection techniques 3.2.1.

The used gestures are simple and easy to remember, which shortens the training period and allows a fast and intuitive interaction. However this approach has its limitations, when including gestures and techniques in other applications, where a gesture might be used for different interaction purposes.

Some of the proposed interaction techniques might not only be applicable to the domain of graph interaction, but in other domains as well. Lenses are used in a wide variety of visualizations, as well as distortion oriented techniques, where the used interaction gestures can be used in similar ways. A completely new approach is the TouchStrumming technique, using a touch gesture performing a natural behavior of elements. This technique could possibly applied to a huge number of other domains, text editor where strumming a word highlights all similar words as well, a feature often useful when writing longer texts.

5.2 Future work

There exists several approaches for future development of the given work. Adding further techniques for editing the graph structure in adding and deleting nodes and edges; for this approach the complete gesture set has to be reconsidered and adapted. For there is already an evaluation of gesture requested by users [7], this approach seems suitable,

even when using a change in mode is necessary.

Node interaction has been researched quite well, but this neither applies to node interaction techniques on touch enabled devices nor the combination of node and edge interaction in simultaneous interaction. I think, especially the second, is underdeveloped because of limitations in given user interfaces and should be further looked at, when developing new techniques.

The provided PushLens is just a small example of lenses for graph exploration, where a wide variety exists [27]. Where this work shows the advantages of usage in multiple lenses of one kind at the same time, this is a field where further advantages can be imagined. Usage of lenses in augmented reality [25, 24] for graph exploration is a further field of development.

Although there has been some positive feedback for the presented application, there was no scientific evaluation which approves the benefits of the given techniques and gestures.

Bibliography

[1] Anand Agarawala and Ravin Balakrishnan. Keepin' it real: pushing the desktop metaphor with physics, piles and the pen. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 1283–1292. ACM, New York, NY, USA, 2006. ISBN 1-59593-372-7.

[2] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998. ISBN 0133016153.

[3] W. Buxton. Multi-touch systems that i have known and loved. http://www.billbuxton.com/multitouchOverview.html (17. July 2010). URL `http://www.billbuxton.com/multitouchOverview.html`.

[4] W. Buxton and B. Myers. A study in two-handed input. In *CHI '86: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 321–326. ACM, New York, NY, USA, 1986. ISBN 0-89791-180-6.

[5] Josep Díaz, Jordi Petit, and Maria Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002. ISSN 0360-0300.

[6] Peter Eades and Roberto Tamassia. Algorithms for drawing graphs: An annotated bibliography. Technical report, Brown University, Providence, RI, USA, 1988.

[7] Mathias Frisch, Jens Heydekorn, and Raimund Dachselt. Investigating multi-touch and pen gestures for diagram editing on interactive surfaces. In *ITS '09: Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, pages 149–156. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-733-2.

[8] Mathias Frisch, Jens Heydekorn, and Raimund Dachselt. Diagram editing on interactive displays using multi-touch and pen gestures. In *Proc. Diagrams '10*. Diagrams '10, 2010.

[9] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, 21(11):1129–1164, 1991. ISSN 0038-0644.

[10] G. W. Furnas. Generalized fisheye views. In *CHI '86: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 16–23. ACM, New York, NY, USA, 1986. ISBN 0-89791-180-6.

[11] Kurtenbach G. and Hulteen E.A. Gestures in human-computer communication. *The Art of Human-Computer Interface Design*, pages 309–317, 1990.

[12] GraphML Working Group. The graphml file format. http://graphml.graphdrawing.org/ (14 July 2010). URL `http://graphml.graphdrawing.org/`.

[13] Nui group. Community core vision. http://ccv.nuigroup.com/ (14 July 2010).

[14] Jeffrey Heer, Stuart K. Card, and James A. Landay. prefuse: a toolkit for interactive information visualization. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 421–430. ACM, New York, NY, USA, 2005. ISBN 1-58113-998-5.

[15] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12: 741–748, 2006. ISSN 1077-2626.

[16] Martin Kaltenbrunner. reactivision and tuio: a tangible tabletop toolkit. In *ITS '09: Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, pages 9–16. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-733-2.

[17] Martin Kaltenbrunner and Ross Bencina. reactivision: a computer-vision framework for table-based tangible interaction. In *TEI '07: Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 69–74. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-619-6.

[18] UC Berkeley Visualization Lab. The flare visualization toolkit. http://flare.prefuse.org/ (14 July 2010).

[19] Bongshin Lee, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete, and Nathalie Henry. Task taxonomy for graph visualization. In *BELIV '06: Proceedings of the 2006 AVI workshop on BEyond time and errors*, pages 1–5. ACM, New York, NY, USA, 2006. ISBN 1-59593-562-2.

[20] Tomer Moscovich. Multi-touch interaction. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 1775–1778. ACM, New York, NY, USA, 2006. ISBN 1-59593-298-4.

[21] Tomer Moscovich and John F. Hughes. Multi-finger cursor techniques. In *GI '06: Proceedings of Graphics Interface 2006*, pages 1–7. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 2006. ISBN 1-56881-308-2.

[22] Tomer Moscovich and John F. Hughes. Indirect mappings of multi-touch input using one and two hands. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1275–1284. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-011-1.

[23] Manojit Sarkar and Marc H. Brown. Graphical fisheye views of graphs. Technical report, Brown University, Providence, RI, USA, 1991.

[24] Martin Spindler and Raimund Dachselt. Exploring information spaces by using tangible magic lenses in a tabletop environment. In *CHI EA '10: Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, pages 4771–4776. ACM, New York, NY, USA, 2010. ISBN 978-1-60558-930-5.

[25] Martin Spindler, Sophie Stellmach, and Raimund Dachselt. Paperlens: advanced magic lens interaction above the tabletop. In *ITS '09: Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, pages 69–76. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-733-2.

[26] Philipp Steurer and Mani B. Srivastava. System design of smart table. In *PERCOM '03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, page 473. IEEE Computer Society, Washington, DC, USA, 2003. ISBN 0-7695-1893-1.

[27] Christian Tominski, James Abello, Frank van Ham, and Heidrun Schumann. Fisheye tree views and lenses for graph visualization. In *IV '06: Proceedings of the conference on Information Visualization*, pages 17–24. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2602-0.

[28] Josh Wall. Demo i microsoft surface and the single view platform. In *CTS '09: Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems*, pages xxxi–xxxii. IEEE Computer Society, Washington, DC, USA, 2009. ISBN 978-1-4244-4584-4.

[29] Jacob O. Wobbrock, Meredith Ringel Morris, and Andrew D. Wilson. User-defined gestures for surface computing. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 1083–1092. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-246-7.

[30] Nelson Wong and Sheelagh Carpendale. Supporting interactive graph exploration using edge plucking. In *Proceedings of IS&T/SPIE 19th Annual Symposium on Electronic Imaging: Visualization and Data Analysis*. SPIE and IS&T, Bellingham, WA and Kilworth Lane, VA, USA, 2007.

[31] Nelson Wong, Sheelagh Carpendale, and Saul Greenberg. Edgelens: An interactive method for managing edge congestion in graphs. *Information Visualization, IEEE Symposium on*, 0:7, 2003. ISBN 0-7695-2055-3.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Studienarbeit in allen Teilen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel genutzt habe. Alle wörtlich oder sinngemäß übernommenen Textstellen habe ich als solche kenntlich gemacht.

Magdeburg, den 19.7.2010

Sebastian Schmidt