RWTH AACHEN UNIVERSITY

Diese Arbeit wurde vorgelegt am Visual Computing Institute

Current Topics in the Field of Virtual Reality
Seminar Summer Term 2017

Optimierte Graphextraktion und Pfadvorhersage für die Anwendung im Redirected Walking

Seminararbeit im Studiengang Informatik von

Sebastian Pape

Prof. Dr. Torsten W. Kuhlen
Virtual Reality & Immersive Visualization

Aachen, 13. Juli 2017

Inhaltsverzeichnis

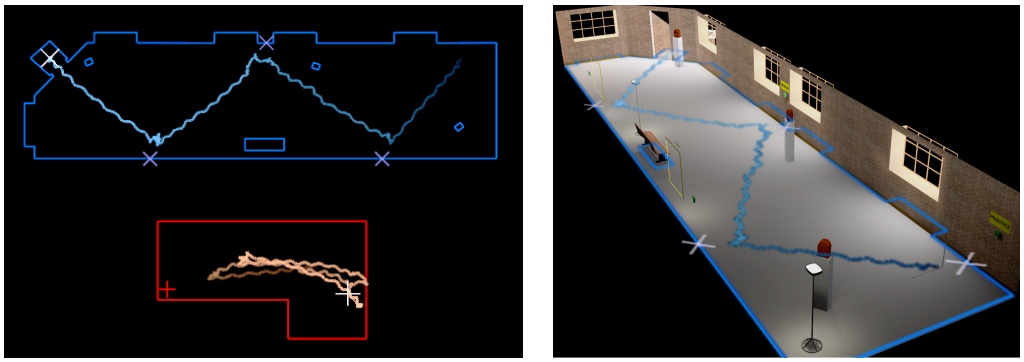
1 Einleitung

Die heutigen Techniken auf dem Markt für Head Mounted Displays erlauben dem Nutzer ein gutes Headtracking und somit eine natürliche Darstellung der virtuellen Landschaft, da die virtuelle Kamera den Kopfbewegungen folgt. Hierbei wird einem Benutzer erlaubt, sich innerhalb des Tracking Bereiches zu bewegen, welcher allerdings typischerweise sehr klein ist. Wenn die im Head Mounted Display dargestellten virtuellen Welten größer als der physisch vorhandene Tracking Bereich sind, werden oft Techniken wie Teleportationssteuerung verwendet, um dem Benutzer die Möglichkeit zu geben, die größere Welt zu erkunden. Diese stellen allerdings lediglich Substitutionen für die natürliche Fortbewegung per Laufen dar. Redirected Walking beschreibt einen Ansatz mit dem es möglich ist eine deutlich größere virtuelle Welt per physischem Laufen in einem kleineren Tracking Bereich zu erkunden. Im Nachfolgenden wird die Technik des Redirected Walking beschrieben, sowie Techniken zur Pfadvorhersage der Nutzer erläutert.

1.1 Redirected Walking

In diesem Kapitel wird der Anfang des Redirected Walking aufgezeigt und die verwendeten Techniken und Ideen dahinter erklärt. Im Anschluss daran werden Möglichkeiten zur Verbesserung der vorgestellten Algorithmen aufgezeigt, welche auf der Einschätzung des Nutzers und einer darauf basierenden Vorhersage des Nutzerpfades aufbauen.

Entwicklung: 1994 probierten Michael Moshell and Dan Mapes von der Central Univerity of Florida Nutzer ein VR System dazu zu manipulieren die Bewegungen des Benutzers in der realen Welt nicht 1:1 zu übertragen. Auf diese Weise sollte der Benutzer dazu manipuliert werden physisch eine leicht andere Bewegung als in der virtuellen Welt zu vollführen. Durch geschickten Einsatz dieser Diskrepanzen in der Übertragung kann der Benutzer in gewissem Maße in der realen Welt gelenkt werden um die Wände des Raumes nicht zu berühren. Damals scheiterte die Umsetzung der Idee daran, dass den Versuchspersonen schlecht wurde, was unter anderem auf technischen Einschränkungen des Tracking Systems beruht haben könnte [Razzaque et al., 2001]. Razzaque et al. kamen im Jahr 2001 auf diese Idee zurück, da sich die Tracking Systeme weiterentwickelt hatten und nun genauere Werte in einer höheren Frequenz liefern konnten. Sie prägten den Begriff "Redirected Walking" und stellten mehrere Verfahren vor, mit denen es in kontrollierten Szenarien möglich sei, eine virtuelle Welt, welche deutlich größer als die vom Tracker erfasste Fläche sei, per



(a) Maßstabsgetreue Abbildung des Nutzerpfades in der virtuellen Welt (blau) neben dem Pfad in der realen Welt (rot)

(b) Pfad des Nutzers in der virtuellen Welt

Abbildung 1.1: Versuchsaufbau von Razzaque et al. [2001]. In dem dargestellten Szenario sollen Nutzer eine Brandschutzübung darstellen und an den mit X markierten Stellen Knöpfe drücken. Bilder entnommen aus [Razzaque et al., 2001]

physischem Laufen zu erkunden [Razzaque et al., 2001].
Erste Erfolge konnten Razzaque et al. mit ihrem Aufbau erzielen, der Nutzer in ihrem $4\,m \times 10\,m$ Tracking Bereich in einem circa doppelt so großen virtuellen Raum eine Brandschutzübung durchführen ließ (Vergleiche Abbildung 1.1). Für diese Übung sollten die Probanden nacheinander vier Knöpfe an den Wänden drücken, welche verschiedene Funktionen in diesem Szenario erfüllten und so platziert waren, dass die Probanden im Idealfall ein Zick-Zack Muster durch den Raum liefen. In der realen Welt wurden die Nutzer so umgelenkt, dass sie bei jedem Knopf eine 180° Drehung vollführten um anschließend auf einer möglichst geraden Linie zurück zu laufen [Razzaque et al., 2001].
Wie an dem speziellen Aufbau zu erkennen ist, war dieses Szenario nicht zufällig gewählt. Bei jeder Drehung der Nutzer wird diese in einer skalierten Form in die virtuelle Welt übertragen und durch das Zick-Zack Muster wurde gewährleistet, dass die Probanden sich selbst genug drehen würden um eine ausreichende Diskrepanz zwischen virtueller und realer Welt erzeugen zu können. Zusätzlich wurde durch die Aufgabe einer Brandschutzübung möglicherweise der Proband stark genug abgelenkt was eventuell stärkere Diskrepanzen zulassen könnte. Sie führten selbst einen Test durch, ob das selbe Experiment auch in einem schmaleren Tracking Bereich funktionieren würde und kamen zu dem Ergebnis, dass beim einem Bereich von $3\,m \times 12\,m$ keiner der Probanden den Test vollführen konnte ohne gegen eine der Begrenzungen zu stoßen.
Trotz der dafür vorbereiteten Szene und einem relativ großen Tracking Bereich kommt es in realen Anwendungen immer wieder dazu, dass Nutzer Wege gehen bei denen Redirected Walking nicht funktioniert, da der Algorithmus den zukünftigen Weg nicht vorhersehen kann und deshalb die falschen Maßnahmen ergreift.

Die momentane Forschung auf dem Gebiet des Redirected Walking gehen in mehrere Richtungen. Die Modelle werden verbessert mit denen die Nutzer eingeschätzt und damit die zuvor geplanten Maßnahmen basierend darauf verbessert werden können [Zank und Kunz, 2015; Gandrud und Interrante, 2016]. Die Forscher einer anderen Richtung arbeiten an einfach zu verwendenden und generell anwendbaren Algorithmen um Redirected Walking in jeder virtuellen Welt möglichst optimal nutzen zu können. Dies könnte dazu führen, dass die Technik später für eine breitere Masse eingesetzt werden könnte, ohne, dass sich Entwickler im Detail mit der Technik auseinander setzen müssen [Zmuda et al., 2013; Azmandian et al., 2016; Zank und Kunz, 2017; Nescher et al., 2014]. Weitere Arbeiten befassen sich mit den psychischen Effekten des Redirected Walking auf den Nutzer, was allerdings für diese Arbeit nicht weiter betrachtet werden soll.

Redirection Techniques (Umlenkoperationen) Schon Razzaque et al. nannten mehrere Möglichkeiten, eine Diskrepanz zwischen Bewegung in der virtuellen Welt und Bewegung in der realen Welt zu erzeugen [Razzaque et al., 2001]. In ihrer Studie wiederum nutzen sie lediglich zwei Techniken zur Rotationsmanipulation für diese Zwecke. Bei der ersten dieser Techniken wird, sobald der Benutzer eine Drehung in der horizontalen Ebene vollführt diese Bewegung skaliert. Das kann dazu führen, dass Nutzer, die sich 180° in der virtuellen Welt drehen, in Wirklichkeit eine $\sim 270°$ Drehung vollführen (*Rotation Gain*) [Steinicke et al., 2010]. Razzaque et al. berichten sogar davon, dass eine Abwandlung dieser Methode angewandt werden kann, um die Welt zu drehen wenn die Nutzer stillstehen, die Welt sich also ohne ihr Zutun langsam um sie herum dreht. Steinicke et al. [2010] fanden in ihren Experimenten zu den Grenzen dieser Technik heraus, dass sich Nutzer 49% mehr oder 20% weniger drehen lassen ohne das es diesen signifikant auffällt.
Die zweite von Razzaque et al. [2001] verwendete Technik besteht darin, für jede Vorwärtsbewegung eine leichte Drehung einzuführen und den Nutzer somit eine Kurve laufen zu lassen (*Curvature Gain*). Diese Technik wird später in Algorithmen wie FORCE oder MPCRed verwendet, da diese somit eine Kurve planen können, ohne auf horizontale Kopfbewegungen des Nutzers angewiesen zu sein. Steinicke et al. [2010] fanden in ihren Experimenten heraus, dass ein möglicher Radius für so eine Kurve nur $3.3\,m$ betragen kann. Dies wird von den Benutzern bemerkt, irritiert diese allerdings nicht so stark, dass sie ihre Aufgabe nicht erfüllen können. Der Radius für eine unbemerkte Ablenkung ist laut Steinicke et al. [2010] bei etwa $22\,m$, was allerdings für viele Räume zu groß sein dürfte.
Neben diesen Methoden zur Rotation nennen Razzaque et al. [2001] die Möglichkeit die Vorwärtsbewegung zu skalieren um den Benutzer schneller oder langsamer als in der realen Welt laufen zu lassen (*Translational Gain*). Die Möglichkeit wurde später von weiteren Teams auf ihre Vorteile und Probleme hin untersucht. Das Ergebnis dieser Untersuchungen war, dass eine erhebliche Steigerung der Bewegungsgeschwindigkeit (x10) zwar von den Benutzern bemerkt wird, allerdings keinen signifikanten Einfluss auf die Navigationsfähigkeit und auftretendes Unwohlsein der

Probanden zeigte [Interrante et al., 2007; Williams et al., 2006]. Steinicke et al. [2010] fanden als Grenzen für den unbemerkten Einsatz dieser Technik eine mögliche Verlangsamung um 14% und eine Steigerung der Bewegungsgeschwindigkeit um 26%. Hierbei ist zu beachten, dass eine Steigerung stärker als die Verlangsamung sein kann, was sie auf Studien zurückführen welche aussagen, dass Nutzer normalerweise Distanzen in virtuellen Umgebungen unterschätzen und ihnen somit eine Verlangsamung stärker auffällt.
Die letzte Möglichkeit, welche vielfach verwendet und auch schon von Razzaque et al. genannt wurde, ist eine Möglichkeit, um den Nutzer um eine große Gradzahl zu drehen. Dies wird benötigt falls der Benutzer eine Bewegung vollführt bei der die verwendeten Umlenkoperationen den Benutzer nicht davor bewahren können gegen die Wände des physischen Raumes zu laufen. Somit wird diese Technik genutzt um Benutzer im Notfall mit ihrer Mithilfe neu auszurichten. Diese Techniken werden *Resets* genannt und sind nur im Notfall anzuwenden, da hier der Nutzer meistens explizit dazu aufgefordert werden muss sich im Kreis zu drehen was einen Eingriff in die Immersion bedeutet [Razzaque et al., 2001; Williams et al., 2007]. Hier gibt es ebenfalls die Möglichkeit diese Operation in die virtuelle Welt einzubauen indem der Benutzer durch andere Einflüsse zu einer plötzlichen großen Drehung gebracht wird.

Algorithmen Im Anwendungsgebiet des Redirected Walking gibt es mindestens vier weit verbreitete Algorithmen die unterschiedliche Vorteile bieten. Diese Algorithmen sollen zu jedem Zeitpunkt den Nutzer von den Wänden/Grenzen des Tracking Bereiches fern halten. Hierzu ist die Unterscheidung in zwei Gruppen für diese Algorithmen wichtig: Algorithmen, die die gerade verfügbaren Daten in eine Umlenkoperation umrechnen oder Algorithmen, welche die Vergangenheit sowie die virtuelle Welt (und eventuell zusätzliche Daten von anderen Nutzern) mit in Betracht ziehen, um vorherzusehen was der Nutzer als nächstes tun wird. In dieser Arbeit werden *Steer to Center* und *Steer to Orbit* stellvertretend für die erste Kategorie vorgestellt, da diese ähnlich gut oder besser in Studien abschnitten als andere Algorithmen dieser Kategorie wie *Steer to Multiple Targets* oder *Steer to Multiple Targets and Center* [Hodgson und Bachmann, 2013]. Weiterhin wurden FORCE und MPCRed für die zweite Kategorie ausgewählt, da diese zwei verschiedene Ansätze verfolgen und für das Hauptkapitel relevant sind.
Steer to Center bezieht in seiner einfachsten Form nur die gerade aufgenommenen Daten in die Berechnungen mit ein und probiert den Nutzer zu jeder Zeit zur Raummitte zu lenken. In den meisten Fällen geschieht dies mit *Rotation-* oder *Curvature Gain*, welcher bei einer Drehung zur Raummitte hin stärker und bei einer Drehung von der Raummitte weg kleiner wird. Dieser Algorithmus wird in mehreren Studien als Basiswert genutzt (z.B. [Nescher et al., 2014; Zmuda et al., 2013]), da er besser oder ähnlich gut wie andere *Steer to Target* Algorithmen funktioniert [Hodgson und Bachmann, 2013].

Steer to Orbit nutzt ähnlich wie Steer to Center in der einfachsten Implementierung nur die gerade verfügbaren Trackingdaten. Während Steer to Center den Nutzer immer wieder in die Mitte lenkt, was je nach Bewegung des Nutzers sehr viel Platz brauchen kann, versucht Steer to Orbit den Nutzer auf eine Kreisbahn um die Raummitte zu lenken, was theoretisch weniger Platz benötigen könnte. In der Praxis führt dies allerdings nicht zu signifikant weniger Zusammenstößen mit den Wänden [Zmuda et al., 2013].
Wie Nescher et al. [2014] zeigten, führt ein Algorithmus, welcher die Techniken dynamisch wechselt, um zu jedem Zeitpunkt die beste auszuwählen, zu einer signifikanten Verbesserung gegenüber den Algorithmen, welche nur eine der Techniken verwenden [Nescher et al., 2014; Zank und Kunz, 2015]. Deshalb werden hier neben diesen Ansätzen zur Berechnung der Umlenkoperation, welche völlig unabhängig von der virtuellen Szene genutzt werden können, weitere Algorithmen, welche die Topologie der virtuellen Welt einbeziehen, um im voraus die nächsten Schritte des Nutzers einschätzen zu können, vorgestellt.
MPCRed (Model Predictive Control - Redirection) wurde in [Nescher et al., 2014] vorgestellt und ist ein Algorithmus, welcher für die Planung von Umlenkoperationen ein Optimierungsproblem definiert. Hierbei wird lediglich der letzte Zustand (von der letzten Berechnung) und der jetzige Zustand in die Berechnungen einbezogen, um die nächste Umlenkoperation aus einer Menge von vordefinierten Operationen auszuwählen. Jeder möglichen Operation werden Kosten zugewiesen und diese anschließend minimiert/optimiert um die beste Operation zu errechnen. Hierbei ist zu beachten, dass mit der Stärke der Umlenkoperation auch ihre Kosten steigen und die Reset-Operationen extrem hohe Kosten zugewiesen bekommen, damit diese niemals anderen Operationen vorgezogen werden können. Da in diesem Algorithmus für jede Auswahl einer Aktion ein Optimierungsproblem gelöst werden muss, erfordert er eine signifikant höhere Rechenzeit, welche vom Planungshorizont, der Anzahl an definierten Aktionen und der Anzahl möglicher Pfade der virtuellen Umgebung anhängt. Da zumindest Letzteres von der Position des Nutzers in der virtuellen Welt abhängt, kann zuvor wenig abgeschätzt werden, wie viel Rechenzeit benötigt wird. In den Experimenten von Nescher et al. [2014] wurde allerdings gezeigt, dass eine Berechnung bzw. Aktualisierung der momentanen Umlenkoperation vergleichsweise wenig zeitkritisch ist und nebenläufig zum Hauptthread mit beispielsweise $0.4\,Hz$ laufen kann, da eine Operation sowieso so lange brauche um einen Effekt zu erzielen. MPCRed wurde gegen Steer to Center getestet und zeigte signifikante Verbesserung in der Anzahl der Reset-Operationen (-41%) und der durchschnittlichen Stärke der Umlenkoperationen (-27%) gegenüber diesem [Nescher et al., 2014].
FORCE (Fully Optimized Redirected Walking for Constrained Environments) wurde von Zmuda et al. [2013] vorgestellt und betrachtet ähnlich wie MPCRed mehr Daten als die momentane Position und Bewegung des Nutzers im Raum. FORCE betrachtet ähnlich wie MPCRed die Auswahl der nächsten Umlenkoperation als Optimierungsproblem, welches es durch eine Tiefensuche zur Laufzeit löst. Zmuda et al. [2013] schlagen vor, FORCE nur in virtuellen Umgebungen einzusetzen, welche dem Nutzer starke Einschränkungen im Sinne von vorgegebenen Wegen aufzeigen

und für alle anderen Umgebungen wie freie Flächen *Steer to Center* oder *Steer to Orbit* zu verwenden. Diese Einschränkung basiert darauf, dass FORCE die Optimierung anhand von vorgeschlagenen Wegen durchführt und es auf einer freien Fläche zu viele davon geben würde um effizient zu arbeiten. Deshalb würde hier Steer to Center signifikant weniger Rechenzeit benötigen, während FORCE ohnehin wenig Informationen über die virtuelle Welt in die Berechnungen einfließen lassen könnte. Weiterhin geht FORCE von einem vorhanden *Path Prediction Module* aus, welches dem Algorithmus zu jeder Zeit die Bewegungsmöglichkeiten des Nutzers aufzeigt, welche beispielsweise in einer Offline Phase berechnet werden könnten. Basierend auf den vom Path Prediction Module gegeben Pfaden führt FORCE eine Tiefensuche bis zu einer bestimmten Tiefe um anschließend den Pfad und die passenden Umlenkoperationen mit dem besten Resultat zu wählen. Hierbei werden für jede Tiefe durch mehrere verschiedene Umlenkoperationen simuliert um basierend darauf die nächste Tiefe zu simulieren. Dadurch wird intern ein Baum an möglichen Pfaden und damit verbundenen Umlenkoperationen aufgespannt, welcher, sofern ein Pfad valide ist, auf der maximalen Suchtiefe, als Blätter des Baumes, verschiedene Endpositionen und Ausrichtungen hat. Diese Blätter werden anschließend mit einer Evaluationsfunktion auf ihre Güte überprüft und der Pfad im Baum mit dem besten Ergebnis gewählt. Hierbei wird von Zmuda et al. [2013] vorgeschlagen, als Evaluationsfunktion eine Distanzfunktion bis zur nächsten Wand in Sichtrichtung des Nutzers in der realen Welt zu wählen, da diese einfach zu berechnen ist und mit steigender Distanz zur nächsten Wand in Sichtrichtung die Möglichkeiten für weitere Umlenkoperationen steigen.
Ihr genereller Ansatz lässt viele Einzelheiten offen, welche angepasst an die Situation weiter optimiert werden können. Neben besseren Evaluationsfunktionen lässt FORCE die Möglichkeit offen, für jeden der gegebenen Pfade aus dem *Path Prediction Module* eine Wahrscheinlichkeit anzugeben und diese in die Entscheidung mit einzubeziehen. Hierdurch ergibt sich beispielsweise die Möglichkeit, Daten von anderen Benutzern in die Wahrscheinlichkeiten einfließen zu lassen und somit die Vorhersage für den momentanen Nutzer zu verbessern, basierend auf der Annahme, dass sie sich ähnlich verhalten würden. Weiterhin bietet FORCE die Möglichkeit, die Funktion zur Validierung eines Pfades auszutauschen, was erlaubt, Redirected Walking in Räumen mit Hindernissen oder ungewöhnlichen Formen zu verwenden.
Die Rechenzeit von FORCE ist durch die Tiefensuche exponentiell, hängt allerdings stark von den verwendeten Parametern ab. In einem Beispiel mit einer durchschnittlichen Notebook-Ausstattung (Intel Core i5, 4GB RAM) schafften sie mit einer nicht optimierten Version ihres C++ Codes und kleinen Parametern (Eine Suchtiefe von 5, zwei Verzweigungen an jedem Punkt und drei mögliche Umlenkoperationen an jedem Punkt) eine Rechenzeit von $80\,ms$ ($12.5\,Hz$), was ähnlich wie bei MPCRed ausreichen sollte, da die getroffenen Entscheidungen des Algorithmus erst einmal Wirkung zeigen müssten, bevor diese neu berechnet werden sollten [Zmuda et al., 2013].

1.2 Path Prediction

Wie schon im vorherigen Abschnitt beschrieben, kann Redirected Walking besser werden, wenn der Laufweg des Nutzers vorher feststeht. Da dies in realen Anwendungsszenarien kaum der Fall ist, muss der Benutzer vorher bzw. währenddessen eingeschätzt und der zukünftige Weg errechnet werden. Unterschiedliche Teams hatten verschiedene Herangehensweisen an dieses Problem, welche sich in zwei Kategorien zusammenfassen lassen: Entweder wird die Bewegung der Vergangenheit aufgezeichnet und basierend darauf der zukünftige Pfad extrapoliert, oder es wird nur die momentane Bewegung gemessen und basierend darauf der zukünftige Pfad errechnet.
Wie Hicheur et al. [2007] herausfanden, ist menschliche Pfadfindung stereotypisch, was dazu führt, dass verschiedene Nutzer in der gleichen Situation eine hohe Wahrscheinlichkeit aufweisen die selben Entscheidungen in der Wegfindung treffen. In ihren Experimenten weichten die Pfade eines einzelnen Probanden oft nur wenige Zentimeter vom Durchschnittspfad aller Nutzer ab [Hicheur et al., 2007]. Durch diese Vergleichbarkeit der Nutzer können alle Möglichkeiten der Pfadvorhersage mit Daten von vorherigen Nutzern in der gleichen Situation verbessert werden.

Extrapolationsmethoden Da es mehrere Modelle für menschliche Bewegungen gibt, um beispielsweise zu errechnen wo sich eine Person, die auf einem Überwachungskamerabild plötzlich verdeckt wird, am wahrscheinlichsten weiter bewegen wird [Bruce und Gordon, 2004], oder Modelle, die von Robotern genutzt werden können, um Menschen zu umfahren [Yen et al., 2008], gibt es mehrere Herangehensweisen, eines dieser Modelle für den Einsatz in VR-Systemen zu nutzen. Hierbei wurde z.B. das Modell von Arechavaleta et al. [2008] genutzt, für das basierend auf der Annahme, dass Menschen einer Art von Optimalitätsprinzip in ihrer Pfadfindung folgen, einen Pfad erzeugt. Dieser Pfad kann anschließend mit den möglichen Pfaden verglichen werden um beispielsweise herauszufinden, welche Tür auf einem Flur der Nutzer wählen wird. Dies führt allerdings dazu, dass ähnlich zu MPCRed eine Rechenzeit eingeführt wird, welche direkt von der momentanen Position innerhalb des Szenarios abhängt und dementsprechend stark fluktuieren kann.
Um die generierten Pfade miteinander zu vergleichen nutzen sie als eine Möglichkeit eine Kostenfunktion, welche auf Bellman's Prinzip der Optimalität beruht. Hierbei wird angenommen, dass jedes Teilstück einer optimalen Lösung wiederum selbst die optimale Lösung für das jeweilige Teilstück darstellt. Basierend darauf stellt jede Abweichung von den möglichen "optimalen" Pfaden im Szenario eine Kostenerhöhung dar.
Als eine weitere Möglichkeit, die generierten Pfade mit den vorgegebenen zu vergleichen, testeten Zank und Kunz [2015] den Algorithmus des *Dynamic Time Warping*. Dieser Algorithmus wird in Gebieten der Sprach- und Gestenerkennung eingesetzt und basiert auf dem Vergleich zweier Wertefolgen via Backtracking. In diesem Szenario sind die Wertefolgen als Punkte in $\mathbb{R}^2$ zu verstehen. Der Algorithmus wird genutzt um vorgefertigte Pfade mit dem momentanen Pfad zu vergleichen. Dyna-

mic Time Warping funktioniert, indem es probiert, die zwei Wertefolgen aufeinander abzubilden und erlaubt dabei die Signale zu strecken und zu stauchen. In der Stimmerkennung findet dies Anwendung bei lang oder kurz ausgesprochenen Silben oder Wörtern. Für Pfadvergleiche eignet sich dieses Verfahren ebenfalls, da verschiedene Nutzer eine unterschiedliche Geschwindigkeit aufweisen können, die aufgenommenen Daten von ihnen allerdings die gleiche zeitliche Auflösung besitzen. Somit könnte eine nicht gestreckte/direkte Abbildung der Wertepaare aufeinander zu falschen Ergebnissen führen. Am Ende muss das Ergebnis des Vergleiches normiert werden, da die Größe des Fehlers nicht von der Länge der Wegstrecke abhängen sollte.
Schlussendlich verglichen Zank und Kunz diese und weitere Möglichkeiten und kamen zu dem Ergebnis, dass das Modell von Arechavaleta et al. [2008] in Kombination mit der Kostenfunktion zum Vergleichen der Pfade zu den besten Ergebnissen führt, die Rechenzeit allerdings stark von der Anzahl der möglichen Entscheidungen, die ein Proband treffen kann, abhängt, da hierfür ein Optimierungsproblem gelöst werden muss [Zank und Kunz, 2015].

Methoden basierend auf momentaner Bewegung Sowohl Zank und Kunz, als auch Gandrud und Interrante [2016] machten Versuche um mit der momentanen Bewegungsrichtung oder Blickrichtung des Nutzers den zukünftigen Pfad vorherzusagen. Hierbei wurden Annahmen gemacht, welche das menschliche Verhalten einschränken, da sie z.B. niemals davon ausgehen, dass Menschen rückwärts laufen oder willkürlich in eine andere Richtung schauen als sie laufen.
Gandrud und Interrante stellten in ihren Versuchen einen einfachen Aufbau mit einem langen Flur und zwei Türen zu beiden Seiten am Ende des Flures auf. Hinter den Türen sollte von den Probanden jeweils eine Aufgabe gelöst werden. Das Ziel des Versuches war es vorherzusagen welche Tür die Probanden als erstes wählen. Sie testeten drei mögliche Indikatoren anhand deren sie die Tür auswählten. Die simpelste Möglichkeit ist hier die Position des Probanden im Raum von den Headtracking Daten, welche ohnehin für die korrekte Projektion verwendet werden, auszulesen und als 2D-Position auf dem Boden zu betrachten. Eine Erweiterung dieser Methode ist ebenfalls die Orientierung des Kopfes, welche genauso wie die Position des Kopfes ohnehin für die Projektion gemessen werden muss, für die Vorhersage zu nutzen. Als kompliziertestes und teuerstes Werkzeug nutzten sie einen *Eye Tracker* um die Blickrichtung der Probanden aufzunehmen.
Die so aufgenommenen Werte werden anschließend als Vektoren im 2D-Raum aufgefasst und der Winkel zwischen ihnen und Vektoren vom Standpunkt des Probanden zu den Zielen als Vergleichswert genutzt. Hierbei steht ein kleinerer Winkel für eine höhere Wahrscheinlichkeit. Das Ergebnis dieses Versuches ist, dass sowohl die Kopforientierung als auch die Blickrichtung eine statistische Signifikanz für die Vorhersage der Zieles in den Daten zeigten. Somit kann sowohl mit der Kopfausrichtung als auch mit der Blickrichtung die zukünftige Bewegungsrichtung des Nutzers vorhergesagt werden. Die Kopfrichtung wurde von den Probanden erst nach der Blickrichtung angepasst, weshalb die Blickrichtung frühere Ergebnisse als die Kopfrichtung liefert.

Wiederrum liefert die Kopfrichtung aufgrund ihrer höheren Trägheit im Vergleich zu den Augen etwas robustere Daten [Gandrud und Interrante, 2016]. Im Hinblick auf die Kosten und die von Gandrud und Interrante [2016] beschriebenen Schwierigkeiten der Kalibrierung des Blickrichtungssensors bleibt fraglich, ob diese Möglichkeit einen signifikanten Mehrwert durch ihre etwas früheren Aussagen gegenüber der leicht zu messenden Kopforientierung hat.

Für die in diesem Kapitel beschriebenen Methoden werden immer Zielpunkte benötigt. In den hier genannten Versuchen wurden diese Ziele per Hand markiert und mögliche Pfade per Hand eingezeichnet (e.g. [Razzaque et al., 2001; Gandrud und Interrante, 2016; Zank und Kunz, 2015]). Da das Einzeichnen der Pfade in großen virtuellen Welten sehr viel Arbeit erfordert, wird im Hauptteil der Arbeit die automatische Generierung dieser Pfade, basierend auf Algorithmen, die schon seit einiger Zeit für die Navigation von virtuellen Figuren in ihren Welten genutzt werden, adressiert.

2 Algorithmus zur automatischen Graphextraktion

Für die Technik des Redirected Walking wird mit Algorithmen wie FORCE und MPCRed eine Karte der virtuellen Welt benötigt. Diese Karte in Form eines Graphen, bestehend aus Knoten und Kanten die diese verbinden, könnte vom Designer der virtuellen Welt per Hand erstellt werden. Da dies in großen Szenen viel Arbeit bedeutet, ist eines der Ziele in diesem Forschungsgebiet, die Markierung von möglichen Laufwegen zu automatisieren. Der hier vorgestellte Algorithmus erstellt aus der virtuellen Welt einen Graphen, welcher die möglichen Laufwege eines Nutzers darin repräsentieren soll. Diese Art von Graphen wird auch Skeleton Graph genannt, da sie sich ähnlich einem Skelett durch die virtuelle Welt zieht. Bevor der Algorithmus vorgestellt wird, muss klar sein, wie der Skeleton Graph, also das Ergebnis, aussehen sollte.

Ziele: Ein Skeleton Graph, wie er beispielsweise für MPCRed und FORCE benötigt wird, sollte optimalerweise die virtuelle Welt vereinfachen um die Rechenzeit der Redirected Walking Algorithmen zu verkürzen, aber dennoch möglichst genau die Pfade des Benutzers abbilden können. Zank und Kunz [2017] definieren daher 5 Punkte, die ein guter Algorithmus zur Graphextraktion einhalten sollte:

1. Der Skeleton Graph sollte automatisch erzeugt werden können, die gesamte begehbare Fläche der virtuellen Umgebung widerspiegeln und für Redirected Walking nutzbar sein.

2. Der Skeleton Graph folgt allen Pfaden der virtuellen Umgebung, schneidet aber niemals deren Wände.

3. Der Skeleton Graph hat keine Knicke in Pfaden, welche diese abwickelt, wenn an den Stellen nicht zu erwarten ist, dass der Nutzer sich tatsächlich in diesem Winkel dreht. Dies stellt erst einmal kein Problem da, solange der Pfad nicht durch Wände geht, allerdings führt dieser Aufbau in Algorithmen wie FORCE und MPCRed zu Problemen, da diese annehmen, dass sich an jedem Knick im Skeleton Graph der Benutzer drehen wird, weshalb die Algorithmen das Potential der Umlenkung des Nutzers stark überschätzen.

4. Zur Laufzeit können mögliche Wegpunkte generiert werden zu denen der Nutzer gehen könnte.

5. Designer der virtuellen Umgebung können eigene Ziele hinzufügen. Spiegelt die virtuelle Umgebung beispielsweise eine Piratenstadt wider, wären Schatzkisten Ziele, die nicht zur direkten Umgebung gehören, jedoch den Nutzer anziehen und somit die Wahrscheinlichkeit in diese Richtung zu laufen signifikant erhöhen.

Obwohl der erste Punkt der wohl grundlegendste Punkt ist, wird für diese Arbeit eine Einschränkung getroffen: Der vorgestellte Algorithmus arbeitet auf 2D-Daten. Durch eine mehrschichtige Struktur können damit auch 3D-Welten abgebildet werden, welche Höhenunterschiede aufweisen, was in dieser Arbeit allerdings nur oberflächlich betrachtet wird. Weiterhin gibt es eine Einschränkung auf statische Welten. Zank und Kunz [2017] erwähnten dieses Problem in ihrer Arbeit und verwiesen darauf, dass dieser Algorithmus mehrmals auf alle möglichen Kombinationen der sich bewegenden Objekte angewendet werden könne. Somit könnte für jede Kombination der sich bewegenden Objekte eine eigene Skeleton Graph Instanz erstellt und diese dynamisch getauscht werden.
Neben den oben genannten Eigenschaften des Ergebnisses sollte für den Algorithmus gelten, dass dieser einfach auszuführen sein sollte. Dies kann am einfachsten erreicht werden indem er als Plugin in eine der vorherrschenden Game Engines integriert wird. Die Autoren von [Zank und Kunz, 2017] haben hierzu ein Skript zur Integration in den Editor der Unity Engine[1] erstellt. Durch dieses Plugin wird nicht nur dem Programmierer/Designer erlaubt, den Skeleton Graph aus der Umgebung zu generieren, sondern ebenfalls eigene Ziele zu den generierten Zielen hinzuzufügen.
Zank und Kunz [2017] teilten ihren Algorithmus in zwei Teile: Der *Offline*-Teil, der den Skeleton Graph erstellt und optimiert, und der *Online*-Teil, der während der Bewegung des Nutzers benötigte Daten aus dem Skeleton Graph extrahiert. Diese Aufteilung basiert darauf, dass die Generierung des Skeleton Graph lediglich einmal erfolgen muss und dies während der Online-Phase zu viel Zeit benötigen würde.

Verwandte Arbeiten Neben der in dieser Arbeit vorgestellten Methode zur Graphextraktion von Zank und Kunz [2017] existierten schon weitere Herangehensweisen an dieses Gebiet beispielsweise von Azmandian et al. [2016], welche einen lokalen Navigationsgraphen zur Laufzeit aufstellen. Im Gegensatz zu Zank und Kunz [2017] generieren sie zur Laufzeit einen lokalen Graphen um den Nutzer herum, statt Informationen aus einem globalen zu extrahieren. Für dieses Vorgehen nutzen sie ebenfalls die Unity Game Engine und extrahieren aus einer von Unity generierten Vereinfachung der virtuellen Welt potentielle Pfade. Dies geschieht, indem sie, beginnend bei der Position des Nutzers, eine abgewandelte Version des Dijkstra-Algorithmus anwenden, welcher nach einer bestimmten Distanz abbricht. Die generierten Pfade beginnen somit bei der Position des Nutzers, reichen bis zu einer definierten Distanz und bestehen aus zusammenhängenden begehbaren Flächen in der virtuellen Welt. Dies führt wie von Zank und Kunz [2017] angemerkt

[1] https://unity3d.com/

dazu, dass der Graph, je nach Triangulierung der Umgebung, viele Knicke aufweist, welche keine Wendepunkte/Drehpunkte der Nutzers darstellen, weshalb dies bei den Redirected Walking Algorithmen zu Problemen führen kann. Allerdings besitzt dieser Algorithmus einerseits den Vorteil, dass nur ein Teil der Graphen generiert wird, was zu weniger Speicherverbrauch führt. Außerdem können, durch die Generierung in der Online-Phase, mit dieser Methode auch nicht-statische Objekte in der virtuellen Welt wie z.B. bewegliche Wände unterstützt werden.

2.1 Algorithmus

In diesem Abschnitt geht es um den Algorithmus zur automatischen Graphextraktion, welcher in [Zank und Kunz, 2017] beschrieben wurde. Hierzu wird zuerst der Input des Algorithmus beschrieben, welcher aus einem Navigation Mesh besteht. Dieses wird anschließend in mehreren Schritten vereinfacht und in einen Skeleton Graph umgewandelt. Als letzten Schritt des Algorithmus wird die Extraktion der benötigten Daten aus dem Skeleton Graph in der Online-Phase erläutert.

2.1.1 Navigation Meshes

Für den Anfang des Algorithmus werden Daten der virtuellen Welt benötigt. Da eine virtuelle Welt insgesamt oft aus mehreren Millionen Polygonen besteht, welche oft einen beeindruckenden visuellen Detailgrad bieten können, ist eine vereinfachte Darstellung der Welt für eine Navigation innerhalb ausreichend. Vergleichbar sind hier die Straßenkarten eines Navigationssystems, welche ebenfalls die Straßeninformationen der realen Welt auf das nötigste reduzieren und dennoch damit ihre Aufgabe erfüllen können. In den allermeisten Fällen reichen viel weniger Polygone aus, um zu beschreiben, auf welcher Fläche sich ein virtueller Charakter bewegen kann. Da die Laufzeit von Wegfindungs-Algorithmen wie A* (A-Star) oder Dijkstra $\mathcal{O}(V \log V)$, mit V als Anzahl der Knoten in einem Graphen, beträgt und somit mit der Anzahl an Polygonen steigen, ist eine Vereinfachung der Umgebung für einen Online-Algorithmus dringend nötig.
Da die autonome Navigation in virtuellen Welten schon seit längerem eine Aufgabe innerhalb von Videospielen darstellt, um Agenten möglichst "intelligent" durch die Welt navigieren zu lassen, gibt es hierfür mehrere Möglichkeiten, die weitreichend unterstützt werden. Für diesen Algorithmus nutzten Zank und Kunz [2017] die Unity Game Engine, ihr Algorithmus wäre allerdings auch mit anderen Engines umsetzbar. Die Unity Game Engine nutzt für die Navigation von virtuellen Agenten die Methode der Navigation Meshes[2] (kurz NavMeshes). Ein Navigation Mesh stellt eine stark vereinfachte Darstellung der virtuellen Welt dar, welche für den virtuellen Agenten angepasst wurde, um dessen mögliche Laufwege einzugrenzen. In Abbildung 2.1 ist ein Navigation Mesh für die daneben dargestellte virtuelle Welt

[2] `https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html` (Zuletzt abgerufen: 01.06.2017)

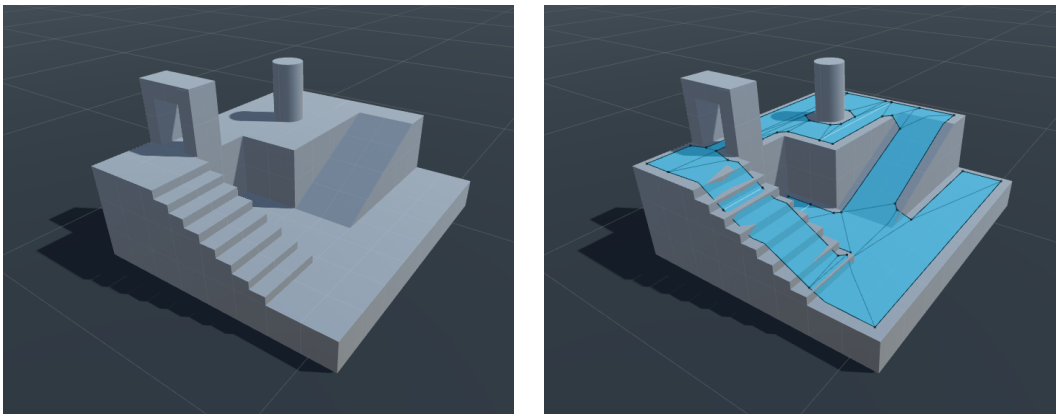


Abbildung 2.1: Eine beliebige virtuelle Welt mit einer Treppe, einer Rampe, sowie einem Durchgang und einer Säule. Im rechten Bild wird in blau das Navigation Mesh für die selbe virtuelle Welt dargestellt

zu sehen. Um dieses Navigation Mesh zu erzeugen kann der Ersteller einer virtuellen Welt im Unity-Editor mittels einer Option automatisch ein Navigation Mesh aus der erstellten Welt erzeugen lassen. Hierfür trianguliert die Engine die Welt erneut unter Berücksichtigung der Parameter des vorgesehenen Agenten. Dies führt beispielsweise bei einer Treppe zu einer Rampe sofern die Parameter erlauben, dass die Agenten diese Treppe laufen können, was im Redirected Walking vorläufig nicht beachtet wird.
In der Engine werden für unterschiedliche Agenten unterschiedliche Navigation Meshes erzeugt, basierend auf den Parametern dieser, was beispielsweise deren Größe oder Fähigkeit zu springen einschließt. Für die Anwendungszwecke des Redirected Walking wird der Agent dementsprechend eine menschliche Größe erhalten. Die Engine approximiert den Agenten beispielsweise durch ein Zylinder in der passenden Größe, um zu vermeiden, dass Agenten durch zu schmale Pfade oder zu nah an Abgründe navigieren können.
Die Unity Game Engine bietet die Navigation-Mesh-Erstellung nur zur Erstellungszeit der virtuellen Welt an (stand Juli 2017), weshalb eine Anpassung des Navigation Meshes durch bewegliche Objekte in der Online-Phase nicht ohne weiteres möglich ist. Unity selbst bietet für dieses Problem seit Version 4.3 eine Lösung damit dynamische Objekte zur Laufzeit "Löcher" in das Navigation Mesh schneiden können[3]. Hierfür wird zur Laufzeit, sobald sich ein NavMesh Obstacle weit genug bewegt hat, seine Standfläche vom ursprünglichen Navigation Mesh subtrahiert. Auf diese Art wäre es auch mit Unity potentiell möglich, mit einer Erweiterung des Algorithmus dynamische Welten zu unterstützen. Andere Game Engines bieten die Optionen Navigation Meshes zur Laufzeit auch mit dynamischen Objekten neu zu berechnen und teilen das Navigation Mesh dafür in kleinere Teile, damit nur die veränderten

[3] `https://docs.unity3d.com/Manual/class-NavMeshObstacle.html` (Zuletzt abgerufen: 01.06.2017)

Gegebenheiten neu berechnet werden müssen. Da es sich bei dem Navigation Mesh allerdings erst um die Eingabe des Algorithmus von Zank und Kunz [2017] handelt, müssten alle weiteren Schritte in der Online-Phase erneut berechnet werden, was zu einer zu hohen Rechenzeit führen könnte.

2.1.2 Optimierung des Navigation Mesh

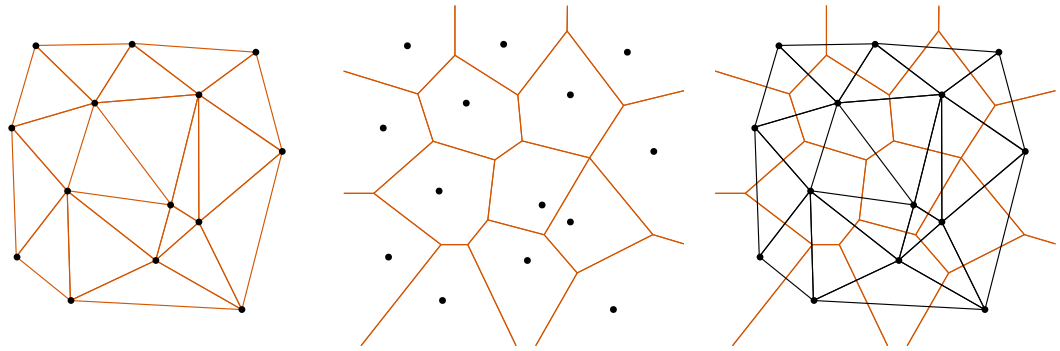
Wie zuvor angegeben wird in dieser Arbeit zu Vereinfachung von virtuellen Welten ohne Höhenunterschiede ausgegangen. Deshalb wird in diesem Abschnitt ein 2D Navigation Mesh (ohne Höhenunterschiede) betrachtet.
Das von Unity erzeugte Navigation Mesh weist laut Zank und Kunz [2017] weiterhin Artefakte auf, welche es für die Anwendung im Redirected Walking unbrauchbar machen könnten. Das Navigation Mesh markiert alle Bereiche innerhalb einer virtuellen Welt, welche begehbar sein könnten. Diese Einschätzung basiert auf der Steigung und der Größe der Fläche an der jeweiligen Stelle. Hierbei könnte allerdings ein sehr stark fragmentiertes Navigation Mesh entstehen, da auch Bereiche hinzugezählt werden, welche eventuell nicht erreichbar sind. Um dies zu beheben soll der Designer der virtuellen Welt interaktiv eine der Standflächen auswählen und alle damit verbundenen Bereiche im Navigation Mesh werden für den nächsten Schritt weiterverwendet.
Als nächster Schritt werden alle Knoten entfernt, welche innerhalb des Meshes liegen, womit vorläufig eine Art Umrandung des begehbaren Gebietes erzeugt wird. Dies führt zu einer Reduzierung der Anzahl an betrachteten Knoten und somit zu weniger Rechenzeit in weiteren Schritten. Diese Umrandung wird danach mit Hilfe eines Algorithmus zur Delaunay Triangulierung erneut trianguliert. Ziel hiervon ist, dass nun jeder Knoten, da er auf der Umrandung liegt, mit zwei Kanten der Umrandung verbunden ist. Weiterhin soll mindestens eine der Kanten jedes Dreiecks eine Kante der Umrandung sein, was für spätere Schritte wichtig wird. Die Delauney Triangulierung kann mit verschiedenen Algorithmen erzeugt werden. Als einer der einfachsten Algorithmen wäre hier der Flip Algorithmus zu nennen. Bei diesem Algorithmus wird zuerst beliebig trianguliert, sodass ein Mesh entsteht indem sich keine Kanten überschneiden und jedes Teilpolygon ein Dreieck darstellt. Jedes Dreieck besitzt in unserem Fall zwei benachbarte Dreiecke. Betrachtet man nun jedes Dreieck in 2 Kombination mit je einem der beiden benachbarten, erhält man Vierecke, welche durch eine Kante geteilt sind. Um eine Delaunay Triangulierung zu erreichen muss für jedes Dreieck die Delaunay Bedingung erfüllt sein, was bedeutet, dass für jedes Dreieck der Umschließende Kreis keinen Punkt eines anderen Dreieckes einschließen darf. Lokal kann diese Bedingung hergestellt werden indem die Kante welche das Viereck teilt *geflipt* wird. Dazu wird diese Kante entfernt und die anderen beiden Punkte des Vierecks, welche zuvor nicht verbunden waren, mit einer neuen Kante verbunden. Da durch diesen Flip die Delaunay Bedingung für andere Dreiecke wieder zerstört werden kann, benötigt dieser Algorithmus im Worst Case $\mathcal{O}(n^2)$ Schritte. Durch andere Algorithmen kann eine Delaunay Triangulierung auch in $\mathcal{O}(n \log n)$ Schritten erreicht werden.

Da das Navigation Mesh für Agenten erzeugt wurde, ist es relativ detailliert und würde beispielsweise Ausbuchtungen in einer Wand eines Flures mit einbeziehen. Da für die Anwendung im Redirected Walking diese Einbuchtung zumeist unnötig für die normalen Laufwege sind, sollten solche Artefakte ebenfalls entfernt werden um spätere Rechenzeiten zu verkürzen. Hierzu schlagen Zank und Kunz [2017] vor, zuerst alle Knoten des Navigation Meshes, welche näher als $0.3m$ zusammen liegen, zu entfernen. Ebenfalls soll jeder Knoten entfernt werden, dessen zwei Kanten der Umrandung ein größeres Skalarprodukt als 0.95 aufweisen, was bedeutet, dass alle Knoten entfernt werden, welche aus der Umrandung herausragen und deren zugehöriger Winkel in ihrem Dreieck kleiner als $arccos(0.95) \approx 18°$ ist. Diese Eigenschaft sollte nacheinander für alle Knoten geprüft werden und zuerst der mit dem größten Skalarprodukt (kleinster Winkel) entfernt werden.
Werden Knoten aus größeren Distanzen zusammengefasst oder Knoten mit einem zu großen Winkel entfernt, könnten wertvolle Informationen der Umgebung verloren gehen. Werden die Werte allerdings zu niedrig gewählt verbleiben eventuell zu viele Artefakte in dem Navigation Mesh. Deshalb sind die hier angegebenen Wert experimentell von Zank und Kunz [2017] ermittelt und sollen einen Kompromiss zwischen dem Entfernen von wichtigen Strukturen und unnötigen Details darstellen.

2.1.3 Der Skeleton Graph

In den vorherigen Schritten wurde nun aus einem generierten Navigation Mesh ein zusammenhängender Plan des begehbaren Bereiches der virtuellen Welt erstellt. Dieser Plan weist nach den Optimierungen im vorherigen Schritt weniger Artefakte auf und besitzt keine innenliegenden Knoten mehr. Weiterhin stellt die Art der Triangulierung eine Delaunay Triangulierung dar. Ziel der nächsten Schritte ist es aus dem Plan der begehbaren Flächen der virtuellen Welt einen Skeleton Graph zu erstellen, welcher die Anforderungen aus Abschnitt 2 erfüllt.

Voronoi-Diagramme Wie im Abschnitt 2 beschrieben, ist ein Skeleton Graph eine Repräsentation der virtuellen Welt, welche diese als Wege/Pfade darstellt. Der Skeleton Graph sollte in Gängen gerade in der Mitte verlaufen und in offenen Flächen in eine möglichst direkte Verbindung aller Zugänge darstellen, da dies am ehesten die Bewegungspfade eines Nutzers approximiert [Zank und Kunz, 2017]. Um eine gerade Linie zwischen den Eckpunkten eines Ganges zu erzeugen nutzen Zank und Kunz [2017] Voronoi-Diagramme. Voronoi-Diagramme stellen eine räumliche Teilung einer Fläche dar, bei der jedem Punkt auf der Fläche, der Bereich auf der Fläche zugeteilt wird der näher an diesem Punkt als an allen anderen Punkten liegt. Falls mehr Details zu diesem Thema gewünscht sein sollte sei hier auf [Aurenhammer, 1991] verwiesen. Zwischen einer Delaunay Triangulierung und einem Voronoi-Diagramm besteht eine Dualität, da in einer Delaunay Triangulierung zwei Knoten genau dann verbunden sind, wenn ihre Voronoi Zellen aneinander grenzen. Eine Beispiel-Delaunay-Triangulierung und die Dualität zwischen den beiden Diagrammtypen ist in Abbildung 2.2 zu sehen. Hierbei ist in (a) eine Delaunay Triangulierung



(a) Delaunay Triangulierung einer Fläche

(b) Einteilung einer Fläche durch ein Voronoi-Diagramm basierend auf den gegeben Punkten

(c) Dualität zwischen Voronoi Diagrammen (orange) und Delaunay Triangulierungen (schwarz)

Abbildung 2.2: Darstellung des Zusammenhanges zwischen einem Voronoi-Diagramm und einer Delaunay Triangulierung

Grafiken von `https://de.wikipedia.org/wiki/Delaunay-Triangulierung` (Zuletzt abgerufen: 06.07.2017)

zu sehen. In (b) ist für die gleiche Datenmenge ein Voronoi Diagramm abgebildet. In (c) ist zu erkennen, dass die Voronoi Diagramme die gewollte Trennung zwischen den Punkten und somit eine Linie zwischen Eckpunkten eines Ganges erzeugen. Das gewollte Voronoi-Diagramm lässt sich somit durch die Dualität (vergleiche Abbildung 2.2(c)) zwischen diesem und der Delaunay Triangulierung aus dem vorherigen Schritt errechnen. Hierzu kann auf jeder Kante der Delaunay Triangulierung eine Mittelsenkrechte eingezeichnet und diese miteinander geschnitten werden.
Bei der Generierung des Voronoi Diagrammes auf diese Art kann es passieren, dass das generierte Diagramm Kanten außerhalb der begehbaren Fläche besitzt. Dies ist in Abbildung 2.2(c) zu sehen. Hierbei kennzeichnen die schwarzen Linien die Delaunay Triangulierung aus dem vorherigen Schritt und somit den begehbaren Bereich, aus dem mehrere Spitzen des Voronoi Diagramms herausragen. Da eine Anforderung an das Skeleton Mesh ist, dass es genau den und nur den begehbaren Bereich abbildet, müssen diese Artefakte entfernt werden. Der Effekt tritt auf, wenn ein Dreieck der Delaunay Triangulierung nur eine Außenkante besitzt und diese deutlich länger als die anderen beiden Seiten des Dreiecks ist. Um dies zu beheben werden vor der Generierung des Voronoi Diagramms die Dreiecke einzeln auf diese Eigenschaft geprüft und potentiell in zwei Teile gespalten indem ein zusätzlicher Punkt in der Mitte der langen Kante eingefügt wird. Hierbei ist zu beachten, dass für die Überprüfung der Eigenschaft "deutlich länger" definiert werden muss und diese Definition die resultierende Menge an Dreiecken und somit die Performance negativ beeinflussen wird. Nach dem Aufsplitten von Dreiecken muss erneut eine Delaunay Triangulierung auf der resultierenden Knotenmenge durchgeführt werden um daraus anschließend das gewünschte Voronoi-Diagramm zu erzeugen.

Vereinfachung In Abbildung 2.2(b) ist ein Voronoi Diagramm zu sehen, welches Artefakte aufweist die für den Einsatz im Skeleton Graph nicht geeignet wären. Neben den in Abschnitt 2.1.3 aufgezeigten Linien außerhalb des begehbaren Bereiches wird der Bereich den das Diagramm umfasst auch außerhalb der begehbaren Fläche in Voronoi-Zellen eingeteilt. Diese Artefakte sind in Abbildung 2.2(b) als die Linien bis zum Bildrand zu sehen. Diese Linien können entfernt werden, indem jede Kante entfernt wird, welche zu keinem Kreis gehört. Das Ergebnis dieses Schrittes ist, dass nun durch jeden Gang der virtuellen Welt in der Mitte eine Zick-Zack Linie verläuft. Da die in Absatz 2 beschrieben Algorithmen eine Drehung des Nutzers bei jedem Knick erwarten, muss eine Zick-Zack Linie in einem geraden Gang erkannt und entfernt werden. Hierzu werden alle Knoten getestet ob sie genau zwei Nachbarn haben, was bedeutet, dass sie entweder eine Abbiegung im Gang oder eine Zick-Zack Linie darstellen. Da Abbiegungen im Gang nicht entfernt werden dürfen, können die so gefundenen Knoten nicht einfach entfernt werden. Da die Winkel des Zick-Zack Musters größer sein können als die einer Abbiegung müssen diese zwei Typen anhand einer anderen Möglichkeit separiert werden. Zank und Kunz [2017] nutzen hierzu Bezier-Kurven, um die Zick-Zack Linien zu glätten. Hierbei wird für jeden Punkt, der genau zwei Nachbarn besitzt, eine Bezierkurve berechnet, welche in beide Richtungen auf zwei Nachbarn (insgesamt 4) und sich selbst beruht. Wenn einer der Nachbarpunkte, auf denen die Bezierkurve basiert, einen Endpunkt oder eine Kreuzung darstellt, also keine weiteren oder mehr als einen Nachbarn in die Richtung aufweist, wird dieser Nachbar als letzter Stützpunkt auf der einen Seite gewählt. Durch dieses Vorgehen werden Endknoten und Abbiegungen im Graph nicht berücksichtigt, da sie mehr oder weniger als zwei Nachbar besitzen, sodass beispielsweise eine Sackgasse nicht entfernt und eine Abbiegung nicht verschoben wird. Die Bezierkurven werden anschließend genutzt, um die Punkte jeweils auf den Punkt auf der Kurve abzubilden welcher ihnen am nächsten liegt. Dies führt bei einer Zick-Zack Linie am Ende zu einer starken Glättung, weshalb anschließend Knoten der Zick-Zack Linie deutlich kleinere Winkel als Abbiegungen aufweisen und somit über diese Eigenschaft separiert und entfernt werden können. Als letzten Vereinfachungsschritt werden alle Knoten fusioniert, welche näher als beispielsweise $1m$ aneinander liegen um letzte Artefakte zu entfernen.
Hierbei sei angemerkt, dass, wenn der Skeleton Graph am Ende für MPCRed genutzt werden soll, ein weiterer Schritt erfolgen muss, welcher Abbiegungen/Ecken durch Kurven ersetzt [Nescher et al., 2014; Zank und Kunz, 2017].
Das Resultat dieses Abschnittes ist ein fertiger Skeleton Graph welcher in Gängen in der Mitte verläuft, auf freien Flächen eine Verbindung zwischen allen Ausgängen enthält und keine Wände schneidet oder begehbare Bereiche verlässt. Da Algorithmen wie FORCE und MPCRed am Ende mit Punkten beziehungsweise Teilstücken dieses Graphen arbeiten, soll im nächsten Kapitel geklärt werden, welche Punkte des Graphen während der Online-Phase an die Algorithmen weitergegeben werden sollten, damit diese optimal arbeiten. Dabei wird schlussendlich auch die letzte Anforderung an den Algorithmus erfüllt, indem die vom Designer eingefügten Ziele mit eingearbeitet werden.

2.2 Zielbestimmung

In einer virtuellen Welt existieren für den Nutzer zwei Typen von Punkten zu denen er sich bewegen könnte:

1. **Anwendungsziele** sind Ziele, die die Anwendung für den Benutzer bereit hält. Läuft der Nutzer beispielsweise durch ein Piraten-Szenario, wäre ein mögliches Anwendungsziel eine Schatzkiste oder eine Person, mit der gesprochen werden kann. Diese Anwendungsziele sind die Ziele, die ein Designer während der Erstellung der virtuellen Welt markiert und welche für die Nutzung von Redirected Walking wichtig wären, da diese die Chance erhöhen, dass der Nutzer in ihre Richtung läuft.

2. **Wegpunkte** sind Ziele die durch die Landschaft induziert werden. Wenn ein Nutzer sich in einem virtuellen geraden Gang befindet wird er höchstwahrscheinlich den Gang weiter entlang anstatt gegen einer der Seitenwände zu laufen.

Da die Anwendungsziele durch einen Designer markiert wurden und eine Liste dieser zur Laufzeit zur Verfügung steht, werden diese später weiter verwendet.
Die Wegpunkte stellen die Punkte in der Landschaft dar welche der Nutzer ansteuern könnte. Die Landschaft wiederum liegt in dieser Phase als Skeleton Graph (Abschnitt 2.1.3) vor. Da der Algorithmus nicht vorhersehen kann ob ein Benutzer die virtuelle Welt bereits kennt und welche Ziele dieser längerfristig ansteuert, wird hier angenommen, dass der Nutzer die virtuelle Welt zum ersten Mal erkundet und dementsprechend nur Ziele ansteuert, welche in seinem Sichtfeld liegen. Die Größe und Position des Sichtfeldes eines Nutzers ist allerdings von der verwendeten Hardware abhängig und verändert sich stark durch schnelle Kopfbewegungen des Nutzers. Um diese Punkte zu kompensieren, wird ein 360° Sichtfeld des Nutzers angenommen, da somit sicher gestellt ist, dass dieser sich beliebig schnell auf der Stelle drehen könnte und dies Hardware Unabhängigkeit in diesem Punkt garantiert.
Um ein 360° Sichtfeld zu errechnen, wird der begehbare Bereich der virtuellen Welt genutzt, welcher schon in Abschnitt 2.1.2 beschrieben wurde. Der einzige Punkt, von dem initial angenommen werden kann, dass er uneingeschränkt sichtbar ist, ist der Standpunkt des Nutzers. Da das Navigation Mesh eine Triangulierung darstellt, befindet sich der Nutzer in einem Dreieck von dem ebenso durch die gegebene geometrische Konvexität angenommen werden kann, dass alle Eckpunkte sichtbar sind. Ausgehend von diesen Punkten soll im Folgenden ein immer größeres Polygon aufgestellt werden, dessen Eckpunkte alle von Standpunkt des Nutzers aus sichtbar sind. Diese Eigenschaft bedeutet, dass das resultierende Polygon eine Sternform besitzt. Für den ersten Lauf in der Online-Phase muss das Dreieck, in welchem sich der Nutzer aufhält, durch Testen aller Dreiecke bestimmt werden. In den nachfolgenden Läufen kann davon ausgegangen werden, dass der Nutzer sich, aufgrund der Größe der Dreiecke im Verhältnis zur Laufgeschwindigkeit, in einem der angrenzenden Dreiecke befindet. Dies verkürzt in jedem erneuten Berechnungsschritt die

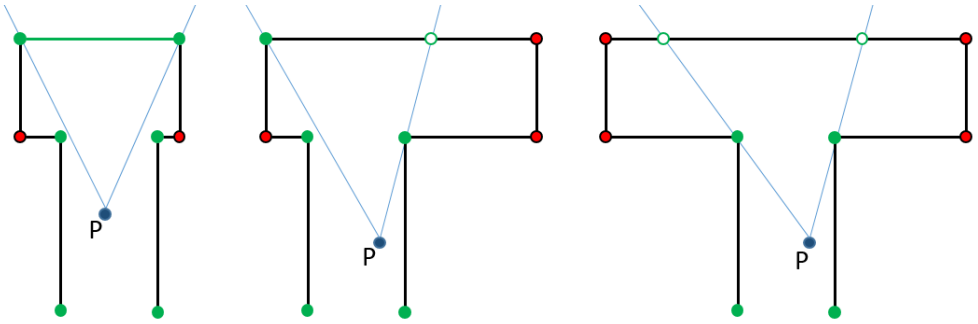
Suche nach dem Dreieck in dem sich der Nutzer befindet stark, da der Nutzer sich somit nur in drei Dreiecken befinden kann. Entweder der Nutzer befindet sich im selben Dreieck wie bei der letzten Berechnung oder in einem der *zwei* angrenzenden Dreiecke. Hierbei ist anzumerken, dass nur zwei Dreiecke angrenzen, da durch die Triangulierung in Abschnitt 2.1.2 sichergestellt wird, dass eine Kante jedes Dreiecks eine Außenkante darstellt.
Für die weitere Generierung des sternförmigen Sichtfeldes werden lediglich die Außenkanten des begehbaren Bereiches betrachtet, da die inneren Kanten keine Verdeckung erzeugen können. Um nun das Polygon zu errechnen, welches das Sichtfeld repräsentiert, muss für jeden Knoten, der auf einer der Außenkanten liegt, getestet werden, ob dieser sichtbar ist. Die geforderte Sternform weist die Eigenschaft auf, dass der Winkel von Punkt zu Punkt (im Kreis betrachtet) immer echt größer bzw. kleiner wird. Da in den nächsten Schritten für zwei Punkte bestimmt werden muss, in welcher Anordnung sie zueinander stehen, werden nun alle betrachteten Punkte vor weiteren Operationen dem Winkel nach sortieren und die Anordnung der Punkte somit durch eine Reihenfolge im Speicher auszudrücken. Auf diese Art und Weise müssen nicht für jede Vergleichsoperation beide Winkel erneut berechnet oder diese gespeichert werden. Anschließend soll für jeden dieser Knoten getestet werden, ob eine der Außenkanten die Sicht auf diesen verdeckt. Damit dies passieren kann, müssen drei Bedingungen erfüllt sein:

1. Die Kante kann den Knoten nur dann verdecken wenn einer ihrer Stützknoten auf der einen und der andere auf der anderen Seite des zu verdeckenden Knoten liegt.

2. Die verdeckende Außenkante muss einen ihrer Knoten näher am Standpunkt des Nutzers haben als der zu verdeckende Knoten. Um dies für alle Punkte der Kante sicherzustellen wird die orthogonale Distanz der Kante zum Standpunkt des Nutzers berechnet, da dies die kleinste Distanz darstellt.

3. Die Kante muss zwischen dem Standpunkt des Nutzers und dem zu verdeckenden Knoten liegen.

Die Bedingungen sind nach der Rechenintensität geordnet, sodass es von vorne herein viele Kanten ausgeschlossen werden können, bevor getestet werden muss, ob die Außenkante tatsächlich den Knoten verdeckt.
Da allerdings trotz Optimierungen der Navigation Meshes entweder die Listen an Kanten und Knoten komplett durchsucht werden müssten, oder diese zuvor sortiert werden um lediglich partiell zu suchen, liegt die Rechenkomplexität hiervon bei mindestens $\mathcal{O}(n \log n)$, was dieses Vorgehen zu langsam für den Online-Einsatz machen könnte. Allerdings können die Ergebnisse dieser Rechnung in einer Offline-Phase errechnet und in der Online-Phase lediglich abgerufen werden. Hierzu können zwei Verbesserungen eingebracht werden. Als erstes ist zu bemerken, dass die Sichtbarkeit eines Knotens bidirektional ist, was bedeutet, dass ein Knoten für den Nutzer genau dann sichtbar ist, wenn der Nutzer sich in dem Sichtfeld des Knotens befindet.



(a) Knoten haben eine gemeinsame Außenkante (b) Ein Knoten sichtbar, der andere verdeckt, Kante muss geschnitten werden (c) Beide Knoten sind verdeckt, Kante muss geschnitten werden

Abbildung 2.3: Die drei möglichen Fälle für die Verbindung zweier sichtbarer Knoten. Der Nutzer steht hier an Punkt P, die grün markierten Knoten sind sichtbar für den Nutzer und die rot markierten nicht sichtbar. Entnommen aus [Zank und Kunz, 2017]

Hierzu kann die gleiche Berechnung wie oben für jeden Knoten der Außenkanten durchgeführt und die Sichtfeldpolygone gespeichert werden. In der Online-Phase müsste somit nur noch getestet werden in welchen der Sichtfeldpolygone der Knoten sich der Nutzer befindet. Da die Rechenzeit damit immer noch stark von der Anzahl an Knoten in den Außenkanten abhängt, kann ein weiterer Schritt in der Offline-Phase durchgeführt werden. Da sich der Nutzer immer innerhalb von einem der Dreiecke des begehbaren Bereiches aufhält, kann für jedes Dreieck im voraus getestet werden von welchen Knoten es potentiell sichtbar ist. Wenn das Dreieck von einem Knoten sichtbar ist, bedeutet dies, dass der Knoten umgekehrt von mindestens einem Punkt im Dreieck sichtbar ist. Somit kann für jedes Dreieck eine Liste von Knoten gespeichert werden welche potentiell sichtbar sein könnten, womit der vorherige Schritt stark beschleunigt werden kann.
Da die bisherigen Berechnungen lediglich eine Punktwolke von sichtbaren Punkten erzeugt hat, müssen diese Punkte nun zu dem für die genauen Sichtbarkeitstests benötigten Polygon zusammengefügt werden. Hierbei ist zwischen drei Fällen zu unterscheiden, welche in Abbildung 2.3 zu sehen sind. Da die Knoten aus den vorherigen Schritten dem Winkel nach sortiert sind, sollten aufeinander folgende Knoten verbunden werden. Im ersten Fall wird gezeigt, dass die betrachteten Knoten sich eine gemeinsame Außenkante teilen, weshalb sie für das Polygon erneut verbunden werden. Die Fälle zwei und drei zeigen, dass die Knoten zuvor im Navigation Mesh nicht durch eine Außenkante verbunden waren. Das führt dazu, dass hier per Ray-Casting die Punkte auf weitere Außenkanten abgebildet werden müssen um diese dann in das Sichtfeldpolygon einzubeziehen.
Da nun für jede Position des Nutzers zur Laufzeit ein 360°-Sichtfeldpolygon errechnet werden kann, soll dies für Algorithmen wie FORCE und MPCRed verwendet

werden. Hierzu werden alle verfügbaren Informationen kombiniert. Da angenommen wird, dass der Nutzer sich nur zu Zielen innerhalb seines Sichtfeldes bewegt, kann durch eine Schnittoperation des Sichtfeldpolygons mit den anderen Informationen bestimmt werden, welche Ziele zu dem jeweiligen Zeitpunkt relevant sind. Zuerst können die vom Designer in der virtuellen Welt markierten Ziele getestet werden ob sie innerhalb des Sichtfeldes liegen. Ist dies der Fall, kann für den Redirected-Walking-Algorithmus ein lokaler Graph generiert werden, welcher für jeden gefunden Punkt eine Kante vom Standpunkt des Nutzers zu dem jeweiligen Punkt zieht. Da jeder der Punkte, sobald er im Sichtfeld des Nutzers liegt, nicht durch andere Objekte/Wände verdeckt wird, führt diese gerade Kante niemals aus dem begehbaren Bereich hinaus. Für alle weiteren potentiellen Ziele, die der Nutzer sehen kann, wird der zuvor generierte Skeleton Graph mit dem Sichtfeldpolygon geschnitten. Alle Teilstücke des Skeleton Graphs innerhalb des Sichtfeldpolygons können somit in den lokalen Graphen übernommen werden. Hierbei sollte lediglich darauf geachtet werden, dass der Standpunkt des Nutzers die Wurzel des lokalen Graphen darstellen sollte, sodass der Skeleton Graph eventuell leicht verformt werden müsste um eine seiner Kanten durch den Standpunkt des Nutzers laufen zu lassen um diesen anschließend als Wurzel zu nutzen.
Für die hier beschriebene Generierung der Sichtfeldpolygone ist zu beachteten, dass diese Einschränkungen bietet. Diese Methoden funktionieren nur gut in eingeschränkten virtuellen Welten und nicht auf offenem Gelände mit einer räumlichen Trennung. Ein Beispiel, in dem die hier beschriebenen Methoden versagen würden, wäre eine offene Fläche mit einer Schlucht in der Mitte. Der Nutzer könnte zu jeder Zeit über die Schlucht schauen und würde dort Ziele sehen können, welche der Algorithmus nicht betrachtet, da dieser, die Schlucht standardmäßig als Wand ansieht und annimmt, dass sie potentielle Ziele verdeckt.

Nicht statische Umgebungen Der hier beschriebene Algorithmus geht davon aus, dass die verwendete virtuelle Welt statisch ist, also keine beweglichen oder selbst bewegenden Objekte enthält. Diese Objekte können in drei Kategorien aufgeteilt werden. Die kleinsten Objekte, wie beispielsweise ein Fußball, sollten den begehbaren Bereich nicht einschränken, könnten allerdings ein weiteres Ziel darstellen. Diese Objekte sind für den hier angegebenen Algorithmus unwichtig, da nur der begehbare Bereich betrachtet wird. Die zweite Gruppe stellen Türen dar. Obwohl Türen, solange sie geschlossen sind, die Sicht des Nutzers verdecken, sollten sie den Skeleton Graph nicht unterbrechen, solange sie für den Benutzer zu öffnen sind. Die Türen sollten allerdings für diesen Fall zu den Anwendungszielen hinzugefügt werden. Die dritte Gruppe von Objekten stellen Objekte dar, welche signifikant größer als der Nutzer sind, sodass sie den Weg blockieren können und für den Nutzer nicht beweglich sind. Zank und Kunz [2017] merken hierzu an, dass diese Art nicht häufig vorkommt, allerdings den begehbaren Bereich und die Sichtfeldpolygone soweit verändert, dass am einfachsten mehrere Skeleton Graphen- und Sichtfeldpolygon-Berechnungen durchgeführt werden sollten um diese bei Bedarf im Online-Teil tauschen zu können.

2.3 Evaluation

Der hier beschriebene Algorithmus benötigt neben der rechenintensiven Offline-Phase und dem Speicher für die potentiell sichtbaren Knoten und dem Skeleton Graph eine relativ hohe Rechenzeit in der Online-Phase. Diese Rechenzeit würde die Zeit, die für die Berechnung eines Frames zur Verfügung steht, überschreiten, vor allem wenn die virtuelle Welt groß/verwinkelt ist. Zank und Kunz [2017] testeten die Online-Rechenzeit auf einem XMG U506 Laptop (Intel i7-6700) für eine relativ kleine virtuelle Welt mit einem begehbaren Bereich aus 351 Knoten und einem Skeleton Graph bestehend aus 211 Kanten. Das Ergebnis dieses Tests ist, dass die Rechenzeit für das Erzeugen des Sichtfeldpolygons $40\,ms$ nicht übersteigt. Dies lässt auch für größere virtuelle Welten genug Zeit, da diese Berechnungen ähnlich wie die Redirected Walking Algorithmen nicht für jeden Frame ausgewertet werden müssen. Da das Sichtfeld des Nutzers für gewöhnlich so groß ist, dass dieser es nicht in sehr kurzer Zeit verlassen kann oder so signifikant ändert und durch die Verwendung eines 360° Sichtfeldpolygons dieses sich auch nicht durch schnelle Drehungen ändert, nehmen Zank und Kunz [2017] an, dass hier die von Nescher et al. [2014] veranschlagten 2.5 Sekunden für einen Planungszyklus ausreichen sollten. Zank und Kunz [2017] geben keine weiteren Zahlen ihrer Versuche an, sodass die Rechenzeit für die Offline-Phase und die anderen Teile der Online-Phase unklar bleibt.

3 Fazit

Nach den anfänglichen Arbeiten von Razzaque et al. [2001] im Bereich des Redirected Walking, erbrachte das Team 2001 den Beweis, dass ihre Idee Potential besitzt, eine natürliche Fortbewegungsweise für virtuelle Realitäten zu bieten. Die darauf aufbauenden Arbeiten beschäftigen sich unter anderem mit der Entwicklung und Verbesserung der verwendeten Algorithmen. Da die entwickelten Algorithmen zeigten, dass diese signifikant besser arbeiten, wenn die Laufstrecke des Nutzers zuvor bekannt ist, ist eines der Ziele weiterer Forschung die Vorhersage von Nutzerverhalten. Vom Standpunkt dieser Arbeit lässt sich somit beschreiben, dass die Techniken, welche ursprünglich von Razzaque et al. stammten und beispielsweise von Zmuda et al. [2013] und Nescher et al. [2014] verbessert wurden, grundsätzlich einsatzbereit sind, allerdings noch viel Arbeit in die einfache und generelle Verwendbarkeit dieser investiert werden muss. Der in dieser Arbeit beschriebe Algorithmus zur Extraktion eines Skeleton Graphs aus einer virtuellen Welt für Redirected Walking, welcher benutzerfreundlich genug ist, damit er von Designern ohne viel Vorwissen genutzt werden kann, bietet eine Verbesserung in der Anwendbarkeit der Techniken für eine breitere Masse. Der Algorithmus besitzt allerdings in der jetzigen Form Einschränkungen im effizienten Abbilden von nicht-statischen Umgebungen.

Literaturverzeichnis

Arechavaleta, Gustavo, J.-P. Laumond, H. Hicheur und A. Berthoz (2008). *An optimality principle governing human walking*. IEEE Transactions on Robotics, 24(1):5–14.

Aurenhammer, Franz (1991). *Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure*. ACM Computing Surveys (CSUR), 23(3):345–405.

Azmandian, Mahdi, T. Grechkin, M. Bolas und E. Suma (2016). *Automated Path Prediction for Redirected Walking Using Navigation Meshes*. In: *3D User Interfaces (3DUI), 2016 IEEE Symposium on*, S. 63–66. IEEE.

Bruce, Allison und G. Gordon (2004). *Better motion prediction for people-tracking*. In: *Proc. of the Int. Conf. on Robotics & Automation (ICRA), Barcelona, Spain*.

Gandrud, Jonathan und V. Interrante (2016). *Predicting Destination using Head Orientation and Gaze Direction during Locomotion in VR*. In: *Proceedings of the ACM Symposium on Applied Perception*, S. 31–38. ACM.

Hicheur, Halim, Q.-C. Pham, G. Arechavaleta, J.-P. Laumond und A. Berthoz (2007). *The formation of trajectories during goal-oriented locomotion in humans. I. A stereotyped behaviour*. European Journal of Neuroscience, 26(8):2376–2390.

Hodgson, Eric und E. Bachmann (2013). *Comparing four approaches to generalized redirected walking: Simulation and live user data*. IEEE Transactions on Visualization and Computer Graphics, 19(4):634–643.

Interrante, Victoria, B. Ries und L. Anderson (2007). *Seven league boots: A new metaphor for augmented locomotion through moderately large scale immersive virtual environments*. In: *3D User Interfaces, 2007. 3DUI'07. IEEE Symposium on*, S. 167–170. IEEE.

Nescher, Thomas, Y.-Y. Huang und A. Kunz (2014). *Planning Redirection Techniques for Optimal Free Walking Experience Using Model Predictive Control*. In: *3D User Interfaces (3DUI), 2014 IEEE Symposium on*, S. 111–118. IEEE.

Razzaque, Sharif, Z. Kohn und M. C. Whitton (2001). *Redirected Walking*. In: *Proceedings of EUROGRAPHICS*, Bd. 9, S. 105–106. Citeseer.

Steinicke, Frank, G. Bruder, J. Jerald, H. Frenz und M. Lappe (2010). *Estimation of detection thresholds for redirected walking techniques*. IEEE Transactions on Visualization and Computer Graphics, 16(1):17–27.

Williams, Betsy, G. Narasimham, T. P. McNamara, T. H. Carr, J. J. Rieser und B. Bodenheimer (2006). *Updating orientation in large virtual environments using scaled translational gain*. In: *Proceedings of the 3rd Symposium on Applied Perception in Graphics and Visualization*, S. 21–28. ACM.

Williams, Betsy, G. Narasimham, B. Rump, T. P. McNamara, T. H. Carr, J. Rieser und B. Bodenheimer (2007). *Exploring large virtual environments with an HMD when physical space is limited*. In: *Proceedings of the 4th symposium on Applied perception in graphics and visualization*, S. 41–48. ACM.

Yen, Hsiao Chieh, H. P. Huang und S. Y. Chung (2008). *Goal-directed pedestrian model for long-term motion prediction with application to robot motion planning*. In: *Advanced robotics and Its Social Impacts, 2008. ARSO 2008. IEEE Workshop on*, S. 1–6. IEEE.

Zank, Markus und A. Kunz (2015). *Using Locomotion Models for Estimating Walking Targets in Immersive Virtual Environments*. In: *Cyberworlds (CW), 2015 International Conference on*, S. 229–236. IEEE.

Zank, Markus und A. Kunz (2017). *Optimized Graph Extraction and Locomotion Prediction for Redirected Walking*. In: *3D User Interfaces (3DUI), 2017 IEEE Symposium on*, S. 120–129. IEEE.

Zmuda, Michael A, J. L. Wonser, E. R. Bachmann und E. Hodgson (2013). *Optimizing Constrained-Environment Redirected Walking Instructions Using Search Techniques*. IEEE transactions on visualization and computer graphics, 19(11):1872–1884.