Projektarbeit

Auseinandersetzung mit einer Multitouch-Hardware mit anschließender Konzeption und Umsetzung einer plattformunabhängigen und synchronisierbaren Anwendung zum kollaborativen Arbeiten

Lars Borchert

Matrikel-Nr.: 849041

Institut für Medienpädagogik

Christian-Albrechts-Universität zu Kiel

SS 2012/2013

Inhaltsverzeichnis

Überblick

Meine Arbeit bestand darin, die zur Verfügung gestellte Hardware in Form eines Multitouch-fähigen Bildschirmes mithilfe entsprechender Software anzusprechen und die daraus resultierenden Möglichkeiten durch ein selbst entwickeltes Programm zu demonstrieren.

Hardware

Die von mir getestete Hardware besteht aus einem handelsüblichen LCD-Bildschirm sowie einem an diesem Bildschirm befestigten Rahmen, der Berührungen mit den Fingern (oder anderen Zeigegeräten) erkennen und an den angeschlossenen Computer weitergeben kann. Zum Rahmen gehört ebenfalls eine Glasplatte, die über dem Bildschirm liegt, sodass man beim Arbeiten nicht direkt die empfindliche Bildschirmoberfläche berührt, sondern lediglich auf der Glasfläche arbeitet. Ein Touch-Event wird zudem durch den umliegenden Rahmen auch ohne tatsächliche Berührung der Glasfläche als solcher erkannt. Durch die verwendete Infrarot-Technologie werden die Finger oder anderweitige Zeigegeräte bereits kurz vor der eigentlichen Berührung korrekt analysiert und verarbeitet. Die Rahmenvariante der Generation G3 mit einer Größe von 27 Zoll ist in der Lage, bis zu 12 parallele Touch-Events zu registrieren und zu verarbeiten. Eine deutlich größere Variante des Bildschirmes inklusive Sockel zum gleichzeitigen Arbeiten mit mehreren Personen soll ebenfalls mit der Software betrieben werden können. Bei diesem Gerät können bis zu 32 gleichzeitige Eingaben über Finger oder andere Zeigegeräte aufgenommen werden.

Installation

Das Anbringen des Rahmens an den Bildschirm wird mittels der mitgeliefertem Klebepads realisiert. Es wäre auch ein Anwendungsszenario denkbar, in dem der Rahmen mit der Glasplatte getrennt von dem Ausgabegerät - also dem Monitor – verwendet werden könnte. Beispielsweise könnten Gesten auf der Glasfläche, die auf einem Tisch liegt, durchgeführt werden und vom Rahmen aufgenommen werden, während die visuelle Darstellung an einem Monitor an der Wand wiedergeben wird. In diesem Fall wäre dieses Szenario jedoch unpraktisch, da eine direkte visuelle Rückmeldung für den Benutzer unterhalb der Touch-Oberfläche erforderlich ist.

Der Rahmen wird per USB-Kabel an einem Computer angeschlossen, eine zusätzliche Stromversorgung ist nicht erforderlich.
Auf der deutschen Homepage des Herstellers findet man Treiber für aktuelle Betriebssysteme wie Windows 7, Windows 8, Mac OS X und Linux. Zu Beginn meiner Projektarbeit waren funktionsfähige Treiber für Windows 8 dort noch nicht vorhanden, ließen sich aber nach etwas Recherche im Internet auf einer anderen Website finden.
Zum Treiber gehört unter Windows 7 bzw. 8 eine Software, die automatisch gestartet wird und über die man einige Einstellungen vornehmen kann. So lässt sich der Touchscreen neu kalibrieren, eine Diagnose bei Fehlern starten sowie Einstellungen zur Unterstützung von zusätzlichen Übertragungsprotokollen oder Betriebssystem-spezifischen Eigenschaften vornehmen.

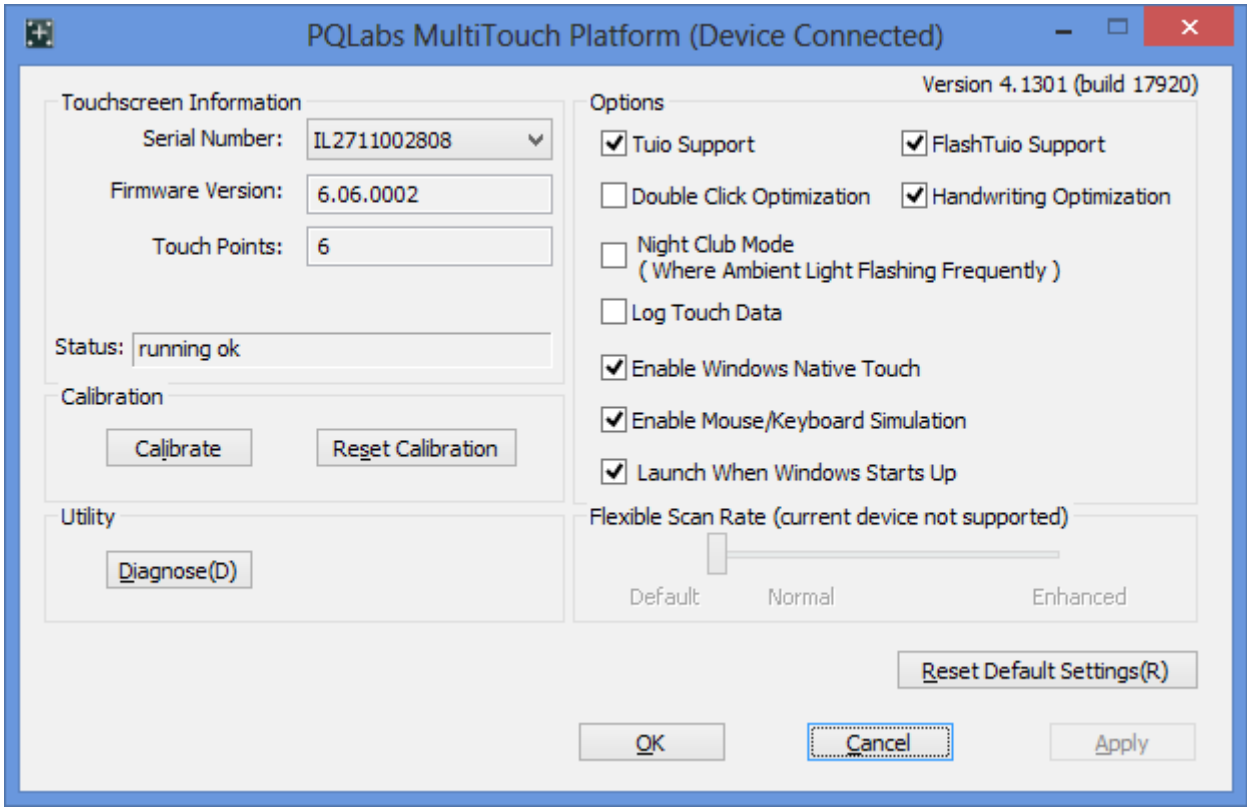

Abbildung 1: Konfigurations-Software unter Windows 8

Testphase

Zunächst testete ich die Funktionsweise unter Microsoft Windows 7. Die Unterstützung von mehreren Touches wird von Windows 7 bereits nativ unterstützt. Das bedeutet, dass Programme, die mehrere gleichzeitige Berührungen verarbeiten können, mit dem mitgelieferten Treiber bereits automatisch benutzt werden können. Ein einfaches Beispiel wäre das zum Betriebssystem gehörige Malprogramm Paint, welches über den Touch-Bildschirm ohne zusätzliche Software mit mehreren gleichzeitigen Berührungen gesteuert werden kann.

Die neueste Version von Microsoft Windows 8 besitzt ebenfalls ab Werk eine Unterstützung für Multitouch. Gerade die Tatsache, dass bei der Entwicklung von Windows 8 die Unterstützung von Touch-gesteuerten Tablets stark im Vordergrund stand, lässt erwarten, dass die Multitouch-Funktionalität bei diesem Betriebssystem sehr gut umgesetzt worden ist.

Konzeption

Als Anwendungsbeispiel habe ich mir eine Software überlegt, die das gleichzeitige Arbeiten gut darstellen kann. Mein Ziel war die Präsentation von unterschiedlichen Medientypen wie Bildern, Videos, PDFs und Text-Dateien, die über eine übersichtliche Oberfläche präsentiert werden sollen.

Die Software sollte folgende Funktionalitäten bieten:

- Öffnen von verschiedenen Medienarten wie Bildern, Videos, PDFs, Text-Dateien:
  - per Datei-Auswahldialog
  - per Drag & Drop durch Ziehen von Dateien in das Programmfenster.
- Verschieben, Drehen und Skalieren der geöffneten Medien mittels bekannter Touch-Gesten.
- Bearbeiten/Benutzen bestimmter Medien:
  - Abspielen von Videos.
  - Bearbeiten von Texten.
- Eine Synchronisation über mehrere Geräte in Form einer Server-Client-Architektur
- Auch ohne verfügbaren Server sollen alle Programmfunktionalitäten zur Verfügung stehen

Umsetzung

Mein erster Prototpy war eine native Windows 8 App, die ich mithilfe des kostenlosten Programms Microsoft Visual Studio 2013 umgesetzt habe. Ich hatte mich für diese Art der Software entschiedenen, da sie mehrere Vorteile bietet: Zum einen hat man direkten Zugriff auf alle in Windows 8 verfügbaren Touch- und Gesten-Schnittstellen, sodass eine zuverlässige Erkennung und Verarbeitung der Benutzereingaben möglich ist. Weiterhin lassen sich Windows 8 Apps mit Unterstützung von modernen und weit verbreiteten Technologien wie HTML 5, JavaScript und CSS 3 realisieren. Diese Techniken lassen sich innerhalb der Windows-8-spezifischen Entwicklungsumgebung direkt benutzen und werden außerdem noch von zusätzlichen Windows 8 Bibliotheken ergänzt (u.a. die im vorherigen Kapitel erwähnten Touch-Bibliotheken).
Nachteilig dagegen ist jedoch die Einschränkung auf das Betriebssystem Microsoft Windows 8, da die zusätzlichen Bibliotheken zur Gesten-Erkennung nicht für andere System verfügbar sind, sondern fest im Windows 8 Betriebssystem verankert sind.

Nachdem ein eingeschränkt funktionsfähiger Prototyp fertig war, habe ich diesen vorgestellt und es wurden weitere Möglichkeiten diskutiert. Die Einschränkung auf Windows 8-Hardware aufgrund der verwendeten Technologien sollte wenn möglich aufgehoben werden und zudem sollte eine Möglichkeit zur Synchronisation zwischen mehreren Geräten untersucht werden.

Ich habe mich daraufhin für die Umsetzung des Programms als eine rein native HTML 5-Version entschieden, um die Anzahl der kompatiblen Geräte zu maximieren, da HTML und Javascript unabhängig vom verwendeten Betriebssystem in allen modernen Browsern lauffähig sein würde.
Der kritische Teil bei dieser Implementierung ist allerdings der verwendete Browser. Die aktuellen Betriebssysteme können mehrere Touch-Eingaben verarbeiten. Die Unterstützung durch die Browser ist allerdings unterschiedlich weit vorangeschritten. Gleiches gilt für die Entwicklung zuverlässiger Touch- & Gesten-Bibliotheken in der Javascript-Community.

JavaScript

Ich habe umfangreiche Recherchen zum Thema Gesten-Erkennung und Multitouch-Fähigkeit und deren Umsetzung mittels frei verfügbarer Javascript-Frameworks durchgeführt. Dabei habe ich nach der besten und zuverlässigsten Lösung gesucht, was sich jedoch als schwieriger herausstellte als vermutet. Viele Entwicklungen sind noch nicht fertiggestellt, da gerade im relativen jungen Bereich 'Multitouch' noch viel bei den Browser-Herstellern geändert wird oder aber auch die Entwickler von Hardware und Anwendungssoftware verschiedene Wege gehen. Ich habe mich aufgrund der Tatsache, dass ich eine Web-Anwendung basierend auf HTML und Javascript geplant habe, bei den Recherchen hauptsächlich auf Javascript-Lösungen bezogen. Zusätzlich habe ich mir aber auch noch bekannte Protokolle wie TUIO, Microsofts Pointer-Konzept und weiterführende Literatur angesehen.
Besonders das Pointer-Konzept von Microsoft, dass ja bereits intern in Windows 8 für eine gelungene Gestensteuerung sorgt, macht einen guten Eindruck, ist allerdings noch nicht als offizieller Standard abgesegnet und daher auch lediglich nur in Microsofts eigenen Internet Explorer ab Version 10 implementiert.

Im Folgenden eine Liste der Javascript-Bibliotheken und -Frameworks, die ich während meiner Recherche getestet habe:

- Caress.js http://caressjs.com/
- DeepTissue.js http://deeptissuejs.com/
- Ember Touch https://github.com/emberjs-addons/ember-touch/
- Fabric.js http://fabricjs.com/
- Fidget http://www.simonboak.co.uk/fidget/
- Hammer.js http://eightmedia.github.io/hammer.js/
- Jester https://github.com/plainview/Jester
- jGestures http://jgestures.codeplex.com/
- jQuery Touch http://www.midemos.com/demos/iphone/touch/

- jQuery Touch Punch http://touchpunch.furf.com/
- jQuery UI multiple draggable http://www.myphpetc.com/2009/11/jquery...
- jquery-ui-for-ipad-and-iphone http://code.google.com/p/jquery-ui-for-ipad...
- jquery.event.drag http://threedubmedia.com/code/event/drag
- jQuery.Pep http://pep.briangonzalez.org/
- PinPanZoom https://github.com/ssweriduk/PinchPanZoom
- pointer.js https://github.com/borismus/pointer.js
- QuoJS http://quojs.tapquo.com/
- SenchaTouch http://www.sencha.com/products/touch
- Touchable https://github.com/dotmaster/Touchable-jQuery...
- Touchy http://touchyjs.org/
- TouchSwipe http://labs.rampinteractive.co.uk/touchSwipe/de...
- WKTouch https://github.com/alexgibson/WKTouch

Die aufgelisteten Projekte haben meistens unterschiedliche Ziele und unterscheiden sich zum Teil auch sehr stark in ihrem Umfang (einige Bibliotheken sind z.B. nur in Bezug auf ein bestimmtes Gerät, einen bestimmten Browser oder eine bestimmte Geste entwickelt worden). Zudem werden einige Scripte bereits gar nicht mehr weiterentwickelt – der Grund dafür ist oft unbekannt. Eine interessante Liste mit noch mehr Bibliotheken findet man außerdem hier: https://github.com/bebraw/jswiki/wiki/Touch.

Neben diesen untersuchten Programmen habe ich mich außerdem noch mit der Simulation von Touch-Eingaben beschäftigt, um eine eingeschränkt funktionsfähige Version des Programms auch mit herkömmlichen Eingabemethoden (also z.B. einer Maus oder einem touchfähigen Gerät mit lediglich einem gleichzeitigen Touch-Event) nutzen zu können. Auch hier gibt es unterschiedlichste Ansätze und Umfänge von Scripten. So gibt es z.B. Javascript-Lösungen, die Mausklicks als Touches an den Browser weitergeben (oder auch umgekehrt, falls erwünscht). Weiterhin können sogar mehrere Touches simuliert werden, indem z.B. eine zusätzliche Taste auf der Tastatur gedrückt gehalten wird.

Ein anderer Ansatz wird durch auf dem Computer installierte Programme verfolgt, die entweder als Browser-Plugin laufen oder als ein zusätzliches Programm, das beim

Hochfahren des PCs mitgeladen wird. Diese können dann die vom Betriebssystem erkannten Mausklicks als Touch-Events an ein anderes Programm, wie z.B. einen Browser, übergeben werden.

Zu guter Letzt gibt es Software für mobile Geräte, die Berührungen auf dessen Bildschirm über WLAN an andere Geräte weitergeben (z.B. an einen PC mit angeschlossenem Monitor ohne Touch-Funktionalität) und dort die Touches darstellen können. So lässt sich z.B. mittels eines iPads oder iPhones eine multitouchfähige Software auf einem normalen PC steuern, sofern dessen Betriebssystem mehrere gleichzeitige Eingaben unterstützt.

Letztendlich habe ich mich bei der Entwicklung meines Programms für die Verwendung des Javascript-Frameworks 'Hammer.js' entschieden, welches zur Zeit aktiv vorangetrieben wird und wie die meisten Bibliotheken in diesem Bereich Open Source ist. Es werden prinzipiell alle Gesten unterstützt und funktionieren außerdem in den bekanntesten Browsern zuverlässig.
Der Umfang an Dokumentation und Beispielen ist dagegen sehr gering, dies wird sich vermutlich aber noch in der nächsten Zeit verbessern. Außerdem überlässt dieses Framework die Umsetzung der Gesten zum größten Teil dem Anwender, statt konkrete fertige Abläufe vorzugeben. Das bedeutet, dass Hammer.js lediglich dafür sorgt, dass die Berührungen auf der Touch-Oberfläche mit allen zur Verfügung stehenden Informationen geordnet aufbereitet erkannt und weitergeben werden. Die Verarbeitung zu Gesten wie etwa das Verschieben oder Drehen eines Elementes sind daher noch mit entsprechender Mathematik umzusetzen.

Trotzdem ist die Vorarbeit des Scriptes eine hilfreiche Unterstützung, da hierdurch z.B. auch die Eigenarten der verschiedenen Browser bereits berücksichtigt werden. Auf der nächsten Seite ist die Kompatibilität von Hammer.js bzgl. sehr vieler Browser und die darin möglichen Touch-Interaktionen aufgelistet (Quelle: wiki Hammer.js).

Diese Liste zeigt deutlich, dass eine sehr große Anzahl verfügbarer Browser getestet worden ist und so gut wie alle Versionen unterstützt werden.

Compatibility

	Tap	Double Tap	Hold	Swipe	Drag	Multitouch
BlackBerry						
Playbook	X	X	X	X	X	X
BlackBerry 10	X	X	X	X	X	X
iOS						
iPhone/iPod iOS 6	X	X	X	X	X	X
iPad/iPhone iOS 5	X	X	X	X	X	X
iPhone iOS 4	X	X	X	X	X	X
Android 4						
Default browser	X	X	X	X	X	X
Chrome	X	X	X	X	X	X
Opera	X	X	X	X	X	X
Firefox	X	X	X	X	X	X
Android 3						
Default browser	X	X	X	X	X	X
Android 2						
Default browser	X	X	X	X	X	
Firefox	X	X	X	X	X	
Opera Mobile	X	X	X	X	X	
Opera Mini	X					
Windows 8 touch						
Internet Explorer 10	X	X	X	X	X	X
Chrome	X	X	X	X	X	X
Firefox	X	X	X	X	X	X
Windows Phone 8						
Internet Explorer 10	X	X	X	X	X	X
Windows Phone 7.5						
Internet Explorer	X					
Chrome OS						
Chrome	X	X	X	X	X	X
Firefox OS (simulator)						
Firefox	X	X	X	X	X	X
Other devices						
Kindle Fire	X	X	X	X	X	X
Nokia N900 - Firefox 1.1	X					
Windows Dekstop						
Internet Explorer 7**	X	X	X	X	X	X*
Internet Explorer 8**	X	X	X	X	X	X*
Internet Explorer 9	X	X	X	X	X	X*
Internet Explorer 10	X	X	X	X	X	X*
OSX						
Firefox	X	X	X	X	X	X*
Opera	X	X	X	X	X	X*
Chrome	X	X	X	X	X	X*
Safari	X	X	X	X	X	X*
	Tap	**Double Tap**	**Hold**	**Swipe**	**Drag**	**Multitouch**

Abbildung 2: von Hammer.js unterstützte Browser

Synchronisation

Neben den Multitouch-Möglichkeiten lag mein weiterer Schwerpunkt auf der Analyse von Sychronisationsmöglichkeiten über mehrere Geräte hinweg. Aufgrund der Idee, ein auf vielen Geräten lauffähiges Programm mittels HTML und Javascript zu entwickeln, musste daher ebenfalls ein Konzept gefunden werden, dass durch diese im Browser laufenden Technologien realisierbar wäre.

Das Hauptproblem ist dabei zunächst das zugrundeliegende Protokoll HTTP, welches für das Laden von Ressourcen im Browser genutzt wird. Es ist nicht für eine kontinuierliche Übertragung (wie bei einer echten Synchronisation) ausgelegt, sondern nur dafür gedacht, auf Anfragen eines Clients (also der Browser, der eine bestimmte Ressource laden möchte) zu reagieren und diese dann zu senden, sofern sie existiert. Somit werden auch keine Daten übertragen, solange sie nicht explizit anfordert werden.
Das bedeutet wiederum, um synchronisieren zu können (also kontinuierlich Daten empfangen), müssten von jedem Client pausenlos neue Anfrage gestellt werden, die von einem Server dann entsprechend schnell verarbeitet werden müssten. So eine Umsetzung ist durchaus möglich und es gibt auch Web-Applikationen, die diese Art der Daten-Synchronisation verwenden. Einfache Tests meinerseits mittels Javascript führten jedoch nicht zu befriedigenden Ergebnissen.

Stattdessen habe ich mich für die Benutzung von WebSockets entschieden, eine relativ neue Technik, die von allen modernen Browsern unterstützt wird. Der entscheidende Vorteil bei dieser Methode ist die Tatsache, dass ein Client einmal eine Verbindung zu einem Server herstellen kann, welche im Gegensatz zu HTTP geöffnet bleibt und dem Server nun die Möglichkeit bietet, von sich aus Daten zu übertragen. Die Verbindung ist zudem bidirektional, also auch der Client kann die Verbindung verwenden, um wiederum jederzeit Daten an den Server zu übertragen. Da die Verbindung zudem nur einmalig je Client aufgebaut werden muss, ist der Datenaustausch deutlich ressourcenschonender als es über HTTP möglich wäre. Somit bietet dieses Protokoll die optimalen Voraussetzungen für eine Synchronisation über ein Netzwerk.

Der einzige Nachteil dieser Technik ist, dass ein Server vorhanden sein muss, der die Verbindungen der Clients entgegennimmt – eine direkte Verbindung zwischen z.B. zwei Browsern lässt das WebSocket-Protokoll nicht zu.

Der Vorteil eines Servers wiederum ist aber, dass dieser Daten über eine Session (also eine Sitzung, in der sich verschiedene Clients an- und abmelden) hinaus zentral speichern könnte. Ein Benutzer könnte also beispielsweise eine Session ablegen, sodass an dieser zu einem späteren Zeitpunkt weitergearbeitet werden könnte. Ebenso lassen sich auf dem Server Techniken wie PHP einsetzen, die auf dem Client normalerweise nicht zur Verfügung stehen, mit deren Hilfe z.B. Bilder oder Videos verarbeitet und abgespeichert werden können. Ebenso wäre die Anbindung an eine Datenbank denkbar, in der beispielsweise Benutzerkonten oder ähnliches hinterlegt werden könnten, die für eine Autorisierung herangezogen werden könnten, sofern dies für den Anwendungsfall notwendig sein sollte.

Für die Kommunikation verwende ich auf Client-Seite das Javascript-Framework 'Autobahn.js', das eine standardisierte WebSockets-Verbindung mit einem sogenannten WAMP-Server herstellen kann. WAMP steht für WebSockets Application Messaging Protocol, welches innerhalb der WebSockets-Spezifikation eine weitere Unterspezifikation definiert - vorstellbar als eine Vorschrift, in welchem Format die Anfragen an den Server gesendet werden und wie dieser darauf antworten darf. Das Protokoll gilt gleichermaßen für Client und Server und gibt somit eine einheitliche Kommunikationsart bei der Entwicklung von WebSockets-gestützten Programmen vor.

Mittels WebSockets ist theoretisch jede Art von Datenaustausch möglich, das Hauptaugenmerk ist allerdings die Geschwindigkeit, weswegen es auf den Austausch von kleinen Datenpaketen optimiert ist. In diesem Fall werden neben den Daten für einen Verbindungsaufbau sämtliche relevante Informationen zu gerade geöffneten Medien transferiert.

Kommunikations-Struktur

Meine bisherige Programmierung erlaubt eine An- oder Abmeldung an einem über das Netzwerk erreichbaren Server. Dabei ist die Anzahl der Clients unbegrenzt. Aktuell kann ein Server jedoch nur eine gleichzeitige Session verwalten, das bedeutet alle Clients, die sich mit diesem Server verbinden, landen in einer Gruppe und können miteinander kommunizieren. Hier wäre z.B. noch eine Erweiterung denkbar, die mehrere verschiedene Sessions auf einem Server zulässt, sodass mehrere Gruppen unabhängig voneinander über den gleichen Server kommunizieren können.
Kombiniert mit einer einfachen Benutzerverwaltung auf dem Server wäre außerdem ein Passwort- oder Benutzer-beschränkter Zugriff denkbar.

Wie bereits in den Zielen erwähnt, habe ich das Programm so gestaltet, dass auch eine Offline-Benutzung möglich ist. Das Bedeutet, dass auch ohne Anmeldung an einem Server verschiedene Medien geöffnet und auf dem Bildschirm angezeigt und bearbeitet werden können. In diesem Fall findet logischerweise keine Synchronisation mit anderen Geräten statt. Es ist jedoch möglich, jederzeit eine Verbindung zu einem Server aufzubauen oder wieder zu trennen. Solange mindestens ein Client noch mit dem Server verbunden ist, bleibt die aktuelle Session erhalten, inklusive aller geöffneten Dokumente und ihre Position auf der Arbeitsfläche. Trennt sich der letzte Client, wird die Session auf dem Server zurückgesetzt. Als zukünftige Erweiterung der Software wäre hier z.B. noch das dauerhafte Speichern einer Session auf dem Server denkbar, sodass diese zu einem späteren Zeitpunkt geladen und die Arbeit fortgesetzt werden kann.

Die Verbindung zu einem Server wird über dessen IP-Adresse im lokalen Netzwerk und einen festgelegten Port durchgeführt. Versucht man sich zu einem nicht erreichbaren oder vorhandenen Server zu verbinden, oder wird die Verbindung getrennt, erhält jeder Client eine entsprechende Fehlermeldung. Das Trennen einer Verbindung kann mit einem Buttonklick im Programm jederzeit vollzogen werden oder durch einfaches Schließen des Browserfensters oder -Tabs, da die WebSocket-Verbindung automatisch beim Schließen des entsprechenden Tabs im Browsers beendet wird.

Sobald ein Client ein Dokument von seiner Festplatte in das Programm lädt, wird dieses ebenfalls an den Server übertragen und dort abgespeichert (sofern eine Verbindung zu diesem besteht). Somit wird das Medium für alle weiteren verbundenen Clients ebenfalls sichtbar und nutzbar. Externe Medien wie YouTube-Videos werden dagegen nicht auf Server gespeichert, sondern einfach von ihrer öffentlich zugänglichen Quelle im Internet geladen. In diesem Fall wird nur die Information, welches Video geladen werden soll, an den Server und von dort an weitere Clients weitergegeben.
Sobald ein Dokument auf der Arbeitsfläche verschoben, manipuliert oder wieder von ihr entfernt wird, werden diese Daten mittels der bereits vorgestellten WebSockets-Verbindung an den Server übertragen und von dort an mögliche weitere Clients. Zu den übertragenen Daten gehören u.a. Position, Größe, Rotation, Name sowie eine eindeutige ID des Objektes.

Zu den unterstützten Dateiformaten gehören bisher die üblichsten Bildformate JPEG, PNG, GIF und BMP, die Video-Formate MP4, OGV und WEBM, einfache Text-Dokumente, PDF-Dokumente sowie YouTube-Videos, die per Link integriert werden können. Die Darstellbarkeit von PDF-Dateien und Videos hängt jedoch vom verwendeten Endgerät und der darauf verfügbaren Software ab.

Hinzugefügte Medien werden innerhalb der Software automatisch in einer Liste geöffneter Dateien aufgelistet. Dort können sie von jedem Benutzer per Klick hervorgehoben und auch wieder aus dem Programm entfernt werden. Hier wäre eventuell als Funktionserweiterung auch das Sperren von einzelnen Dokumenten denkbar oder die Sichtbarkeit einzelner Dokumente, sofern dies in einem Anwendungsfall von Interesse ist.

Zur besseren Übersicht hat jeder Client zudem die Möglichkeit, sich eine Farbe und einen Benutzernamen zuzuweisen, um bei mehreren verbundenen Clients diese besser zuordnen zu können. Aus dem Namen des Benutzers wird zusätzlich ein individuelles Benutzerbild generiert.

Funktionsumfang

Die bereits im vorgestellten gewünschten Ziele konnten zum größten Teil umgesetzt werden. Die Synchronisation über die WebSockets-Technologie funktioniert sehr zuverlässig und lässt auch die Übertragung von größeren Bildern und Videos zu. Verschiedene Datei-Formate können angezeigt werden, jedoch ist das Bearbeiten und Speichern von Text-Dateien über den Browser aus Sicherheitsgründen nicht möglich.

Die Gesten zum Verschieben, rotieren und Zoomen von Elementen auf der Arbeitsoberfläche funktionieren wie von anderen Multitouch-Geräten bereits bekannt. Eine alternative Benutzung per Maus und Tastatur ist ebenfalls möglich.
Das gleichzeitige Arbeiten mehrerer Personen an einem Gerät führt zum Teil noch zu ungewünschten Effekten, die durch falsch interpretierte Gesten des verwendeten Frameworks zurückzuführen sind. Keine der von mir untersuchten Javascript-Bibliotheken hat von Haus aus eine Unterstützung von mehreren gleichzeitigen Gesten vorgesehen, sondern kann lediglich mehrere Touches, die zu einer Geste wie z.B. Zoom verknüpft werden, verarbeiten. Dies lässt sich vermutlich darauf zurückzuführen, dass die wenigsten aktuellen handelsüblichen Geräte wie Tablets und Smartphones die Bedienung von mehreren Personen und mehreren Berührungen vorgesehen haben.
Aufgrund dieser Tatsache habe ich versucht, die Erkennung von mehreren parallelen Gesten robuster zu implementieren, was sich jedoch als relativ schwierig erwiesen hat. Eine Trennung von verschiedenen Benutzern ist lediglich über mehrere zu einem Server verbundene Clients zuverlässig möglich, während die Erkennung mehrerer Benutzer an einem Gerät lediglich durch Abstände der Berührungen erahnt werden kann.

Da dieses Thema sehr komplex ist und viel Entwicklungs- und Arbeitsaufwand bedeutet, konnte ich es innerhalb des Projektzeitraumes nicht zuverlässig umsetzen, sondern nur einen ersten Ansatz für eine zuverlässigere Erkennung erarbeiten.
Meiner Einschätzung nach wird jedoch mit der zunehmenden Verbreitung von Multitouch-fähigen Endgeräten für mehrere Personen auch die Unterstützung für solche Geräte mit

ihren zusätzlichen Einsatzmöglichkeiten steigen und entsprechende Bibliotheken umgesetzt werden.

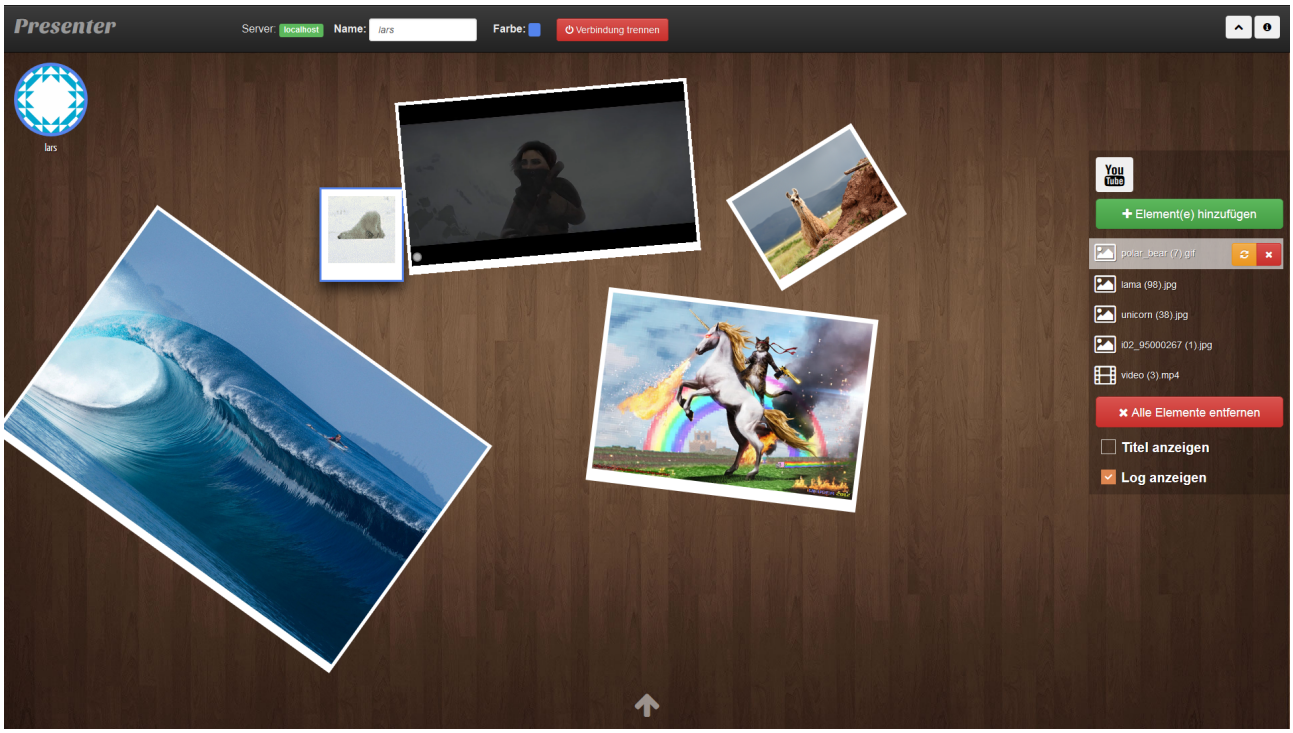

Abbildung 3: Ergebnis des Projektes: Entwickelte Software 'Presenter' mit Beispiel-Dateien

Quellen

Touch / Multi-Touch

- https://dvcs.w3.org/hg/webevents/raw-file/tip/touchevents.html (Browser Touch Spezifikation)
- http://www.w3.org/TR/pointerevents/ (Browser Pointer Events Spezifikation)
- http://caniuse.com/touch (Übersicht aller Browser und deren Touch-Funktionalitäten)
- http://www.quirksmode.org/mobile/tableTouch.html (Mobile Browser Touch-Support)
- http://www.html5rocks.com/de/mobile/touch/ (Überblick Touch-Funktionalität)
- http://www.sitepen.com/blog/2011/12/07/touching-and-gesturing-on-iphone... (iOS Touch)
- http://blogs.msdn.com/b/davrous/archive/2013/02/20/handling-touch-in-your-html5-apps-thanks-to-the-pointer-events-of-ie10-and-windows-8.aspx (Pointer-Konzept von Microsoft)
- http://de.slideshare.net/ysaw/creating-responsive-html5-touch-interfaces (Anregungen zum Thema Multitouch)

WebSockets:

- http://www.html5rocks.com/en/tutorials/websockets/basics/ (Allgemein)
- http://mustafaakin.wordpress.com/2011/10/16/introducing-websocket-file-transfer/
- http://wamp.ws/ (WAMP Protokoll)
- http://socketo.me/ (Ratchet – PHP Framework)
- http://autobahn.ws/ (Autobahn.js – Javascript Framework)