UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Arbeitsgruppe Aktives Sehen

Demonstrator für Interaktion durch Augenbewegung II

Studienarbeit
im Studiengang Computervisualistik

vorgelegt von

Christoph Schaefer

Betreuer:	Dipl.-Inf. Detlev Droege, Institut für Computervisualistik, Fachbereich Informatik
Erstgutachter:	Dipl.-Inf. Detlev Droege, Institut für Computervisualistik, Fachbereich Informatik
Zweitgutachter:	

Koblenz, im Juni 2007

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Richtlinien der Arbeitsgruppe für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	ja ☐	nein ☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	ja ☐	nein ☐

Koblenz, den .... Unterschrift

Inhaltsverzeichnis

Kapitel 1

Einleitung

Nachdem an der Universität Koblenz schon zwei Diplomarbeiten über das Thema Interaktion per Augenbewegung verfasst wurden, fehlte nur noch eine geeignete Anwendung, um die Möglichkeiten dieser Art der Kommunikation zwischen Mensch und Computer anschaulich zu demonstrieren.

Das Endergebnis soll möglichst leicht zu installieren und plattformübergreifend einsetzbar sein. Aus diesem Grund wurde für die praktische Umsetzung die Klassenbibliothek Qt benutzt. Die erhaltenen Blickrichtungskoordinaten werden grafisch aufbereitet und plakativ dargestellt.

Nach einer kurzen Kalibrierung kontrollieren die Augen den Mauszeiger in Echtzeit. Verweilt der Blick längere Zeit auf einem Punkt, wird ein Mausklick ausgelöst, wodurch auch Tasten und Links betätigt werden können. Da die Entwicklung eines Demonstrators ausreichend Stoff für zwei Arbeiten liefert, ist die Umsetzung und Auswertung der Kalibrierung nicht Teil dieser, sondern Teil einer parallel laufenden Studienarbeit von Sascha Lange.

Die Interaktion mit einem Computer, allein mit Hilfe der Augenbewegung, eröffnet ein breites Spektrum an Anwendungen, wie zum Beispiel die Texteingabe oder das Erstellen von Bildschirmkarten mit eingezeichneter Blick-Verweildauer. Besonders publikumswirksam ist sicherlich das vorgestellte Spiel, mit dem auf ansprechende Art und Weise die Steuerung per Eye Tracker demonstriert werden kann.

Der Aufbau der Arbeit ist wie folgt: Im Folgenden wird auf den Stand der Technik an der Universität Koblenz sowie in der freien Wirtschaft eingegangen. Im darauf anschließenden Kapitel geht es um die eigene Umsetzung der Demonstratorsoftware, insbesondere um den Cursor, die Heatmap und ButtonUp. Als nächstes werden die durchgeführten Experimente beschrieben und ausgewertet. Die Zusammenfassung in Kapitel 5 bildet den Schluss dieses Textes.

Kapitel 2

Stand der Wissenschaft

2.1 Ausgangspunkt

Als Ausgangspunkt dieser Arbeit diente Fabian Fritzers Diplomarbeit „Texteingabe per Augenbewegung“ [Fri05]. Sein Gaze Tracker wurde in C++ unter Verwendung des PUMA-Frameworks[1] implementiert. Als Betriebssystem ist ausschließlich Linux vorgesehen.

Um dieses System inklusive Kamera und Infrarot-LEDs zum Laufen zu bringen sind mehrere Schritte notwendig. Zum einen muss ein Framegrabber mit analogem Videoeingang vorhanden sein und zum anderen muss dieser auch von PUMA unterstützt werden. Da das Programm den Bilderstrom mit Hilfe von PUMA bezieht, kann diese Tatsache auch nur schwer umgangen werden. So stellt sich die Installation auf einem fremden Computer als äußerst zeitaufwändig und fehlerbehaftet heraus. Gerade das Kompilieren von PUMA und den notwendigen Erweiterungen wie KONIHCL erschweren den Einstieg.

Die Einschränkung auf ausgewählte Hardware wie den Framegrabber, die analoge Videokamera, die LEDs und nicht zuletzt das Batteriepack zur Stromversorgung der Beleuchtung, macht das ganze System sehr unflexibel. Zur Verwendung des Gaze Trackers unter Windows ist gar ein zweiter Linux-PC nötig, der die Rechenaufgaben übernimmt und an-

[1] „Programmierumgebung für die Musteranalyse“ der Arbeitsgruppe aktives Sehen

schließend nur noch die Mauskoordinaten an Windows übermittelt.

Ein solch aufwändiger Aufbau erschwert das Demonstrieren der Ergebnissen gerade außerhalb der Universität. Der neue Demonstrator soll deshalb handlicher und damit flexibler sein. Dazu zählt auch, dass die Einstiegsschwelle, inklusive der Installation und Inbetriebnahme, gesenkt wird.

Ein erster Schritt der Vereinfachung ist die Verwendung einer USB-Kamera, die ohne Umweg über einen Framegrabber angesprochen werden kann. Da USB eine Spannung von 5 Volt und bis zu 2,5 Watt liefert, kann auch die Beleuchtung mittels eines zweiten USB-Anschluss mit Energie versorgt werden. Mit dem Auswechseln der Kamera müsste aber auch die PUMA-Anbindung angepasst werden. Hierzu wurden verschiedene Versuche mit CLIP [CLI] und unicap [uni] unternommen. CLIP von John Robinson zusammengestellt, ist eine C++ Template Sammlung zur Bildverarbeitung unter Linux, Mac OS und Windows. Unicap ermöglicht die Videoaufnahme von einer Vielzahl verschiedener Kameras. Mit Hilfe dieser Bibliotheken sollte die Anbindung einer neuen USB-Kamera an PUMA gelingen. Nach einer langwierigen aber letzten Endes ergebnislosen Versuchsphase, beschlossen wir, uns weg von der Hardwareverbesserung hin zur reinen Softwareentwicklung um zu orientieren.

Eine zweite, zur Zeit noch laufende Diplomarbeit, von Thorsten Geier [Gei07] beschäftigt sich ebenfalls mit dem Thema Interaktion durch Augenbewegung. Er verwendet zwar die gleiche Technik, erstellt die Software jedoch, ohne Einbindung von PUMA, unter Windows. Geiers Ansatz setzt auf Fritzers Arbeit auf und erscheint auf Hard- sowie Softwareseite etwas ausgereifter und einfacher handhabbar zu sein als das Vorgängermodell.

Deshalb konzentriert sich der vorgestellte Demonstrator auf die Zusammenarbeit mit Geiers Gaze Tracker, welcher komplett unverändert blieb. Veränderungen der bestehenden Hardware sind nicht Teil dieses Ausarbeitung. Daher ist mit dem Begriff „Demonstrator“ in diesem Zusammenhang ausschließlich die von uns entwickelte Software gemeint. Die von Thorsten Geier verwendete Hard- und Software wird im Folgenden mit dem Wort Gaze Tracker zusammengefasst.

2.2 Kommerzielle Anwendungen

Auf dem Markt befinden sich eine Vielzahl unterschiedlicher Produkte. Anwendungen, die einen Gaze Tracker zur Eingabe nutzen, können in verschiedenen Einsatzgebieten verwendet werden..

Angefangen beispielsweise bei der Firma Tobii, die unter anderem eine Software zur Texteingabe via Augenbewegung speziell für Körperbehinderte im Angebot hat [AG].

Oder ein System von Optom, dass die Signalverarbeitung im Gehirn von Lernbehinderten, mit Hilfe eines Eye-Tracker-gestützten Programms trainieren will. Kindern mit Legasthenie oder ADS soll damit im frühen Alter geholfen werden [OL].

Die Gesellschaft AMTech hat sich auf die Untersuchung der Tagesschläfrigkeit bei Menschen spezialisiert. Ihre Anwendungen kommen insbesondere in der Schlafmedizin, Arbeitsmedizin, Pharmakologie, Rechtsmedizin, Neurologie und Psychologie zum Einsatz [AG].

Eine breite Produktpalette auf dem Gebiet der Blickverfolgung, führt auch das Unternehmen LC Technologies. Sie beinhaltet neben Analysetools für Bilder, Filme und Webseiten auch Software, die die Interaktion mit einem Computer ermöglicht. Dazu zählt unter anderem eine Steuerung für Militärfahrzeuge, die mit einer Kombination aus Sprach- und Blickbefehlen bedient wird [LTI]. Bleibt nur zu hoffen, dass der Satz „Wenn Blicke töten könnten“, auch weiterhin nur im Konjunktiv Verwendung findet.

Unter Berücksichtigung der Bilder von sechs verschiedenen Kameras, erweitert die Firma Smart Eye ihren Augenverfolger um eine 360 Grad Abtastung [SEA].

Bei Seeing Machines gibt es ein System zu kaufen, dass zusätzlich zur Augenstellung gleich die gesamten biometrischen Daten des Gesichts erfasst [SM].

Die Thomas RECORDING GmbH schreckt selbst vor Tierversuchen mit Affen nicht zurück und untersucht mit Hilfe ihres Gaze Trackers unter anderem auch die Augenbewegung der Primaten [TRG].

Kapitel 3

Implementierung des Demonstrators

3.1 Wahl der Hilfsmittel

3.1.1 Qt 4

Zu Beginn der Programmierarbeit stand noch nicht fest, mit welchem Gaze Tracker der Demonstrator letztendlich kombiniert werden soll. Da beide Modelle mit Linux und Windows unter verschiedenen Betriebssystemen laufen, war eine plattformübergreifende Lösung unbedingt erforderlich. Aus diesem Grund wurde bei der Programmierung auf die Klassenbibliothek Qt der Firma Trolltech zurück gegriffen.

Hierbei handelt es sich, ähnlich wie bei Java Swing, um ein C++ Framework, das speziell zur grafischen Darstellung von Fenstern und anderen Elementen entwickelt wurde. Für die Verwendung in freier Software steht Qt unter der GNU General Public Licens und ist damit kostenlos erhältlich.

In der Version 4.2 werden Windows 98 bis XP, Mac OS X, Linux, Solaris, HP-UX und viele andere Unix Systeme, die X11 zum Anzeigen ihrer grafischen Benutzeroberfläche nutzen, unterstützt [Tro06]. Zusätzlich wurde Qt mit der Version 4.3 auch für Windows Vista zertifiziert.

Die Bibliothek ist stark objektorientiert aufgebaut und bietet eine Vielzahl an Fenster-elementen zur Auswahl. Benutzeroberfläche können plattformübergreifend einheitlich mit den üblichen Widgets wie Menüs, Toolbars und Buttons ausgestattet werden. Dabei sehen Qt Fenster und deren Inhalte normalerweise genau so aus, wie alle anderen Fenster im verwendeten Betriebssystem.

Eine Besonderheit stellt die Kommunikation zwischen Objekten dar. Sie geschieht nach dem „Signal-Slot“ Prinzip. So kann zum Beispiel ein Button nachdem ein Klick-Ereignis auf ihm ausgelöst wurde, ein Clicked() Signal versenden. Dieses Signal kann jeden beliebigen Slot aufrufen, mit dem es verbunden ist. Slots sind also eine besondere Art von Funktionen, die neben dem normalen Aufruf auch durch Signale in Gang gesetzt werden können.

Im Demonstrator findet sich eine solche Signal-Slot-Verbindung unter anderem zwischen dem neu geschriebenen Cursor und allen Klassen die eine Instanz der Klasse Cursor erzeugen. Wird eine bestimmte Zeit überschritten, sendet er ein Signal cursorClicked(), welches mit einem Slot im jeweiligen Eltern-Widget verbunden ist.

Ein weiterer Grund für Qt ist die gute 2- und 3D Grafikunterstützung. Sie macht es sehr komfortabel 2D Inhalte auf den Bildschirm zu bringen. Viele gängige Bildformate werden bereits von Haus aus unterstützt und können so recht schnell geöffnet oder gespeichert werden. Von zusätzlichen Möglichkeiten wie Antialiasing, die Einbettung von OpenGL oder die Nutzung eines kompletten Drag and Drop Mechanismuses wird in unserem Entwurf kein Gebrauch gemacht.

3.1.2 QDevelope

Zur Erstellung des Programms kam die speziell auf Qt 4 zugeschnittene Open Source Entwicklungsumgebung QDevelop zum Einsatz. Diese von Jean-Luc Biord geschriebene IDE ist mit der Version 0.23 noch in einem sehr frühen Stadium und bei weitem noch nicht so ausgereift wie beispielsweise KDevelop. Trotz des gleich klingenden Namen legt der Urheber Wert darauf, dass es sich bei QDevelop nicht um den kleinen Bruder von KDevelop handelt, sondern um eine schlanke Umgebung mit eigenem Ansatz.

Ihr großer Vorteil ist nämlich, dass sie ebenso wie Qt plattformübergreifend verfügbar ist.

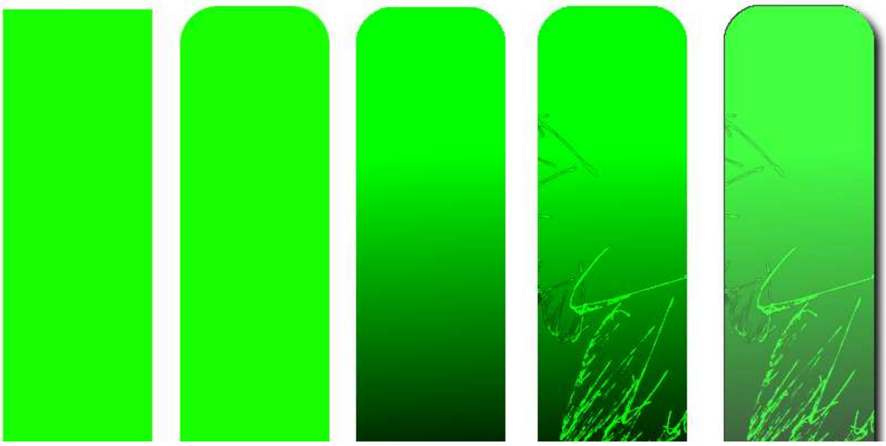

Bild 3.1: Die Entstehung einer Schaltfläche

Da Linux, Mac OS X, FreeBSD und Windows unterstützt werden, konnten während der Programmierung neben dem Quellcode auch die Projektdateien zwischen den verschiedenen Betriebssystemen ausgetauscht werden.

3.1.3 GIMP

Die zahlreichen grafischen Elemente wurden mit dem **G**NU **I**mage **M**anipulation **P**rogram, kurz GIMP, designt. Vor allem im Spielmodus des Demonstrators wurde die Oberfläche optisch aufgewertet. Dort kommen keine Qt Standard Widgets wie Menüs oder Toolbars zum Einsatz. Das Fenster wird im Vollbildmodus aufgerufen und zeigt nur selbst erstellte Element, die groß genug zur groben Augensteuerung ausgelegt sind. Um solch große Knöpfe und Flächen nicht einfarbig und damit auch eintönig erscheinen zu lassen, wurden fast alle Formen mit Verläufen gefüllt. Zusätzlich kam im Fall des Hintergrunds und der vier Hauptbuttons der Filter „Render“ → „Natur“ → „Flammen...“ zum Einsatz. Er erzeugt ein zufälliges Muster von Punkten und Linien, die über verschiedene Parameter verstreut werden können. Mit Hilfe des Skriptes „Schatten“ → „Xach-Effekt“ wurden Schatten generiert, die die Plastizität der Elemente vor dem Hintergrund erhöhen sollen. Die Schritte von einem Rechteck bis zu einer Schaltfläche mit abgerundeten Ecken und

Schatten, werden exemplarisch an Hand von Abbildung 3.1 gezeigt.

Alle Bilder einschließlich der Icons wurden im PNG-Format gespeichert und können so direkt von Qt verarbeitet werden.

3.2 Mögliche Anwendungen

Bei der Interaktion via Augenbewegung entsteht die ungewöhnliche Situation, dass das Auge eine zusätzliche Aufgabe übernimmt. Neben der Bildaufnahme steuert es plötzlich einen virtuellen Zeigefinger, der immer entlang der Sichtachse zeigt. Wird dieser Blickpunkt sichtbar gemacht, zeigt sich deutlich wie unstetig das Auge wandert, um Objekte zu erfassen. Solche Abtastbewegungen geschehen meist unterbewusst und sind auch nach einem Training nur schwer zu kontrollieren.

Die beiden Funktionen der Bildaufnahme und der gleichzeitigen Steuerung, stellen nicht nur den Probanden, sondern auch die Entwickler möglicher Anwendungen, vor eine große Herausforderung. Deshalb wurde im Entwurf des Demonstrators dieser Dualität besondere Beachtung geschenkt.

Eine weitere Eigenschaft, die den Gaze Tracker von einer gewöhnlichen Maus unterscheidet, ist das Fehlen sämtlicher Tasten. Damit werden für einen „Klick“ oder „Doppel-Klick“ ganz neue Konzepte benötigt. Kompliziertere Mausgesten, wie das Gedrückthalten und kontrollierte Loslassen (Drag and Drop) der Tasten ist eigentlich unmöglich. Auch die Unterscheidung zwischen linker und rechter Maustaste, ist ohne zusätzlichen Aufwand, ausgeschlossen.

Es wird klar, dass sich potenzielle Anwendungen mit einem sehr eingeschränkten Cursor begnügen müssen. Die gesamte Benutzeroberfläche zur Eingabe von Informationen, wird auf große Knöpfe, die mit einem einfachen Klick betätigt werden können, reduziert. Diese Tatsache verkleinert das Feld der möglichen Anwendungen sehr stark. Dennoch soll der Demonstrator den Betrachter spielerisch an das Thema ‘’Interaktion per Augenbewegung” heranführen. Vier statische Knöpfe, die mit unterschiedlichen Ereignissen verknüpft sind, reichen dazu sicherlich nicht. Dynamische Semantik der Knöpfe würden den Spaßfaktor des Spiels länger erhalten.

Vorzustellen wäre in dieser Richtung zum Beispiel ein Quiz, dessen vier Antwortmöglichkeiten auf vier Flächen erscheinen, von welchen die Richtige ausgewählt werden muss. Bei diesem Ansatz hinge der Erfolg jedoch nicht nur von der Bedienung des Gaze Trackers, sondern auch von dem Allgemeinwissen der Testperson ab. Das geplante Spiel soll aber ungefiltert die reine Augenbewegung in den Mittelpunkt stellen. Darum wurde diese Idee nicht weiter verfolgt.

Allein durch die Macht des Augenblicks, etwas auf dem Bildschirm in Bewegung zu setzen, wäre sicher reizvoller. Auf Grund der oben beschriebenen Dualität, würde das Lenken einer Spielfigur durch ein Labyrinth jedoch nicht funktionieren. Denn um den Weg zu erkunden, springt das Auge ohne Rücksicht auf Wände über die gesamte Fläche hinweg. Die Figur würde ständig unbrauchbare Steuerbefehle erhalten und den virtuellen Spieler gegen die Wand laufen lassen. Das Ignorieren von kurzen Blicksequenzen die zur Orientierung dienen, würde zwar die Stabilität erhöhen, allerdings den Spielfluss sehr hemmen.

Mit dem gesuchten Spiel soll ohne Taktik oder Vorwissen in Echtzeit interagiert werden können. Die Eingabemöglichkeiten sollen auf ein Minimum begrenzt sein und dennoch durch ihre Dynamik keine Langeweile aufkommen lassen. Diese Voraussetzungen führten letztendlich zu dem in Abschnitt 3.3.3 beschriebenen Konzept.

Was für die Steuerung seines digitalen Egos auf dem Spielfeld nur hinderlich ist, kann in anderen Anwendungsgebieten gerade als Chance ergriffen werden. Die unterbewusste Augenbewegung zur Abtastung des Blickfeldes, birgt zum Beispiel nützliche Informationen über den Nutzer und das Betrachtete. Liegt der Blickpunkt lange auf einer Stelle die keine besonderen Merkmale besitzt und sind die Blickrichtungsänderungen eher träge, könnte dies auf fehlende Konzentration oder Müdigkeit der Testperson hindeuten. Springt der Blick im Gegenteil von einem Punkt zum nächsten und werden unterschiedliche Regionen fixiert, ist dies vielleicht auf eine Suche nach einer bestimmten Information zurück zu führen.

Um solche Bewegungen über die Zeit festzuhalten, müssten sie entweder in einem Video oder in einem Bild mit ausgeklügelter zeitlicher Visualisierung gespeichert werden.

Denkbar wären durchnummerierte Knoten, die durch Kanten entlang der Bewegungsrichtung verbunden sind. Der Knoten mit der Startnummer 1 wäre dabei die erste Fokussierung

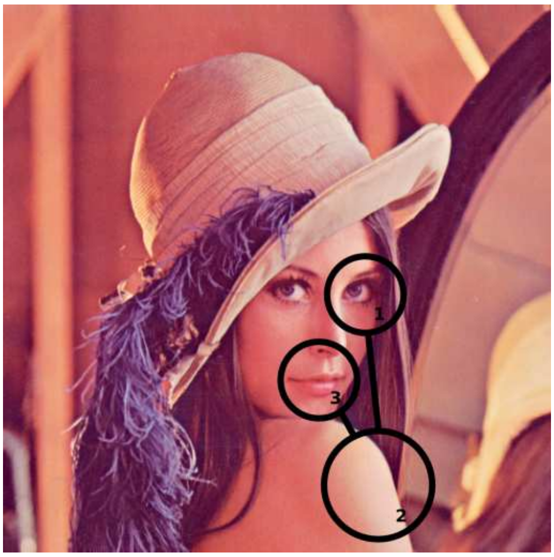

Bild 3.2: Drei Fokussierungen unter Berücksichtigung der zeitlichen Abfolge

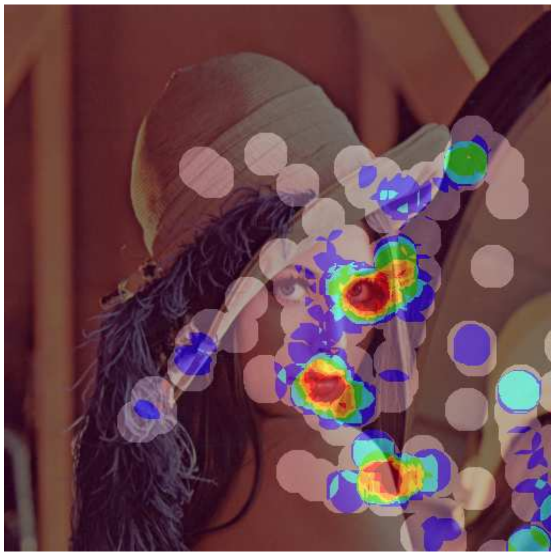

Bild 3.3: Beispiel einer Heatmap ohne Beachtung der Reihenfolge

auf dem Bildschirm. Verbunden mit einer Kante würde der Knoten 2 die zweite Fokussierung anzeigen, dieser wiederum wäre mit dem dritten Knoten verbunden und so weiter. Die Verweildauer des Blickes an einer Stelle, könnte durch den Radius des zugehörigen Knotens zum Ausdruck gebracht werden. Die Ergebnisse einer solchen Analyse würden dann Abbildung 3.2 gleichen.

Doch auch ohne die zeitliche Reihenfolge der auftretenden Fixationen, ist diese Art der Daten von Nutzen. Wird die Komponente des „Wann“ vernachlässigt, geht zwar die Abfolge der Blickbewegung verloren, dafür vereinfacht sich das Modell erheblich.

Das reine Aufaddieren der Aufenthaltszeit des Blickes auf jedem Pixel, erzeugt bereits eine sehr nützliche Information. Interessante Stellen im betrachteten Bild werden sicherlich länger fixiert als uninteressante Regionen. Zusätzlich kann ermittelt werden, wie lange ein Betrachter benötigt, um das Dargestellte aufzunehmen. Später werden die unterschiedlich hohen Zeiten der einzelnen Pixel, ähnlich wie auf einem Infrarot-Wärmebild, ihrem Betrag entsprechend eingefärbt und zu einer Heatmap zusammengefügt. Durch Überblenden der Ergebnisse mit dem Testbild, entsteht so eine Karte wie in Abbildung 3.3 dargestellt. Auf die praktische Umsetzung und weitere Details zu diesem Thema wird in Abschnitt 3.3.2 eingegangen.

3.3 Die umgesetzten Komponenten

Ein funktionierender Gaze Tracker eröffnet zahlreiche Einsatzmöglichkeiten. Neben den bereits kurz angerissenen Anwendungen werden in diesem Abschnitt nun drei ausgewählte Konzepte im Detail besprochen. Da die Entwicklung des Demonstrators zusammen mit Sascha Lange erfolgte, besteht das fertige Programm noch aus zusätzlichen Komponenten, die nur Teil seiner Studienarbeit sind.

3.3.1 Der Cursor

Das Fundament für jede Applikation, die einen Blickverfolger als Eingabegerät verwendet, ist der Mauszeiger selbst. Er übernimmt die Anzeige des momentanen Blickpunkts und stellt fest, wann und wohin der Benutzer klickt.

Der vorgestellte Cursor setzt voraus, dass die erhaltenen Bildschirmkoordinaten bereits kalibriert und auf den sichtbaren Bereich zugeschnitten sind.

Seine Hauptaufgabe ist es festzustellen, wann und wo ein Klick-Event ausgelöst werden muss. Am einfachsten wäre es, die Auslösung direkt an das „Mouse-over-Event“ zu hängen und ein Klick zu versenden, sobald sich ein Button oder sonstige Ziele unter dem Cursor befinden. Dieser Ansatz führt allerdings zu einer hohen Anzahl von ungewollten Klicks, da der Blick zur Orientierung ständig auf dem ganzen Bildschirm wandert. Will der Betrachter nur die Beschriftung der Knöpfe lesen, würde er sie mit dieser Methode gleichzeitig drücken. Solche unerwünschten Tastenklicks beim Gaze Tracking werden auch „Midas-Touch“ genannt. Um das Verhältnis von gewollten zu ungewollten Auslösungen zu optimieren stellt Fabian Fritzer in seiner Diplomarbeit verschiedene Lösungen vor:

Dwell Time Der Knopf wird erst nach einer bestimmten Verweildauer auf dessen Fläche ausgelöst.

Off Screen Targets Die virtuellen Tasten liegen außerhalb des Bildschirms, sie verbrauchen keinen Platz und können so beim normalen Betrachten der Inhalte nicht getroffen werden.

Blinzeln Tasten werden ausgelöst, indem der Betrachter seinen Blick auf sie fixiert und dann die Augen kurz schließt.

Blickrichtungssequenzen Tasten werden nicht durch einen starren Blick, sondern durch bestimmte Kombination mehrerer Blicke aktiviert.

In Fritzers Experimenten stellten sich vor allem die Off Screen Targets und eine kurze Dwell Time als geeignete Mittel der Eingabe heraus. Neben den experimentellen Ergebnissen, waren noch andere Gründe ausschlaggebend dafür, dass in unserem Entwurf des Demonstrators nur diese Eingabemethode per Dwell Time implementiert wurde. Denn die neue Software wurde ohne jegliche Anbindung an eines der bestehenden Gaze Tracker Systeme entwickelt und soll auch universell einsetzbar sein. Als standardisierte Schnittstelle wurde der gewöhnliche Systemcursor verwendet, welcher weder Koordinaten außerhalb des Monitors, noch „Blinzel-Signale“ liefern kann.

Also nimmt der hier beschriebene Cursor an, dass der Betrachter bei längerem fokussieren eines kleinen Bereichs, dort ein Klick-Ereignis auslösen möchte. Wie groß dieser Bereich sein darf und wie lange er mit seinem Blick dort verweilen muss, um ein Klicken zu provozieren, ist von dem Verwendungszweck, dem Benutzter selbst und auch der GUI abhängig. Ist die Kalibrierung der Eingabe sehr genau und damit die Anzahl der eindeutig unterscheidbaren Regionen sehr hoch, kann die Benutzeroberfläche mit kleineren Icons und Knöpfen ausgestattet werden. Kleinere Knöpfe fordern auch einen sensibleren Cursor, was eine Anpassung der Einstellungen erforderlich macht.

Ist der Benutzer noch ungeübt und hat seinen Blick noch nicht gut unter Kontrolle, muss die Verweildauer hoch sein, damit zu häufiges Verklicken unterbunden wird. Ein geübter Nutzer sähe sich andererseits in seinem Arbeitsfluss gehemmt, wenn er eine zu lange Dwell Time abwarten müsste, um ein Ziel anklicken zu können. Weil diese Faktoren von sehr vielen äußeren Einflüssen abhängen, werden sie variabel im Programm angelegt und sind während der Laufzeit individuell einstellbar.

Die Frage ist nun, wie der Cursor berechnen kann, ob die Aufenthaltszeit des Blickes in einer Region einen Schwellwert überschritten hat. Auf Grund der stetigen Wanderung des Blickes, muss diese Region, anders als bei einer unbewegten Maus, größer als ein Pixel sein. Bleibt eine Maus unbewegt auf dem Mousepad liegen, verbleibt auch der Mauszeiger unbewegt auf einem Pixel. Der Versuch per Gaze Tracker genau ein Pixel zu fixieren und diesen dann eine gewisse Zeit lang anzuschauen, muss allein schon aus körperlichen Gründen scheitern.

Ein naiver Ansatz wäre es, für jeden Pixel eine Stoppuhr bereit zu halten, die bei dem ersten Kontakt des Blickes startet und von diesem Zeitpunkt an die Dwell Time für genau diese Stelle misst. Wird die Entfernung zwischen den folgenden Blickkoordinaten und dem besagten Pixel zu groß, kann die jeweilige Uhr wieder gestoppt und auf Null zurück gesetzt werden. Ist hingegen die Dwell Time überschritten, wird ein Klick-Event mit den Koordinaten der Stoppuhr versendet.

Dieser Entwurf ist allerdings nicht praktikabel, denn er hat einige schwerwiegende Schwachstellen. Das größte Manko ist die hohe Zahl der benötigten Timer. Sie müsste im schlimmsten Fall so groß wie die in einem Kreis, mit dem Radius der maximalen Entfernungstolleranz enthaltenen Pixel sein. Bei einer realistischen Toleranz von 15 Pixel kämen bereits $\pi \cdot 15^2 \approx 707$ Timer zusammen. Bis zu 707 Timer gleichzeitig laufen zu lassen, soll von unserem System nicht verlangt werden. Alleine die Verwaltung und das ständige Starten und Stoppen, würde zu großen Schwierigkeiten führen.

Ein weiteres Problem dieser Methode ist auch der Ort und die Häufigkeit der Klicks. Wird nämlich an einer Stelle die Dwell Time überschritten, ist es sehr wahrscheinlich, dass wenige Millisekunden später im näheren Umkreis weitere Zeiten auslaufen. Denn wurde bei einem Pixel der Entfernungsschwellwert nicht überschritten, ist das Gleiche auch für seinen Nachbarn sehr wahrscheinlich. Ein Klickfeuerwerk entlang des Zeigerweges wäre die Folge. Ungewollte kurz hintereinander auftretende Ereignisse dieser Art könnten nur vermieden werden, indem bei jedem Klick alle laufenden Timer gestoppt und auf Null gesetzt werden.

Optimal wäre es also wenige oder sogar nur einen Timer zu nutzen. Hierzu muss das mathematische Modell um weitere Größen erweitert werden. Aussagekräftiger und viel leichter zu händeln als die Entfernung zwischen sämtlichen Pixeln und den aktuellen Blickkoordinaten, ist die Varianz zwischen den verschiedenen Blickpunkten.

Denn die Varianz der letzten zehn Cursorkoordinaten gibt bereits einen Überblick über den Zustand des Zeigers. Ist sie sehr groß, kann davon ausgegangen werden, dass der Cursor mit hoher Geschwindigkeit in Bewegung ist und weite Strecken zurück legt. Ein gewollter Klick zu diesem Zeitpunkt wäre sehr unwahrscheinlich.

Bei sehr kleinen Werten der Varianz, bewegt er sich hingegen in einem geringen Radius oder mit langsamer Geschwindigkeit. Jetzt könnte der Benutzer einen Klick innerhalb einer bestimmten Fläche beabsichtigen. Daher wird, sobald die Varianz einen Schwellwert unterschreitet, der Dwell Time Timer in Gang gesetzt. Während der gesamten Laufzeit aktualisiert ein festes Intervall von 50 ms, die Schlange der letzten Punkte mit einer neuen Blickkoordinate. Die älteste Ortsangabe wird ausgereiht und muss Platz für die aktuellste schaffen. In diesem Takt kann auch die Varianz neu berechnet und auf das Überschreiten einer Schwelle getestet werden. Bei einer Schlange der Länge zehn kann so beispielsweise alle 50 Millisekunden eine Aussage darüber getroffen werden, wie stark sich der Cursor die letzten 500 Millisekunden bewegt hat. Ist die Varianz zu einem Zeitpunkt zu groß, wird der Dwell Time Timer sofort gestoppt. War die Varianz während der gesamten Verweildauer klein genug, kann ein Klick-Ereignis versendet werden.

Im späteren Programm soll die Toleranz mit einem Slider einstellbar sein. Um eine quadratische Skalierung der Eingabe zu vermeiden, tritt die Standardabweichung, also die Wurzel der Varianz, an deren Stelle.

Durch das Bewegen des Sliders kann nun eine optische Rückmeldung über die maximale Toleranz gegeben werden, indem sich ein Kreis, mit dem Radius der Standardabweichung, um den Cursor entsprechend vergrößert oder verkleinert. Mit der Varianz als Maß des Radius, entstünde ein Widerspruch zwischen dem linearen Verschieben des Sliders und dem quadratischen Wachsen des Kreises.

Die Anzahl der gespeicherten Mauspositionen, zwischen welchen die Varianz ermittelt wird, kann über einen Slider mit der Beschriftung „Maus-Glättung“ verändert werden. Die Glättung bezieht sich auf den Mittelwert aller Punkte. Ist die Liste besonders lang, folgt der Mittelpunkt mit großer Verzögerung und unter starker Glättung den Bewegungen des Cursors. Bei wenigen Elementen in der Liste, ist die Verzögerung kleiner und die

Verfolgung direkter. Befindet sich nur ein Punkt in der Reihe, liegt der Mittelpunkt direkt auf den Cursorkoordinaten.

Die sehr häufige Berechnen der Varianz σ^2, der Standardabweichung σ und des Mittelwerts μ sollte sehr schnell sein und muss deshalb, wie aus der Bildverarbeitung bekannt, optimiert werden. Denn würden

$$\mu = \frac{1}{N}\sum_{i=0}^{N-1} x_i$$

und

$$\sigma^2 = \frac{1}{N}\sum_{i=0}^{N-1}(x_i - \mu)$$

nacheinander berechnet, wären wegen der beiden Summen mindestens zwei Schleifendurchläufe über alle Elemente x_i mit $i \in N$ notwendig. Durch geschicktes Umstellen und Einsetzen kann die gesamte Berechnung auf einen Durchlauf verkürzt werden:

$$\begin{aligned}
\sigma^2 &= \frac{1}{N}\sum_{i=0}^{N-1}(x_i^2 - 2x_i\mu + \mu^2) \quad \text{(Binomische Formel)} \\
&= \frac{1}{N}(\sum_{i=0}^{N-1} x_i^2 - \sum_{i=0}^{N-1} 2x_i\mu + \sum_{i=0}^{N-1} \mu^2) \\
&= \frac{1}{N}\sum_{i=0}^{N-1} x_i^2 - \frac{1}{N}2\mu\sum_{i=0}^{N-1} x_i + \frac{1}{N}\sum_{i=0}^{N-1}\mu^2 \\
&= \frac{1}{N}\sum_{i=0}^{N-1} x_i^2 - \frac{1}{N}2\mu\sum_{i=0}^{N-1} x_i + \mu^2 \\
&= \frac{1}{N}\sum_{i=0}^{N-1} x_i^2 - 2\mu^2 + \mu^2 \\
&= \frac{1}{N}\sum_{i=0}^{N-1} x_i^2 - \mu^2
\end{aligned}$$

Im Inneren der Schleife über alle Elemente der Liste brauchen nach der erhaltenen Formel nur noch x_i und x_i^2 summiert werden. Der dazugehörige Codeausschnitt sieht dann so aus:

```
Iterator i( punktliste );
while( i.hasNext() ){
    punktwert = i.next();
    summe += punktwert;
    summeQuadrad += punktwert * punktwert;
}
mittelwert = summe / punktlistenLänge;
varianz = summeQuadrad / punktlistenLänge
          - mittelwert * mittelwert;
standardabweichung = sqrt( varianz );
```

Ein Punkt im zweidimensionalen Raum hat zwei Koordinaten x und y. Sie können beide in der gleichen Schleife behandelt werden, als Ergebnis der Varianz wird aber am Schluss ihr arithmetisches Mittel zurückgegeben.

Auch diese Lösung ist gerade bei längeren Listen noch nicht optimal. Denn zwischen zwei Varianzberechnungen hat sich lediglich ein Punkt verändert. Um den Algorithmus weiter zu beschleunigen, kann von der vorherigen Summe der älteste Wert subtrahiert und der neue Wert addiert werden. Danach müssen nur noch die drei Schritte nach der Schleife wiederholt werden, um auf das gleiche Ergebnis zu kommen. Damit ist selbst die letzte Schleife überflüssig.

Wann ein Klick ausgelöst wird, kann mit dem oben gezeigten Verfahren sehr effizient ermittelt werden. Offen bleibt noch, wo genau der Klick platziert werden soll. Für die Schätzung dieses Ortes gibt es unterschiedliche Ansätze. Eine Möglichkeiten wäre es, immer den aktuellen Blickpunkt mit dem Ereignis zu verknüpfen. Da das ausgelöste Ereignis aber nicht nur aus diesem Punkt resultiert, ist es logischer, den Mittelwert aller Punkte in der Schlange, als Klickpunkt festzulegen.

Damit der Betrachter auch eine optische Rückmeldung erhält, wird für einen Augenblick ein grüner Kreis rund um den Cursor sichtbar gemacht. Außerdem wäre auch ein akustisches Signal als Vergewisserung denkbar.

Die Dwell Time zu visualisieren könnte eine weitere Unterstützung sein. Vorstellbar wäre das Schrumpfen des Cursorumkreises, das Verfärben ähnlich einer Ampel, die Vervoll-

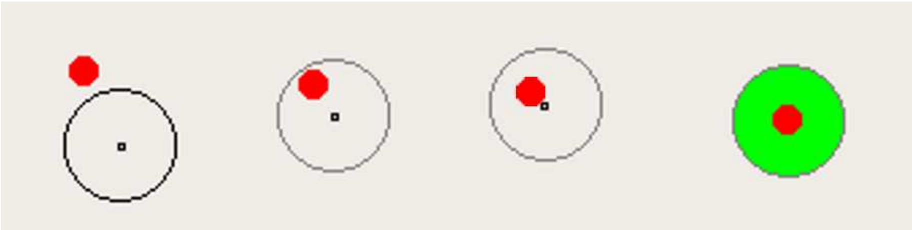

Bild 3.4: Visualisierung des Cursors von der normalen Bewegung bis hin zum Klick

ständigung eines Kuchendiagramms oder das Einblenden eines Fortschrittbalkens. Das Problem bei diesen kleinen Animationen ist aber die extrem kurze Dauer der Dwell Time. Sie variiert in etwa zwischen 100 und 1000 Millisekunden. Auch wenn sie theoretisch 1000 ms dauern würde, kann sie jederzeit durch eine zu hohe Varianz gestoppt werden. Früh unterbrochene Animationen, sind dann nur noch als unangenehmes Flackern sichtbar. Selbst bei einer standardmäßig eingestellten Zeitspanne von 500 ms, die ungestört bis zum Ende durchlaufen wird, ist jede Darstellung lediglich als kurzes Zucken wahrnehmbar.

Aus diesem Grund unterscheidet der Demonstrator nur zwei verschiedene Zustände. Ist die Varianz zu groß, umrandet den Cursor ein schwarzer Ring. Fällt die Varianz unter den angegebenen Schwellwert, so dass der Dwell Time Timer aktiviert wird, verfärbt sich der Ring in ein dunkles Grau.

Diese Methode ist zwar nicht so bunt und auffällig wie eine der vorher angesprochenen Versionen, dafür verhindert sie ständiges Aufflackern von kurz nach ihrem Beginn abgebrochenen Animationen.

Der Ring kann auch als Toleranzgrenze aufgefasst werden. Befindet sich der rote Mittelpunkt aller gelisteten Punkte in seinem Inneren, ändert er seine Farbe in Grau. Verändert sich durch Verstellen eines Slider die Toleranz, mathematisch gesehen die Standardabweichung, verändert sich auch der Durchmesser des Rings proportional zum neu eingestellten Wert. Bei der praktischen Umsetzung des Cursors, wird ein experimentell ermittelter Faktor mit der Toleranz multipliziert, damit bei der Umrechnung ins Pixelmaß geeignete Beträge für den Durchmesser entstehen.

In Abbildung 3.4 ist ein typischer Ablauf der Cursoraktivitäten gezeigt. Zuerst befindet sich der rote Mittelpunkt außerhalb des schwarzen Rings. Bleibt der Blick etwas länger in einer kleinen Region, tritt der Punkt in das Innere des Umkreises ein und dieser verfärbt sich in Grau. Die Dwell Time beginnt zu laufen und sobald das Timeout erfolgt, färbt sich die gesamte Kreisfläche grün ein. Wird die Dwell Time unterbrochen, wechselt die Farbe zurück zu Schwarz.

Einen echten Zusammenhang zwischen Mittelpunkt und Ring, gibt es so in Wirklichkeit nicht. Es ist nämlich durchaus möglich, dass sich der Mittelpunkt, trotz zu hoher Varianz, innerhalb des Kreises befindet. Beispielsweise bei einer schnellen Auf- und Abbewegung in vertikaler Richtung, durchquert der Mittelpunkt ständig den Ring, obwohl die Varianz sehr hoch ist.

Aber auch ohne harten mathematischen Hintergrund ist diese Auffassung so anschaulich, dass sie eine Erwähnung wert ist.

3.3.2 Die Heatmap

Eine andere Anwendung, die ganz ohne Klicks und komplexe Cursortechnik auskommt, ist die Heatmap. In ihr wird protokolliert wo und wie lange sich der Blick in einer bestimmten Zeitspanne aufgehalten hat.

Wie bereits in Abschnitt 3.2 auf Seite 18 erwähnt, entsteht die hier vorgestellte Heatmap durch reines Iterieren der Pixel, auf die ein Blick gefallen ist. Hierfür wird ein Integer Array mit der Größe der Bildschirmauflösung angelegt und jedem Pixel eine Zelle zugeordnet. Nach der Initialisierung ist die Komplette Matrix mit Nullen gefüllt. Streift der Blick einen bestimmten Pixel, wird der Wert in der zugehörigen Zelle um eins erhöht.

Dabei kommt wieder das Problem der räumlichen Ausdehnung einer Fixation ins Spiel. Würde nur genau die zur Blickkoordinate passende Zelle iteriert, sähe die Karte am Schluss meistens sehr leer aus. Die Wahrscheinlichkeit, dass ein Pixel wiederholt getroffen wird, wäre zu gering um aussagekräftige Bilder mit mehreren Farben zu erhalten. Daher werden nicht nur die eigentliche Koordinate, sondern auch weitere Punkte in ihrer Umgebung erhöht, sobald ein Blick auf sie fällt.

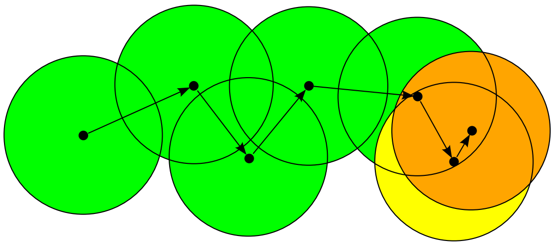

Bild 3.5: Unbeabsichtigte Darstellung: Die Farbe des gesamten Kreises richtet sich nach dem Betrag seines Mittelpunktes. Bei Überschneidungen überdeckt der zuletzt gezeichnete Kreis alle älteren.

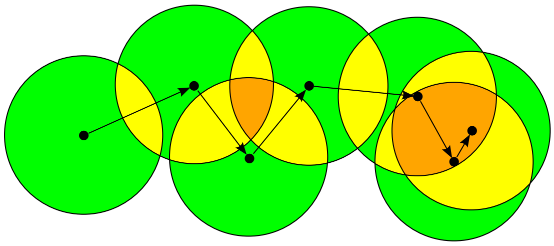

Bild 3.6: Beabsichtigte Darstellung: Jedes Pixel wird anhand seines eigenen Betrags eingefärbt. Die zeitliche Reihenfolge spielt, für die Farbwahl bei Überdeckungen, keine Rolle.

Zur Darstellung gibt es wieder verschiedene Möglichkeiten. Die eine wäre, nach dem Erhalt einer neuen Koordinate einen Kreis zu zeichnen, dessen Farbe und Mittelpunkt sich nach dem Betrag beziehungsweise dem Ort der zugehörigen Zelle richtet. Der Kreisradius stünde dabei für die angenommene Fläche, die eine einzige Fokussierung einnimmt. Anschließend könnten alle Zellen, deren Pixel nach dem Zeichnen ihre Farbe gewechselt haben, um eins erhöht werden. Für diese Umsetzung ist nur einen Paint-Befehl pro neue Blickkoordinate nötig. Das einmalige Aufrufen des Zeichenbefehls, ist im Gegenteil zu der im Anschluss vorgestellten Methode, mit sehr geringem Aufwand zu erledigen.

Wie Abbildung 3.5 zeigt, führt diese Methode leider zu einem ungewollten Ergebnis, da Überschneidungen nicht berücksichtigt werden. Zusätzlich spielt hier die Reihenfolge, in der die Kreise gezeichnet werden, eine wichtige Rolle. Würde nur das Array abgespeichert und nach Beendigung der Eingabe Zeile für Zeile in eine Karte verwandelt, entstünde ein anderes Resultat, als beim direkten Zeichnen während der Eingabe.

Es ist also unausweichlich, jeden Pixel nach der Erhöhung gesondert zu zeichnen. Um nicht bei sämtlichen Updates jedes Pixel, mit einem Zellenwert ungleich Null, neu zeichnen zu müssen, braucht nur eine Boundingbox rund um den Cursoraufenthaltsort aktualisiert zu werden. Bei dieser Lösung, wie in Bild 3.6 zu sehen, ist die Reihenfolge der Paint-Aufrufe irrelevant.

Zur praktischen Anwendung wurden zwei unterschiedliche Modi implementiert. In der

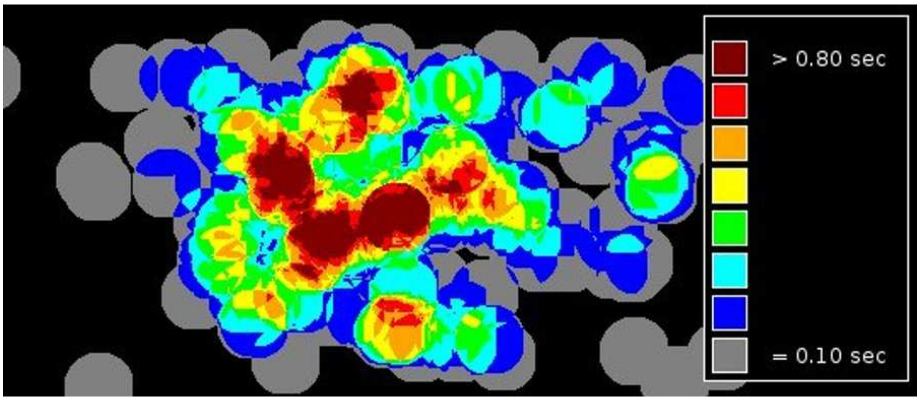

Bild 3.7: Detailansicht einer Heatmap mit Legende

ersten Einstellung, kann der Betrachter beim Entstehen der farbigen Karte zuschauen. Sie gleicht einem kleinen Malprogramm und ist nur zu Demonstrationszwecken gedacht.

Die zweite Einstellung kann in vielen Einsatzgebieten verwendet werden. In ihr können Bilder in den Hintergrund geladen und auf Knopfdruck sichtbar gemacht werden. Anschließend entsteht die Heatmap unsichtbar im Hintergrund und kann, nach Beendigung des Experimentes, dargestellt und gespeichert werden. Ein solches Verfahren kann zum Beispiel in der Softwareergonomie zur Erforschung der Benutzerfreundlichkeit einer neuen Oberfläche, in der Marktforschung, zur Produktentwicklung oder bei psychologischen Tests, eingesetzt werden.

Weil das menschliche Auge nur wenige Farben eindeutig unterscheiden kann, ist nicht für jeden möglichen Arraywert eine exklusive Farbe vorgesehen. Wie die Farben in Bezug auf die Werte und damit der Aufenthaltszeiten skaliert sind, kann dem Anlass entsprechend, während der Laufzeit eingestellt werden. Eine Legende, wie in Bild 3.7 dargestellt, macht den Zusammenhang zwischen den Farbstufen und ihren zugehörigen Zeitwerten deutlich.

Obwohl es bei den Arrayzugriffen schneller gegangen wäre, die Flächen der Blicke rechteckig zu wählen, wurden sie aus ästhetischen Gründen in runder Form gestaltet. Denn es ist wohl anzunehmen, dass trotz intensiver Bildschirmarbeit, der Blick eines PC-Nutzers nicht viereckig ist.

3.3.3 ButtonUp

Bei ersten Tests des Demonstrators in Verbindung mit dem Gaze Tracker, stellte sich heraus, dass die Genauigkeit des Trackers, durch Radialverzerrungen, an den Rändern vermindert wurde. Um diesem Phänomen zu begegnen, wurde das Design des Spiels dem entsprechend entworfen. Das Hauptgeschehen spielt sich in der Mitte des Schirms ab, da dort die höchste Präzision bei der Eingabe erreicht wird. Zusätzlich sind alle Spielelemente groß genug, um auch bei schlechter Kalibrierung getroffen werden zu können.

Ferner muss das Spiel auch ohne Mausgesten oder Tastatureingaben auskommen und ausschließlich mit den eingeschränkten Fähigkeiten des Cursors bedienbar sein. Das heißt, es stehen nur einfache Klicks und sonst keine weiteren Befehle, wie das Gedrückthalten von Tasten, zur Verfügung.

Eine ganz zentrale Anforderung an die Spielidee, ist die örtliche Bindung zwischen optischer Ausgabe und interaktiver Eingabe. Dort wo sich etwas bewegt, oder sonst wie den Blick anzieht, soll auch die Eingabe zur Steuerung erfolgen.

Ein Puzzlespiel, mit Bildschirmtastatur neben dem Spielfeld, wird daher ausgeschlossen. Dieser Aufbau würde ein ständiges Wechseln des Blickes, zwischen Tasten und Puzzle, nötig machen. Eine Solche Trennung zwischen Spiel und Bedienung, würde das Eintauchen in die augengesteuerte Welt unangenehm stören, und muss daher unbedingt vermieden werden.

Sämtliche Vorüberlegungen führten schließlich zu dem Spiel „ButtonUp“. Wie der Namen schon andeutet, handelt es sich um ein sehr einfaches Prinzip, bei dem verschiedene Knöpfe von unten ins Bild gefahren werden. Um Punkte zu sammeln, können sie angeklickt werden. Bei einem Klick ins Leere, geht ein Leben verloren und der Punktestand verringert sich um zwei Zähler. Die Punktaddition beziehungsweise -subtraktion wird jeweils mit einem kleinen Icon, am aktuellen Aufenthaltsort des Cursors, angezeigt.

Eine maximale Spieldauer oder ein maximal zu erreichender Punktestand ist nicht vorgesehen. Das Ende des Spieles ist damit nicht eindeutig bestimmt. Es wird nur nach dem Verlust aller sechs Leben automatisch erreicht und ist damit theoretisch beliebig lange heraus zu zögern.

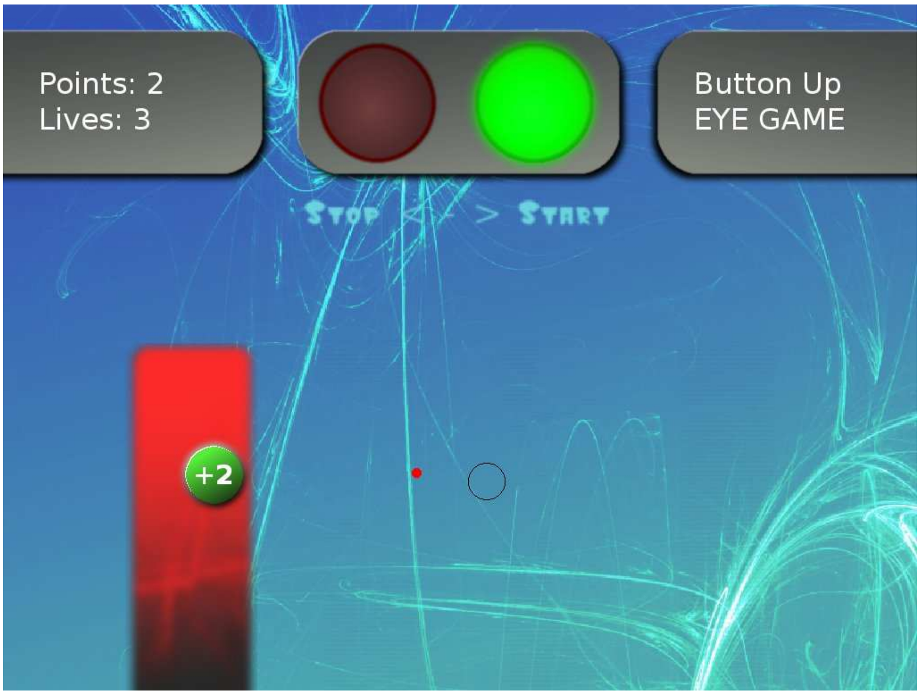

Bild 3.8: Screenshot aus ButtonUp. Der rote Button wurde getroffen und die Punktzahl um zwei erhöht. Zur Verdeutlichung dieses Erfolgs, verschwimmen die Konturen des getroffenen Ziels.

Gestartet wird das Spiel durch den Blick auf das grüne Feld einer Ampel, die sich am oberen Rand des Feldes befindet. Ein entsprechender Klick auf die rote Fläche der Ampel, bewirkt den Abbruch der Runde. Die Ampel sowie der Bereich der vier Buttons erscheinen immer horizontal zentriert, da dort die höchste Genauigkeit bei der Eingabe erreicht wird.

Damit das Spiel bei verschiedenen Bildschirmauflösungen immer den gleichen Schwierigkeitsgrad besitzt, sind die Buttongrößen und deren Abstände untereinander mit absoluten Pixelwerten festgelegt. Die Raumeinteilung sieht bei einer Bildschirmauflösung von 1024 x 768 dann wie in Abbildung 3.8 aus.

Kapitel 4

Experimente und Ergebnisse

4.1 Versuchsaufbau

Sämtliche Versuche wurden mit Thorsten Geiers Gaze Tracker unter normalen Bürobedingungen durchgeführt. Da Geiers Arbeit zur Zeit der Durchführung noch nicht beendet war, muss hinzugefügt werden, dass die Hard- und Software des Gaze Trackers auf dem Stand von Mitte Juni 2007 war.

Wie bereits erwähnt, ist die Ansteuerung der Kamera, die Bildanalyse und anschließende Ermittlung der Blickkoordinaten nicht Teil der hier vorgestellten Anwendungssoftware und damit auch nicht Teil der durchgeführten Experimente. Daher wird auf die verwendete Hardware auch nicht weiter eingegangen und nur soviel gesagt, dass nach dem Start des Gaze Trackers, die Augen des Benutzers den Systemcursor steuern.

Alle beschriebenen Tools benutzen als einzige Eingabe den normalen Mauszeiger, welcher entweder mit den Augen oder der Maus gesteuert wird. Der Demonstrator unterscheidet nicht zwischen den beiden unterschiedlichen Eingabemodi. Er läuft nur parallel auf dem gleichen Computer und geht davon aus, dass es sich bei den übermittelten Mauskoordinaten um Blickpunkte handelt.

Die nötige Kalibrierung wird in allen folgenden Experimenten als gegeben vorausgesetzt und nur in Sascha Langes Ausarbeitung genauer behandelt.

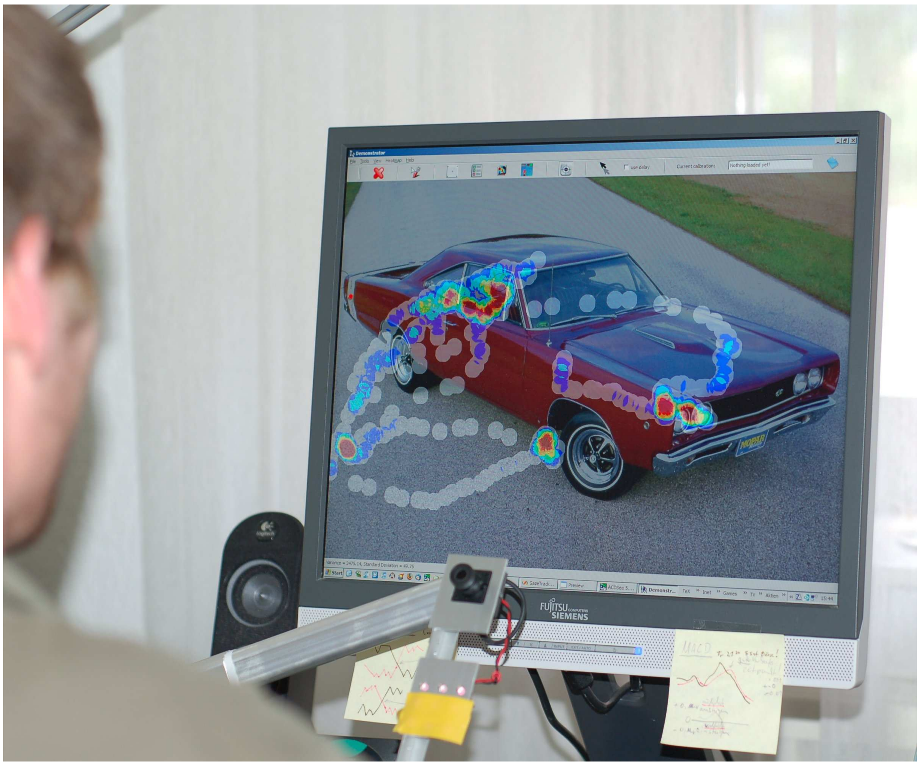

Bild 4.1: Die Heatmap im praktischen Einsatz: Ein Testbild wird mit den aufgezeichneten Blicken überlagert

4.2 Heatmap Test

Die Heatmap wurde in beiden Betriebsarten getestet. War sie auf direkte Anzeige geschaltet, konnte wie erwartet mit den Augen auf dem Bildschirm gemalt werden. Vor allem im Vollbild, ohne störende Taskleiste, konnte live bei der Entstehung der Heatmap zugeschaut werden.

Das Starten, Stoppen und Speichern der Aufzeichnung musste, auf Grund des außer Betrieb gesetzten manuell gesteuerten Cursors, mittels Shortcuts befohlen werden.

Im Bildbetrachtungsmodus wurden verschiedene Motive in den Hintergrund geladen. Durch Drücken der Space-Taste erschien das Testbild, während gleichzeitig unsichtbar die Aufnahme der Heatmap begann. Nach wiederholter Betätigung derselben Taste, wurde die Karte, überblendet mit dem vorher betrachteten Testbild, wie in Bild 4.1 angezeigt und konnte anschließend wieder per Tastenkombination gesichert werden.

Alle Zeitsummen, die über einem gewissen Grenzwert liegen, werden nach oben abgeschnitten und mit der gleichen Farbe dargestellt. Aus diesem Grund wurde bei längeren Tests die Zeitspanne zwischen den Farbstufen etwas erhöht, um zu viele dunkelrote Stellen zu vermeiden. Waren die Spannen zu kurz gewählt, entstand eine Binäre Heatmap mit den Farben Schwarz, für nicht angeschaute Regionen und Dunkelrot für berührte Bereiche.

Da der Toleranzring gleichzeitig die Dicke des Pinsels bestimmt, brachte bei den verschiedenen Cursoreinstellungen nur das Verstellen der Toleranz eine direkt sichtbare Veränderung. War sie zu groß gewählt, füllte sich die gesamte Zeichenfläche sehr schnell und die Details bei der Darstellung gingen verloren. In der größten Einstellung betrug der Durchmesser des Cursorumkreis etwa ein Viertel der Bildschirmhöhe. Bei einer solchen Größe sind $200 \cdot 200 = 40.000$ Abfragen ob der Pixel innerhalb des Kreises liegt und weiteren $\pi \cdot 100^2 \approx 31.416$ Paintaufrufe pro 50 ms zur Darstellung notwendig.

Diese enorme Menge von Anweisungen zwang den verwendeten Computer deutlich in die Knie.

4.3 ButtonUp Test

Beim anschließenden Versuch wurde das Spiel ButtonUp getestet. In seinem Bedienkonzept sind keine Shortcuts von Nöten. Alleine durch den Blick auf die grüne Lampe der Ampel, konnte das Spiel in Gang gesetzt werden. Nur nach einer besonders schlechten Kalibrierung, erreichte der Cursor nicht den grünen Bereich, so dass die Runde per Shortcut gestartet werden musste. Nach durchschnittlich einer Minute waren alle fünf Leben verloren und zwischen -5 und +30 Punkten erreicht.

Auch innerhalb des Spiels konnten die Cursoreinstellungen variiert werden. Zu starkes Verkürzen der Dwell Time zog ein unkontrolliertes Klicken ins Leere und damit eine sehr kurze Rundendauer nach sich. Gleiches galt für eine zu niedrige Toleranz oder die Einbeziehung von zu wenigen Punkten bei der Berechnung der Varianz. Die gegenteiligen

Extrama führten dazu, dass gar keine Klicks mehr ausgelöst wurden. Auffällig war auch eine Sequenz von zwei Klicks, wovon der erste den Button traf und der unmittelbar darauf folgende, auf der gleichen Stelle, zu einem Punktabzug führte.

4.4 Ergebnisse und Verbesserungsvorschläge

Bei allen Versuchen blieben negative Überraschungen die Funktionalität betreffend aus. Alle implementierten Tools reagierten erwartungsgemäß und lieferten die gewollten Ergebnisse.

Die gegebenen Voreinstellungen stellten sich, im Spiel ButtonUp zum Treffen der Ziele, als geeignet heraus. Beim Erstellen der Heatmaps musste die zeitliche Auflösung zwischen den Farben etwas erhöht werden, um bei kürzeren Tests die Aussagekraft zu erhöhen.

War die Toleranzgrenze, welche sich proportional zur Pinseldicke verhält, zu hoch gewählt, entstand sehr schnell eine komplett dunkelrot eingefärbte Karte. Auf ihr waren keine farblichen Abstufungen oder örtlichen Details mehr zu erkennen.

Das Problem, dass dunkelrote Bereiche nicht weiter verfärbt werden, könnte durch die Hinzunahme von weiteren Zwischenfarben abgeschwächt werden. Das Abschneiden von zu hohen Werten, könnte auch durch die ständige Normalisierung aller Pixel, umgangen werden. Dazu müssten alle Punkte durch den maximal vorkommenden Betrag geteilt und anschließend die Farben gleichmäßig zwischen 0 und 1 verteilt werden. Obwohl ButtonUp wie vorgesehen funktionierte, traten doch einige Schwächen in der Spiellogik zu Tage. Wird zum Beispiel ein Button getroffen, ändert er zwar seine Erscheinung jedoch nicht seinen Bewegungsablauf. Er sollte nach einem Treffen sofort den schnellen Rückzug nach unten antreten, um das positive Ereignis noch deutlicher darzustellen.

Um die mehrfach beobachteten Doppelklicks nach einem Treffer zu vermeiden, müsste die Dwell Time, sobald ein Punkt erzielt wurde, für wenige Millisekunden angehalten werden.

Insgesamt ist fraglich, ob das System der Bestrafung durch Punktabzug bei Midas-Touch, also des ungewollten Klicks ins Leere, überhaupt sinnvoll ist. Der Spielspaß wäre vielleicht höher, wenn für ein solches Vergehen nur ein Leben abgezogen würde und die Punkte erhalten blieben. Stattdessen könnte jedes Mal wenn ein Button unberührt verschwindet die Punktsubtraktion durchgeführt werden.

Das größte Manko in allen Tests war ein ganz anderes Problem. Nämlich die fehlende Möglichkeit zwischen manueller und augengestützter Steuerung umzuschalten. Solange sich die Testperson im Vollbildmodus in einer der Tools befand, verlief alles normal. Wollte sie allerdings die Ergebnisse speichern, irgendwelche Einstellungen verändern oder das Programm verlassen, wurde es kompliziert. Intuitiv ging der Griff zur Maus um das Menü zu bedienen. Ihre Eingabe wird allerdings vom Gaze Tracker überschrieben, so dass sie unbrauchbar ist. Deshalb musste immer auf Shortcuts zurückgegriffen werden. Das Wechseln zwischen Tracker und Maus kann nur außerhalb des Demonstrators geschehen, was zu einem ständigen Fensterwechsel und einem großen Tastenkombinations-Wirrwarr führt.

Sämtliche Funktionen sollten entweder mit den Augen zugänglich gemacht werden, was viele große Buttons und damit einen erheblichen Mehraufwand erforderlich machen würde, oder das Umschalten zwischen Hand- und Augeneingabe müsste innerhalb des Demonstrators ermöglicht werden.

Letzteres hört sich trivial an, ist aber nicht zu unterschätzen. Der Demonstrator kennt nämlich gar keinen Unterschied zwischen den verschiedenen Modi. Er erhält nur Mauskoordinaten auf welchen er all seine Berechnungen aufbaut. Läuft die Tracker-Software, von der aus der Wechsel möglich wäre, im Hintergrund, dringen auch keine Shortcuts bis zu ihr hindurch. Alle Befehle werden von dem Demonstrator im Vordergrund abgefangen.

Findet sich keine Lösung die Befehle zur Applikation im Hintergrund durch zu leiten, verbleibt nur noch die Möglichkeit, die Koordinaten über einen anderen Weg zu übermitteln.

Für eine Übertragung per Pipes oder Sockets müsste zwar auch die Fremdsoftware angepasst werden, der Aufwand würde allerdings mit einem großen Zuwachs von Flexibilität belohnt. Mit einer dieser Übertragungsarten, könnten zusätzlich, sowohl negative und nicht auf den Bildschirm geclippte Werte, als auch ganze Ereignisse wie beispielsweise „Blinzeln“, übertragen werden.

Kapitel 5

Zusammenfassung

Im Rahmen dieser Studienarbeit wird eine Software zur Demonstration der Fähigkeiten eines Gaze Trackers beschrieben und im praktischen Teil auch implementiert. Den Anfang macht ein Überblick über vorhandene Anwendungen an der Universität Koblenz, sowie in der freien Wirtschaft. Anschließend kommt die Sprache auf die Funktionalität des Cursors, der Klicks anhand einer Dwell Time ermittelt und im gesamten Programm einsetzbar ist. Dazu werden verschiedene Optimierungen vorgestellt und mathematisch hergeleitet.

Es folgt die Beschreibung eines Tools zur Erstellung von Heatmaps, welches in zwei unterschiedlichen Modi, entweder zur reinen Vorführung oder zur produktiven Nutzung dient. In diesem Zusammenhang wird im Detail auf die Frage der zweidimensionalen Ausbreitung eines Blickes auf dem Bildschirm eingegangen.

Ein weiterer Bestandteil des Programms ist das Spiel ButtonUp, mit dessen Hilfe auf spielerische Art und Weise mit dem Computer interagiert werden kann. Sein Design richtet sich ganz gezielt nach den Gegebenheiten des Gaze Trackers und ist damit besonders angenehm steuerbar.

Die abschließend durchgeführten Experimente, zeigen die Stärken und Schwächen des Demonstrators. Sie führen zu einigen Vorschlägen, zu denen vor allem, die Verbesserung des Imports der Blickkoordinaten zählt. Mit der erstellten Software können aber auch jetzt schon die Fähigkeiten des vorhanden Gaze Trackers publikumswirksam demonstriert werden.

Literaturverzeichnis

[AG] AMTECH GMBH, Dossenheim, www.amtech.de

[CLI] CLIP, www.intuac.com/userport/john/clip103.html

[Fri05] FRITZER, Fabian: *Texteingabe per Augenbewegung*, Universität Koblenz-Landau, Campus Koblenz, Fachbereich 4 Informatik, Institut für Computervisualistik, Diplomarbeit, 2005. file:///lab/as/Archive/1802_ffko/Thesis/Fritzer2005.pdf

[Gei07] GEIER, Thorsten: *Gaze-Tracking zur Interaktion unter Verwendung von Low-Cost-Equipment*, Universität Koblenz-Landau, Campus Koblenz, Fachbereich 4 Informatik, Institut für Computervisualistik, Diplomarbeit, voraussichtlich 2007

[LTI] LC TECHNOLOGIES INC., Virginia, www.lctinc.com

[OL] OPTOMOTORIK LABOR, Freiburg, www.optom.de

[SEA] SMART EYE AB, Sweden, www.smarteye.se

[SM] SEEING MACHINES, Australia, www.seeingmachines.com

[TRG] THOMAS RECORDING GMBH, Giessen, www.thomasrecording.com

[Tro06] TROLLTECH: *Qt 4.2 Whitepaper*. Trolltech ASA, Norwegen, 2006. www.trolltech.com

[uni] UNICAP, http://sourceforge.net/projects/unicap/