Fakultät für Bauingenieur- und Vermessungswesen
Lehrstuhl für Computergestützte Modellierung und Simulation
Prof. Dr.-Ing. André Borrmann

Visualisierung von Fußgängersimulationsdaten auf Basis einer 3D-Game-Engine

Daniel Büchele

Masterarbeit

Author:	Daniel Büchele
Matrikelnummer:	10022975 (LMU)
Betreuer:	Prof. Dr. André Borrmann
	Oliver Handel, M.Sc. M.Phil., Dipl.-Inf. (FH) Peter Kielar, M.Sc.
Anmeldedatum:	12. März 2014
Abgabedatum:	29. Juli 2014

Beteiligte Organisationen

Fakultät für Bauingenieur- und Vermessungswesen
Lehrstuhl für Computergestützte Modellierung und Simulation
Technische Universität München
Arcisstraße 21
D-80333 München

Ludwig-Maximilians-Universität München
Institut für Informatik
LFE Medieninformatik
Amalienstrae 17
Vordergebäude, 5. Stock, Zimmer 507
D-80333 München

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe. Diese Arbeit wurde zuvor nicht bei einer anderen akademischen Institution eingereicht oder anderweitig veröffentlicht.

München, 29. Juli 2014

Daniel Büchele

Daniel Büchele
Clermontstr. 10
82131 Gauting
daniel@buechele.cc

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Vorhersage von menschlichem Verhalten

"I predict a riot!" nennt Hannah Fry ihren Vortrag auf der *re:publica* 2014. Sie forschte am University College London zu den Aufständen von 2011 in England. [DFWB13] Zunächst begannen am 6. August 2011 die Proteste als friedliche Demonstration zur Aufklärung eines Todes bei einem Polizeieinsatz. Doch bald breiteten sich die Proteste in großen Teilen Londons und anderen Städten Englands aus, und demonstrierten gegen soziale Ungerechtigkeit im Land. Bei diesen Protesten kam es zu vielen gewalttätigen Ausschreitungen, Plünderungen und Verhaftungen. Fry versuchte mittels der Daten, von den damit einhergehenden Verhaftungen, zu berechnen, an welchen Orten Ausschreitungen als nächstes Auftreten werden. Als Datenbasis nutzte sie die Orte und Zeiten an denen Demonstranten verhaftet wurden sowie den Wohnort der Verhafteten. Damit gelang es ihr Vorhersagen zu treffen, wo als nächstes Straftaten passieren könnten. Diese Erkenntnisse könnten in Zukunft der Polizei helfen, solche Situationen mit weniger Einsatzkräften effizienter zu kontrollieren. [Fry14]

Ursprünglich kommt Frys Forschung aus der Fluiddynamik und später aus der Komplexitätstheorie. Dass sie nun das menschliche Verhalten untersucht scheint fernab von ihrem ursprünglichen Forschungsgebiet. Bei genauerer Betrachtung sind diese Themen aber eng verwandt. [Fry] Ein komplexes System beschreibt das zusammen wirken vieler einzelner Teile, die sich gegenseitig beeinflussen. Dadurch können Phänomene wie Selbstorganisation und -regulation entstehen. Solche komplexen Systeme können vielfältig sein, von der Bewegung von Teilchen, über das Wetter und dem menschlichen Gehirn bis hin zum Menschenmassen. [RR02]

Jetzt ist auch ersichtlich, wie Frys Weg zur Fußgängerforschung führte. Zwar waren die meisten Demonstranten zu Fuß unterwegs, in ihrer Arbeit spielt dies aber nur eine untergeordnete Rolle. Trotzdem ist ihre Forschung eng mit dem Thema dieser Arbeit verwandt – worum es im Kern geht ist die Voraussage von menschlichem Verhalten. [Fry] Denn solche Vorhersagen werden erst dann möglich, wenn es möglich ist das menschliche Verhalten soweit zu verstehen, dass es in logische Modelle übersetzt werden kann, die dann für Simulationen genutzt werden. [GKL08]

Dieses Vorgehen wird auch bei der Verkehrsforschung angewandt. Ein großer Schwerpunkt liegt dabei auf dem Kraftfahrzeugverkehr und im speziellen der Stauforschung. Zur Verbesse-

rung der Verkehrssituation und Planung neuer Straßen werden Ursachen von Staus untersucht und der Verkehrsfluss gemessen. Doch auch abseits der Straßen gibt es interessante Gebiete der Verkehrsforschung. Das Verhalten von Fußgängern scheint wesentlich komplexer zu sein, da diese sich nicht wie Autos ausschließlich entlang der vorgegebenen Straßen bewegen können und wesentlich dynamischer bei der Wegewahl sind. [GKL08] Erste Analysen des Fußgängerfluss wurden bereits 1958 von Hankin und Wright [HW58] gemacht, die den Personenfluss in den unterirdischen Gängen der Londoner U-Bahn untersuchten. Schon damals wurden speziell Treppen und Ecken untersucht um das Wissen bei zukünftigen Bauprojekten zu nutzen.

Spätestens mit dem Unglück bei der Loveparade 2010 in Duisburg ist die Aufmerksamkeit der Verkehrsforschung im Bezug auf Fußgängerströme gewachsen. Als Folge einer Massenpanik starben dort 21 Menschen und 541 weitere wurden verletzt. [Han12] Wodurch solche Paniken entstehen und wie die Situation, gerade beim zusammentreffen großer Menschenmassen, sicherer gestaltet werden kann sind aktuelle Forschungsgebiete mit einem praktischen Anwendungsbereich. [BM10]

1.2 Fußgängerforschung als interdisziplinäres Forschungsfeld

Abbildung 1.1: Verschiende Fachrichtungen tragen Erkenntnisse zur Fußgängerforschung bei

Die Analyse und Untersuchung von einzelnen Fußgängern und Fußgängerströmen sind ein hochgradig interdisziplinäres Forschungsgebiet, wie in Abbildung 1.1 schematisch dargestellt ist. Die Forschung aus den verschiedensten Fachrichtungen trägt Erkenntnisse dazu bei.

Betrachtet man die Fortbewegung des Menschen, beginnt man vermutlich bei der Biologie und untersucht den Bewegungsapperat des Menschen und seine natürlichen Voraussetzungen. [Wei92]

Wohin sich der Mensch bewegt und welchen Weg er dabei wählt sind dann Forschungsgebiet der Psychologie. Im Kontext von größeren Gruppen, und der Dynamiken die innerhalb dieser Gruppen entstehen können, gibt es auch viel Forschung in der Soziologie. [AHR+93]

Modelle aus der Physik können verwendet werden um das Verhalten der Menschenmassen zu modellieren [Kir02] und die Informatik untersucht die algorithmische Simulation dieser Modelle, beispielsweise in zellulären Automaten. Außerdem sind auch die effiziente Datenverarbeitung großer Simulationsdaten und die Visualisierung der Simulationen wichtige Forschungsgebiete. [Kne13]

Genutzt werden die gewonnenen Erkenntnisse unter anderem in der Architektur und im Bauingenieurwesen. Bei der Konstruktion neuer Gebäude und Veranstaltungsflächen werden

die gewonnenen Erkenntnisse in der Planung berücksichtigt um die Sicherheit der Besucher durch bauliche Maßnahmen zu verbessern. [Unt06]

1.3 Motivation und Ziele der Arbeit

Im Jahr 2012 habe ich in meiner Bachelorarbeit eine erste webbasierte Software zur Post-Visualisierung von Fußgängersimulationsdaten entwickelt. [Büc12] In dieser Arbeit wird eine neue Visualisierungssoftware im selben Stil entwickelt, die die Konzepte aus der Bachelorarbeit aufgreift, weiterentwickelt und neue Aspekte betrachtet.

Ziel von Software zur Post-Visualisierung ist es, nach einer erfolgten Simulation die Simulationsergebnisse darzustellen und aufzubereiten. Die meisten Simulationsverfahren erzeugen die Simulationsergebnisse in Form von rohen Datensätzen oder stellen nur einfache Visualisierungsmöglichkeiten zur Verfügung. Die Software, die in dieser Arbeit entwickelt wird, setzt den Schwerpunkt auf die Visualisierung und soll deshalb eine wesentlich anspruchsvollere Darstellung ermöglichen, als bestehende Visualisierungs-Lösungen bieten können.

Anders als bei der webbasierten Lösung aus der Bachelorarbeit, soll für die Umsetzung eine Game-Engine genutzt werden, die die grafische Performance von Computerspielen der Post-Visualisierung zugänglich macht.

Die Nutzung von Post-Visualisierungen entkoppelt die Darstellung von der eigentlichen Simulation bzw. Erzeugung der Daten. So kann modular der Art der Erzeugung der Daten verändert werden und trotzdem die selbe Post-Visualisieurngslösung genutzt werden. Andersherum können auch je nach Anforderung unterschiedliche Post-Visualisierungslösungen für die selben Datensätze verwendet werden.

Die Post-Visusalisierung von Simulationsdaten ist ein wichtiger Schritt um die bei der Simulation erzeugten Daten zugänglich zu machen und auszuwerten. Denn die Daten können nur dann effizient genutzt werden, wenn sie leicht zu verstehen und interpretieren sind. Die realitätsnahe Darstellung der Post-Visualisierung hilft dabei, dass beispielsweise Experten eine Situation betrachten, einschätzen und bewerten können, ohne dabei die grundlegenden Datensätze verstehen zu müssen. Auch für die Schulung von Einsatzkräften, die Verwendung des Bildmaterials für Pressearbeit oder die Lehre kann die Post-Visualisierung gute Dienste leisten.

Kapitel 2

Fußgängersimulation

In vielen Fällen ist es nicht möglich oder sinnvoll das Verhalten von Fußgängen in bestimmten Situationen empirisch zu beobachten und zu analysieren. Da der Fußgängerfluss bereits in die Planung neuer Gebäude einfließt, werden Analysen für Szenerien benötigt die in der Realität noch gar nicht existieren. [HFMV02] Entsprechende Testaufbauten mit echten Personen sind aufwändig und teuer umzusetzen. Hier können Simulationen eine praktische alternative bieten. Auch für Sicherheitsanalysen zu Großveranstaltungen mit mehreren tausend Teilnehmern sind empirische Versuche auf Grund der Sicherheitsrisiken und Kosten kaum zu realisieren.

Abhilfe für diese Probleme schaffen Simulationen, die das entsprechende Szenario virtuell berechnen. So kann kostengünstig und flexibel abgeschätzt werden, mit welchen Fußgängerströmen zu rechnen ist. Als Grundlage für die Simulation muss aber zunächst das Verhalten der Fußgänger untersucht werden. Daraus können dann Verhaltensmodelle abgeleitet werden (vgl. Kapitel 2.2). Mit der algorithmischen Implementierung der Verhaltensmodelle können Simulationen erzeugt werden. [Kne13]

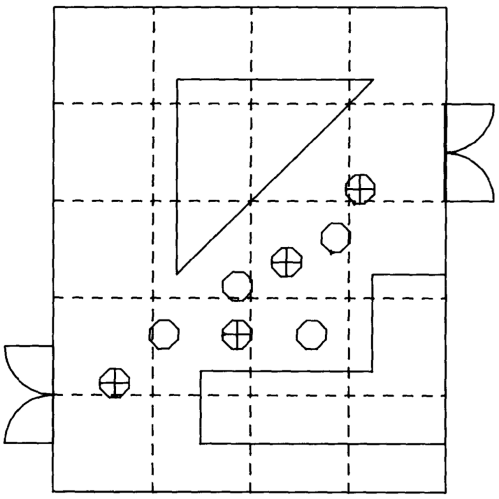


Abbildung 2.1: Fußgängersimulations-Visualisierung von Gipps und Marksjö (1985). Leere und gefüllte Kreise stellen verschiedene Typen von Fußgängern dar. [GM85]

Frühe Fußgänger-Simulationsvisualisierung Eine der ersten computergestützten Simulation stellten Gipps und Marksjö bereits 1985 vor. [GM85] Sie beschreiben eine mikroskopische Simulation (vgl. Kapitel 2.2.1), die verschiedene Typen von Fußgängern unterscheidet (z.B. Touristen, Personen auf dem Arbeitsweg, etc) und jeden Fußgänger als individuelles Objekt mit seinen jeweiligen Eigenschaften simuliert. Bemerkenswert für diese Zeit war, dass sie eine Visualisierung in Echtzeit implementierten. Für jeden Fußgänger werden Ziele definiert (z.B. kürzester Weg, wenige Kurven), die verschieden gewichtet in die Routenauswahl einfließen. Die Szene wird in einem Zellenmuster gegliedert auf dem die Fußgänger sich bewegen können. In jedem Simulationsschritt werden die benachbarten Zellen nach den obigen Regeln bewertet. Der Fußgänger

bewegt sich in die Zelle mit der höchsten Gewichtung. Diese Idee entspricht einem zellulären Automat, wie er in Kapitel 2.2.1 beschreiben wird. Die Darstellung erfolgte als Liniengrafik in einer Draufsicht (siehe Abbildung 2.1).

Diese erste computergestützte Simulation und Visualisierung legte den Grundstein für viele weitere Implementierungen. Heute sind solche Simulationen ein wichtiger Bestandteil der Planung von Veranstaltungen und Bauvorhaben. Sie liefern Daten über mögliche Konfliktpunkte, berechnen die Wartezeit und den Platzbedarf und werden genutzt um die Dauer von Evakuierungen vorherzusagen. [HFMV02] [Kne13]

2.1 Modellierungsebenen bei der Fußgängersimulation

Um das Verhalten von Fußgängern zu modellieren gibt es verschiedene Ansätze, die das menschliche Verhalten in unterschiedlichen Situationen beschreiben und voraussagen können. Um diese Modelle vergleichbar zu machen, gliedern verschiedene Arbeiten die Modelle in Modellierungsebenen und -schichten. Die Unterteilung erfolgt nach einer Schichtenarchitektur, wie sie auch an vielen Stellen der Informatik genutzt wird (z.B. Softwarearchitektur). Jede Schicht bietet der darüberliegenden Schicht ein Schnittstelle zu darunterliegenden Schichten, abstrahiert so Komplexität und vereinfacht den Austausch zwischen einzelnen Komponenten. Die Strukturierung der Simulationsmodelle in Schichten soll dann auf beliebige Modelle anwendbar sein. Im Folgenden werden verschiedene Schichtenmodelle vorgestellt und anschließend gegenübergestellt.

2.1.1 Schichtenmodell nach Hoogendoorn et al.

2004 unterteilte Hoogendoorn et al. die Wegewahl der Fußgänger in drei hierarchische Ebenen: die strategische, taktische und operationelle Ebene. Auf allen drei Ebenen werden Entscheidungen getroffen, die einen Einfluss auf die Bewegung des Fußgängers haben. Alle Aktionen, die von einem Fußgänger ausgeführt werden, erfüllen einen Nutzen für den Fußgänger, erzeugen aber zu gleich auch Kosten. Ziel des Fußgängers ist die Optimierung des Nutzens bei möglichst geringen Kosten. [HB04]

Die **strategische Ebene** legt die grundlegenden Rahmenbedingungen für die Bewegung fest. Hierzu gehören die Bestimmung notwendiger Aktivitäten zur Zielerreichung und Wahl des Zeitpunkts der Bewegung. Auf der strategischen Ebene wird eine Menge an Aktivitäten erzeugt, die als Eingabewert für die taktische Ebene fungiert. Die taktische Ebene ist dann zuständig diese ungeordnete Menge von Aktivitäten in eine Reihenfolge zu bringen. [Hoo01] [HB04]

Die Gründe für Fußgängerbewegungen lassen sich in drei unterschiedliche Kategorien unterteilen: alltägliche Szenarien (z.B. Arbeitsweg), Freizeit (z.B. Konzerte, Fußballspiele) und Evakuierungs- und Notfallsituationen. Entsprechend des Grunds der Bewegung wählen Fußgänger andere Strategien um ihr Ziel zu erreichen. Die Parameter für Geschwindigkeit, Zielstrebigkeit oder Gruppengroße variieren dabei stark. Oft mischen sich aber auch Fußgängerströme mit verschiedenen Motivationen. [DH03] Beispielsweise wenn sich Berufs- und Freizeitverkehr kreuzen. Um solche Szenarien zu simulieren müssen dann für einzelne Fußgänger unterschiedliche Implementierungen auf strategischer Ebene verwendet werden. [Kne13]

Die nächste Ebene wird als **taktische Ebene** bezeichnet und beschreibt die Navigation des Fußgängers zu seinem Ziel. Das Ziel besteht aus einer Reihe von Teilzielen die der Fußgänger in einer bestimmten Reihenfolge abarbeitet ("activity schedule"). Durch die Reihenfolge der Teilziele ergibt sich eine Route, für die sich der Fußgänger entscheidet. Allerdings beschreibt Hoogendoorn die Routenwahl als ein nicht deterministisches Verfahren: Bei der Auswahl einer Route, aus einer unendlich großen Menge an Alternativrouten, gibt es auch Zufallsfaktoren, die beispielsweise fehlende Informationen über die Persönlichkeit des Fußgängers simulieren sollen.

Der Wegfindungsprozess vom aktuellen Standort zum Zielpunkt wird durch externe und interne Faktoren beeinflusst. Diese Faktoren werden als Eingabewerte bezeichnet auf deren Basis dann die Routenwahl getroffen wird. [HB04]

Externe Faktoren sind Hindernisse auf dem Weg, wie beispielsweise Gebäude, aber auch Umgebungsumstände wie Stau oder die durchschnittliche Fluss-Geschwindigkeit. Die internen oder persönliche Faktoren können beispielsweise Zeitdruck, Einstellung, Wesenszüge oder körperliche Beschaffenheit sein. [HB04]

Aus der taktischen Ebene leitet sich die **operationelle Ebene** ab. Sie beschreibt die Ausführung der Bewegung. Auf dieser Ebene entscheidet sich der Fußgänger für eine geeignete Bewegungsart um sein Ziel zu erreichen. Abhängig vom Grund der Bewegung und der Wegfindung (externe und interne Faktoren) wählt jeder Fußgänger ein Gehverhalten ("walking behavior") das der Situation gerecht wird. [Kne13] [HB04]

2.1.2 Schichtenmodell nach Reynolds

Das Schichtenmodell nach Reynolds (1999) [Rey99] bezieht sich nicht auf die Simulation von Fußgängern, sondern auf die Steuerung von "non-player characters" (kurz *NPC*), also computergesteuerten Einheiten in Computerspielen. Diese bewegen sich autonom durch eine Szene, ohne dass ihre Wege dabei vorgegeben sein sollen. Außerdem sollen sie auf nicht vorhersehbare Ereignisse, wie Eingaben vom Spieler, reagieren können und entsprechend ihre Wegewahl anpassen. Damit das Verhalten der NPCs möglichst realistisch wird, werden auch hier Simulationsmechanismen eingesetzt. [JSC06] [Ber05]

Die drei Ebenen gliedern sich in "action selection" (Aktionsauswahl), "steering" (Steuerung) und "locomotion" (Lokomotion/Bewegung). [Rey99]

Bei der **Aktionsauswahl** entscheidet sich der NPC welche Aktion ausgeführt werden soll. Dazu werden Status bzw. Veränderungen der Spielwelt ausgewertet und eine mögliche Aktion ausgewählt. Durch die ausgewählte Aktion ergibt sich auch ein Zielpunkt, den der NPC erreichen möchte. [Rey99]

Wie dieses Ziel erreicht wird bestimmt die **Steuerungsschicht**. Der Pfad zum Ziel wird aus vielen kleinen Zwischenzielen zusammengesetzt und ist somit ein zusammengesetzter Pfad entlang der Zwischenziele. Solche Zwischenziele können zum Beispiel, Vermeidung von Hindernissen, Ausweichen vor Gegnern, etc. sein. Um den generierten Pfad zu verfolgen werden dann einzelne Signale erzeugt. [Rey99]

Die **Lokomotsionsschicht** bekommt Signale von der darüberliegenden Steuerungsschicht. Diese Signale können beispielsweise einfache Richtungsanweisungen sein, oder Veränderungen

der Geschwindigkeit. In der Lokomotionsschicht werden diese Anweisungen dann in konkrete Bewegungen umgesetzt. Im Bezug auf Computerspiele oder 3D-Animation kümmert sich die Schicht um die Bewegungsanimation. Dies beinhaltet beispielsweise die Animation der Beine, die Drehung oder Positionierung des 3D-Modells. [Rey99]

2.1.3 Schichtenmodell nach Blumberg

Auch Blumberg bezieht sich in seinem Schichtenmodell auf die Computeranimation und die Verhaltenssimulation von nicht-spielergesteuerten Charakteren. Er definiert einen "animated autonomous characters" dabei als "animiertes Objekt, das sich zielgerichtet und zeitabhängig verhält" [BG95] und aus Sensoren, einem Bewegungssystem sowie der dazugehörigen Geometrie besteht. Im Prinzip beschreibt die Arbeit, wie auch die Arbeit von Reynolds, die Steuerung von NPCs.

Blumberg definiert ebenfalls drei unterschiedliche Schichten. Aber anders als beiden anderen beiden Modellen, bezeichnet er in seinem Modell die Schichten als verschiedene Ebenen der Manipulation ("three levels of input" [BG95]).

Während bei Hoogendoorn die Ebenen zur qualitativen Beschreibung des Verhaltens dienen, sind bei Blumberg, ähnlich wie bei Reynolds, die Schichten eine technische Modellierung. In Blumbergs Modell können auf jeder Ebene Anweisungen an das Simulationsobjekt übergeben werden. Je nachdem um welche Kategorie von Information es sich handelt wird diese auf der entsprechenden Schicht an das Objekt übergeben und verarbeitet. Die Schichten kapseln dabei ihre Aufgaben gegenüber darüber- und darunterliegenden Schichten ab und stellen sie diesen, gemäß des klassischen Schichtenmodells der Informatik, zur Verfügung.

Die **Motivationsschicht** beschreibt die Auswahl der auszuführenden Aufgabe anhand eines Motivationsmodells. Die Auswahl der auszuführenden Aktionen erfolgt durch die Festlegung verschiedener Motivationslevel für jede mögliche Aufgabe. Die Aufgabe mit dem höchsten Motivationslevel wird dann ausgewählt und ausgeführt. Bei der Bestimmung des Motivationslevel wird zwischen inneren Anreizen (innere Zustände, wie z.B. Hunger) und externer Motivation (Wahrnehmungen aus der Umwelt) unterschieden. Wurde das Ziel mit der höchsten Motivation ausgewählt wird das Ergebnis an die Aufgabenschicht weitergereicht. [LL08] [BG95]

Die **Aufgabenschicht** dient zur Steuerung und Ausführung von konkreten Anweisungen. Wie diese umgesetzt werden, wird nicht vorgegeben. Blumberg nennt als Beispiel für eine Anweisung "gehe zu diesem Baum", dass bei der Ausführung der Anweisung aber der Weg um ein Hindernis herum genommen werden muss, wird nicht in der Anweisung vorgegeben. [BG95]

Auf der **direkten Schicht** werden direkte Steuerungsanweisungen an den Character gegeben. Diese beeinflussen die Geometrie des Characters und erzeugen so die Bewegungsanimationen. Auf dieser Schicht werden beispielsweise Gehbewegungen animiert und auf dem 3D-Modell des Characters umgesetzt. [Sha06] [BG95]

2.1.4 Gegenüberstellnug der verschieden Schichtenmodelle

Die drei vorgestellten Modelle von Hoogendoorn, Reynolds und Blumberg zeigen einen vergleichbaren, aber nicht identischen Ansatz zur Strukturierung der Bewegungssimulation in

einzelne Schichten. Während sich Hoogendoorn konkret auf die Fußgängersimulation bezieht, konzentrieren sich die Arbeiten von Reynolds und Blumberg auf die Simulation autonomer Charaktere/NPCs für Computerspiele. Die beiden Modelle beschreiben technische Schichten der Implementierung. Sie sollen eine Struktur für die Programmierung vorgeben. Dagegen ist das Modell von Hoogendoorn dazu gedacht das Verhalten von Fußgängern auf einer qualitativen Ebene zu erklären.

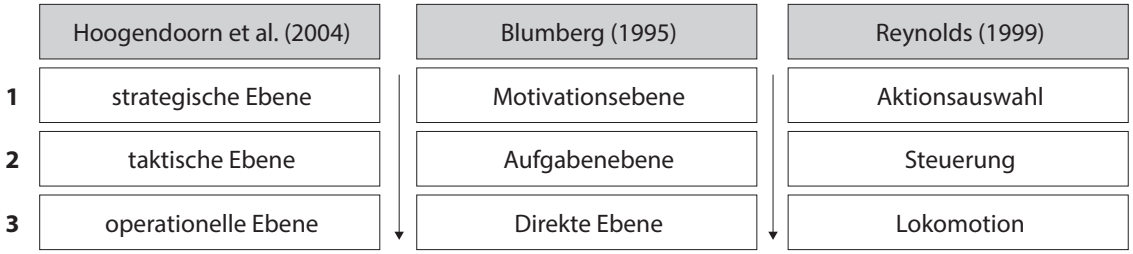


Abbildung 2.2: Schichtenmodelle für Bewegungssimulation nach Hoogendoorn [HB04], Reynolds [Rey99] und Blumberg [BG95]

Wie in Abbildung 2.2 zu sehen, beschreibt die oberste Schicht in allen Modellen den Grund, die Motivation oder Absicht einer Bewegung. Auf der zweiten Schicht geht es um die konkrete Umsetzung dieser Absicht und auf niedrigster Ebene dann um die Bewegungen die ausgeführt werden.

Die Simulation von Fußgängern ist, nach den oben erläuterten Schichtenmodellen auf den Ebenen 1 und 2 angesiedelt. Die Aufgaben dieser Schichten werden in verschiedenen Simulationsmodellen implementiert (vgl. Kapitel 2.2). Bei der Post-Visualisierung von Fußgängern liegen bereits fertige Simulationsergebnisse vor. Hierbei wurde sowohl das Modell zur Simulation (vgl. Kapitel 2.2) schon ausgewählt, als auch Trajektorien berechnet, die die Wege der Fußgänger beschreiben (vgl. Kapitel 2.4.2). Somit sind die Daten auf den Schichten 1 und 2 bereits generiert. Die Post-Visualisierung beschäftigt sich mit der Ausführung und der Darstellung der Bewegung (Schicht 3), die Richtungsanweisungen der einzelnen Fußgänger in konkrete Bewegungsabläufe übersetzt.

2.2 Modellierungsmethoden zur Fußgängersimulation

Modelle zur Fußgängersimulation können in mikroskopische und makroskopische Modelle unterteilt werden. [DH03]

Die mikroskopischen Modelle betrachten jeden Fußgänger als eigenes Objekt und berechnen für jedes Objekt einen individuellen Weg, anhand des Ziels und der Interaktion mit anderen Objekten. [TTI00]

Makroskopische Modelle versuchen Fußgängerströme zu modellieren, in dem sie diese als Gesamtes betrachten und nicht jeden Fußgänger individuell. Hierbei sind auch Informationen über die gesamte Szene (Anzahl der Fußgänger, Durchschnittliche Dichte, etc.) bekannt und können als Parameter für die Simulation verwendet werden. Das Verhalten eines einzelnen Fußgängers ist bei makrosokopischen Modellen nicht von Intersse, sondern nur die kollektive Bewegung aller Fußgänger. Die Verwendung solcher Modelle ist jedoch nur sinnvoll, wenn

die Fußgänger als Kollektiv/Strom unterwegs sind. Für sehr niedrige Fußgängerdichten oder eine geringe Anzahl von Fußgängern müssen mikroskopische Modelle verwendet werden. Für Szenarien mit vielen Fußgängern, bei denen die Strombewegung von Interesse ist, sind makroskopische Modelle geeignet. [JZWL10] [Kne13]

2.2.1 Mikroskopische Modellierungsmethoden

Social-Force-Modell

Dieses Kräftemodell ist das weitest verbreitete mikroskopische Modell und wurde 1995 von Helbing und Molnár [HM95] vorgestellt. Es nimmt an, dass auf jeden Fußgänger anziehende und abstoßende Kräfte wirken. Die Summe dieser Kräfte ergibt dann die Richtung der Bewegung. Jedoch wird die Bewegung des Fußgängers nur durch externe Faktoren bestimmt und kein eigener Wille des Fußgängers implementiert. [LKF05]

Anziehende Kräfte auf einen Fußgänger können beispielsweise das eigentliche Ziel der Bewegung, Gruppenmitglieder oder Schaufenster sein. Abstoßende Kräfte dagegen sind Hindernisse oder fremde Fußgänger. Eine Herausforderung, vor allem bei einer hohen Anzahl an Kräften, ist die Gewichtung der einzelnen Kräfte. [HM95] Ein Problem des Modells ist, dass es passieren kann, dass Fußgänger in einem lokalen Minimum hängen bleiben und z.B. zwischen zwei Wänden oszillieren. Diese Gefahr besteht hauptsächlich bei geringen Dichten. Bei hohen Dichten dagegen ist das Modell dagegen gut geeignet, da der Fußgänger seinen individuellen Willen nicht mehr durchsetzen kann und den physikalischen Kräften der Menge folgen muss. [MPG+10]

Zelluläre Automaten

Ein zellulärer Automat ist ein System bestehend aus einem Gitter, dessen Zellen einen bestimmten Zustand haben. Diese Zustände der Zellen verändern sich in diskreten Zeitschritten anhand bestimmter Regeln. Grundlage für die Fußgängersimulation ist ein Gitter mit dreieckigen, quadratischen oder hexagonalen Zellen. [BKSZ01] Diese Zellen können entweder leer sein, ein Hindernis oder genau einen Fußgänger enthalten. Durch die Größe und Form der Zellen ist allerdings die maximale Dichte (alle Zellen mit je einem Fußgänger belegt) und die exakten Wege (durch die Form der Zellen) vorbestimmt. [KR03]

Für jeden Fußgänger werden alle benachbarten, im nächsten Schritt erreichbaren Zellen, bewertet und daraufhin die Zelle ausgewählt, die die beste Bewertung hat. Gegebenenfalls müssen Konflikte zwischen mehreren Fußgängern, die die gleiche Zelle betreten wollen gelöst werden. Erweiterungen des einfachen zellulären Automaten mit einem Kräftemodell können bessere Ergebnisse liefern. [KR03] [KKS]

Agentenbasierte Ansätze

Ein verbreiteter Ansatz für Simulationen aller Art ist die agentenbasierte Modellierung. Jeder Fußgänger (genannt "Agent") handelt und entscheidet dabei autonom. Jeder Agent hat dafür ein eignes Regelset ("behaviour rule set") anhand dessen er seine Entscheidungen

trifft. [KHW01] So können auch komplexere Abläufe (mehrere Ziele in bestimmten Reihenfolgen) oder individuelle Unterschiede modelliert werden. [SOHTG99]

Das definieren der Regelsets ist hier die größte Herausforderung und überschneidet sich mit dem Forschungsfeld der künstlichen Intelligenz. Daraus resultiert auch ein hoher Berechnungsaufwand für die Simulationen. [KHW01]

2.2.2 Makroskopische Modellierungsmethoden

Netzwerkflussmodelle

Zur Simulation des Fußgängerflusses können auch Graphen oder Netzwerke verwendet werden. Die Quell- und Zielpunkte sind Knoten des Graphs. Zusätzlich müssen noch weitere Knoten des Navigationsgraphs bestimmt werden. Es gibt verschiedene Methoden zur Berechnung von Navigationgraphen, wie zum Beispiel den Sichtbarkeitsgraph, der jeden Hindernis-Polygonpunkt als Knoten verwendet und Kanten zwischen allen Knoten einfügt, auf deren Verbindung kein Hindernis liegt. [CJ03] So entsteht ein Graph, auf dessen Kanten sich die Fußgängerströme bewegen können. Kantengewichte können über verschiedene Faktoren, wie z.B. Länge des Wegs, Anzahl der Personen auf dem Weg, etc. bestimmt werden. [Løv94]

Die Simulation des Fußgängerfluss kann mit graphentheoretischen Algorithmen wie *Fastest Path* oder einer heuristischen Wegewahl durchgeführt werden. [CJ03]

Strömungsbasierte Modelle

Aus der Physik werden strömungsbasierte oder gaskinetische Modelle übernommen, die das Verhalten von Fußgängerströmen analog zum Verhalten von Flüssigkeiten oder Gasen beschreiben. Die einzelnen Fußgänger entsprechen dabei den Atomen/Molekülen oder Partikeln. Erste Modelle dazu wurden bereits in den 1970er-Jahren von Henderson [Hen74] vorgeschlagen. Allerdings weisen diese Modelle einige Probleme auf, denn kognitive, psychologische und soziologische Aspekte werden hier nicht berücksichtigt. [MIN99]

Auf der makroskopischen Ebene beschreiben Schadschneider et al. einige kollektive Effekte und selbst-organisierende Phänomene, die bei Fußgängerströmen beobachtet werden können. Diese können mit strömungsbasierten Modellen gut dargestellt werden: [SKK+09]

Übersteigt die Zahl der Fußgänger die Kapazität eines Orts, entsteht Stau. Üblicherweise passiert dies an Stellen, an denen sich Wege verengen. Diese werden dann als "bottleneck" (engl. Flaschenhals) bezeichnet. Stau kann sich auch bilden, wenn zwei Ströme in entgegengesetzte Richtungen auf einander treffen. Hier kann oft auch eine Spurbildung beobachtet werden, bei der die verfügbare Breite des Wegs in eine oder mehrere Spuren pro Richtung aufgeteilt wird, um so Kollisionen zu vermeiden. Lässt ein Bottleneck nur Durchfluss in einer Richtung zu, kann es zu Oszillieren kommen. Durchläuft ein Fußgänger das Bottleneck, ist es für weitere Personen einfacher zu folgen. Schafft dann ein Fußgänger der anderen Richtung den Weg durch das Bottleneck, dreht sich die Durchflussrichtung um. [SKK+09] [HD05] Solche Phänomene können auch in der Fluiddynamik beobachtet werden und daher bieten sich strömungsbasierte Modelle an, um Mechaniken dieser Art zu simulieren.

2.3 Aufzeichnung realer Daten

Simulationen eignen sich dazu, das Fußgängerverhalten in unterschiedlichen baulichen Umgebungen zu analysieren. Um jedoch neue Simulationsmodelle zu entwickeln, bestehende zu verbessern und zu verifizieren werden Realdaten benötigt. Deshalb ist auch die Aufzeichnung und Aufbereitung von echten Fußgängerströmen ein interessantes Forschungsfeld. Für die Visualisierung können neben Simulationsdaten auch Realdaten verwendet werden.

Für die Aufzeichnung von mikroskopischen Daten gibt es verschiedene Methoden: Optische Messung (Videokameras, Infrarot), Radarsysteme oder GPS. [ISS09] Die Aufzeichnung mit Videokameras ist am weitesten verbreitet. Eine große Herausforderung ist hier die automatische Extraktion der Bewegungsdaten aus dem Videobild um ein arbeitsintensives manuelles Extrahieren zu vermeiden. Hoogendoorn et al. [HDB03] beschreiben den Aufbau eines Experiments zur automatischen Extraktion der Bewegungsdaten aus Videoaufnahmen. Der Aufbau beschreibt eine kontrolliere Experimentumgebung mit Probanden und kann so einige Probleme lösen, die außerhalb der Laborbedingungen auftreten können. Zur Aufzeichnung der Daten wurde eine Kamera über dem Aufbau angebracht, die das Experiment aus der Draufsicht filmt. Die Probanden tragen farbige Kopfbedeckungen zur besseren Erkennung und eine gleichmäßige Beleuchtung verbessert die Bildqualität.

Die algorithmische Auswertung zur Erzeugung der mikroskopischen Bewegungsdaten beinhaltet folgende Schritte: [HDB03]

1. Extraktion einzelner Standbilder aus dem Videomaterial
2. Korrektur von perspektivischen Verzerrungen und Normalisierung der Farb- und Helligkeitsinformationen des Frames
3. Fußgängererkennung durch verschiedene Algorithmen. Beispielsweise durch Subtraktion eines Referenzbilds (leere Szene, ohne Fußgänger), Anwendung eines Schwellwert-Filters und Clustering der Pixel zu einzelnen Fußgänger-Objekten.
4. Zeitliche Verknüpfung der Fußgänger-Objekte über die Einzelbilder zur Erstellung von Trajektorien

Die so erzeugten Daten können dann wie Simulationsdaten zur Post-Visualisierung genutzt werden.

2.4 Parameter und Messgrößen für Simulationen

2.4.1 Eingabeparameter in Fußgängersimulationen

Für eine Simulation müssen einige grundlegende Parameter gesetzt werden, die das Szenario beschreiben und als Eingabeparameter für das Simulationsmodell dienen. Welche Eingabeparameter das jeweilige Modell benötigt hängt stark vom Modell und seiner Implementierung ab. Trotzdem lassen sich einige wichtige Eingabeparameter für Simulationen mit mikroskopischen Modellen benennen: [Kne13]

– **Platzbedarf einer Person/maximale Dichten** Beschreibt den minimalen Platzbedarf einer Person oder im Kehrwert die maximale Anzahl an Personen pro Fläche (maximale Dichte). Angenommen wird $0,15\mathrm{m}^2/\mathrm{P}$ bzw. $6,6\mathrm{P}/\mathrm{m}^2$. [Wei93]

- **Simulationszeit** Um den Rechenaufwand und die Datenmengen gering zu halten, sollte die Simulationszeit auf einen kurzen, relevanten Bereich beschränkt werden.
- **Geschwindigkeitsverteilung** Durchschnittliche Geschwindigkeit eines Fußgängers ohne Behinderung ("free flow speed"). Üblich ist eine Normalverteilung um den Mittelwert von $1,34\text{m/s}$ ($\sigma = 0,26$) [Wei93]
- **Gruppengroßenverteilung** Ob Fußgänger alleine oder in bestimmten Gruppengrößen unterwegs sind hängt stark vom Szenario ab.

Entsprechend der Verteilungen werden dann für die simulierten Fußgänger zufällig Werte bestimmt.

Für Quell- und Zielpunkte kann definiert werden wie viele Fußgänger pro Sekunde erzeugt bzw. entfernt werden können.

2.4.2 Datengrundlage für die Post-Visualisierung

Die Simulation erzeugt mit den Eingabeparametern auf Basis des gewählten Simulationsmodells dann die Simulationsergebnisse. Diese sind wiederum die Datengrundlage für die Post-Visualisierung. Die bei der Simulation erzeugten Ergebnisdaten können je nach Simulationsverfahren und Anwendungsfall unterschiedliche Inhalte und Strukturen haben. Grundsätzlich gibt es aber immer raumzeitliche Fußgängerdaten, die die Positionen der Fußgänger beschreiben und statische Geometriedaten, die Hindernisse (z.B. Gebäude, Mauern, Bäume, etc.) und Bereiche (z.B. Quelle, Ziel) beinhalten. Damit eine Post-Visualisierung möglich ist wird eine Mindestmenge an Daten benötigt, die wie folgt definiert wird:

Fußgängerdaten

Eine Fußgängerposition p wird mit $p(\text{ID}, t)$ bezeichnet. Die ID identifiziert einen bestimmten Fußgänger über alle Zeitschritte hinweg und ist notwendig um beispielsweise die zurückgelegte Stecke eines Fußgängers zwischen zwei Zeitpunkten zu berechnen. Der Zeitstempel $t \in \mathbb{R}_0^+$ bezeichnet einen diskreten Zeitpunkt seit Beginn der Simulation ($t = 0$).
Die Fußgängerposition p wird auf ein n-Tupel abgebildet, der eine zwei- oder dreidimensionale Positionsangabe ($x, y, z \in \mathbb{R}_0^+$) enthält, und noch weitere Daten wie eine vorberechnete Dichte, Gruppenzugehörigkeit, etc. beinhalten kann.
Somit gilt für zweidimensionale Fußgängerdaten: $p(\text{ID}, t) \rightarrow (x, y)$
Die Fußgängerdaten sind die Menge P aller Fußgängerpositionen $p \in P$, die bei der Simulation erzeugt wurden.

Geometriedaten

Ein Geometrieobjekt g besteht mindestens aus einem offenen oder geschlossenen Kantenzug $k := (q_1, q_2, ..., q_n)$ aus $n \geq 2$ zwei- oder dreidimensionalen Punkten ($x, y, z \in \mathbb{R}_0^+$).
Zusätzlich kann es noch weitere Informationen wie Höhe, Bezeichnung, etc. enthalten. Die Geometriedaten sind die Menge G aller Geometrieobjekte $g \in G$, die für die Simulation verwendet wurden.

S bezeichnet die zweidimensionale Grundfläche der Simulation, innerhalb derer sich alle Objekte befinden und wird dementsprechend wie folgt definiert $S := ((0,0),(x_{max},y_{max}))$. Oft wird die Grundfläche auch als eigenes Geometrieobjekt angelegt.

Weitere optionale Datensätze können beispielsweise die Gruppenzuordnung enthalten, die einzelne Fußgänger zu größeren Gruppen, die gemeinsam unterwegs sind, verbindet.

2.4.3 Messgrößen in Fußgängersimulationen

Zum Vergleich mehrerer Szenarien und Simulationsmodelle sowie zur Analyse können neben der Eingabeparameter können, nach Ablauf der Simulation, Werte auf den Simulationsergenissen gemessen werden. Auf Grund der zufälligen Generierung und abhängig vom Simulationsverfahren können sich die Messwerte, auch für mehrere Simulationsdurchgänge des selben Szenarios, unterscheiden. Selbst dann, wenn die gleichen Eingabeparameter verwendet wurden.

Folgende Werte sind häufig verwendete Kennzahlen zur Analyse von Fußgängerströmen und können anhand einer erfolgten Simulation ausgewertet werden. Die Werte, die auf einer Verteilungsfunktion beruhen, werden folglich auch in der Auswertung diese Verteilung zeigen. Allerdings kann es dennoch interessant sein diese Werte nicht für die gesamte Szene, sondern nur für ausgewählte Bereiche zu analysieren. [Løv94]

- **Population** Die Zahl der aktiven Fußgängerobjekte verändert sich über den zeitlichen Verlauf abhängig von Emission/Absorption der Quelle(n)/Ziel(e).
- **Dichte** beschreibt die Anzahl an Personen pro Flächeneinheit. Sie kann nach räumlicher oder zeitlicher Dimension aufgeschlüsselt werden und verwendet in der anderen Dimension Mittelwerte.
- **Fluss** beschreibt die Anzahl an Personen die eine bestimmte Messlinie pro Zeit- und Längeneinheit überqueren. Zur Auswertung muss eine Linie definiert werden, auf der der Fluss gemessen wird.
- **Fundamentaldiagram** bezeichnet im Verkehrsingenieurwesen den Zusammenhang zwischen Größen Fluss q, Dichte k und Geschwindigkeit v. Für jeden Fußgänger ergibt sich mit diesen Werten ein dreidimensionaler Punkt. Die Summe der Punkte aller Fußgänger kann als dreidimensionale Punktwolke ("scatter plot") dargestellt werden. Üblicherweise wird jedoch eine Projektion auf zwei der drei Größen vorgenommen. Die drei Diagramme werden als q-v-Diagramm, q-k-Diagramm und k-v-Diagramm bezeichnet. [Küh04]

2.5 Bestehende Systeme zur Post-Visualiserung

2.5.1 SumoViz

Mustafa K. Isiks "SumoViz" implementierte eine zweidimensionale Post-Visualisierung auf Basis des HTML5-canvas-Elements und der damit Verbundenen JavaScrip-API zum Zeichnen von 2D-Inhalten. [Isi12] Die Visualisierung zeigte eine einfache Draufsicht der Simulation, die alle Hindernisse als Linien darstellt. Die Fußgänger werden als einfache Punkte eingezeichnet.

Der Ablauf der Simulation kann frei gesteuert werden und die Entwicklung der Population wird zur Analyse als Graph dargestellt. Jedoch ist die Darstellung ist nicht anpassbar und der Bildausschnitt kann nicht verändert werden. Minh-Quang Nguyen erweiterte 2012 die Analysewerkzeuge, damit ist es möglich ist Dichte, Geschwindigkeit und Fluss für einzelne Bereiche zu ermitteln und als Graph darzustellen. [Ngu12]

2.5.2 SumoViz3D

In meiner Bachelorarbeit [Büc12] aus dem Jahr 2012 wird die Nutzung von Web-Technologien zur Simulations-Visualisierung betrachtet. Sowohl bei der Verarbeitung der Daten als auch der Darstellung wurden dabei ausschließlich offene Web-Standards eingesetzt. Die Darstellung von 3D-Inhalten im Browser mit *WebGL*, einer Grafikbibliothek auf der Basis von *OpenGL ES 2.0*, wurde auf ihre Leistungsfähigkeit und Nutzbarkeit für Simulations-Visualisierung hin untersucht. Dazu wurde die Applikation "SumoViz3D" als Weiterentwicklung der "SumoViz"-Applikation (vgl. Kapitel 2.5.1) entwickelt. SumoViz3D nutzt die vorhandenen Datenstrukturen von SumoViz und implementierte darauf eine interaktive 3D-Darstellung der Simulationsdaten mit WebGL.

WebGL ist eine plattformunabhängige und browserübergreifende Lösung zur Darstellung hardwarebeschleunigter 3D-Inhalte in modernen Webbrowsern. Mit WebGL als offenem Standard für 3D-Inhalte im Browser ist eine Darstellung der Inhalte ohne Plugins möglich, sofern der Browser WebGL unterstützt. [CJ12] Allerdings ist die Unterstützung und Umsetzung innerhalb der Browser sehr unterschiedlich und hat sich seither auch nicht wesentlich verbessert. [CIU]

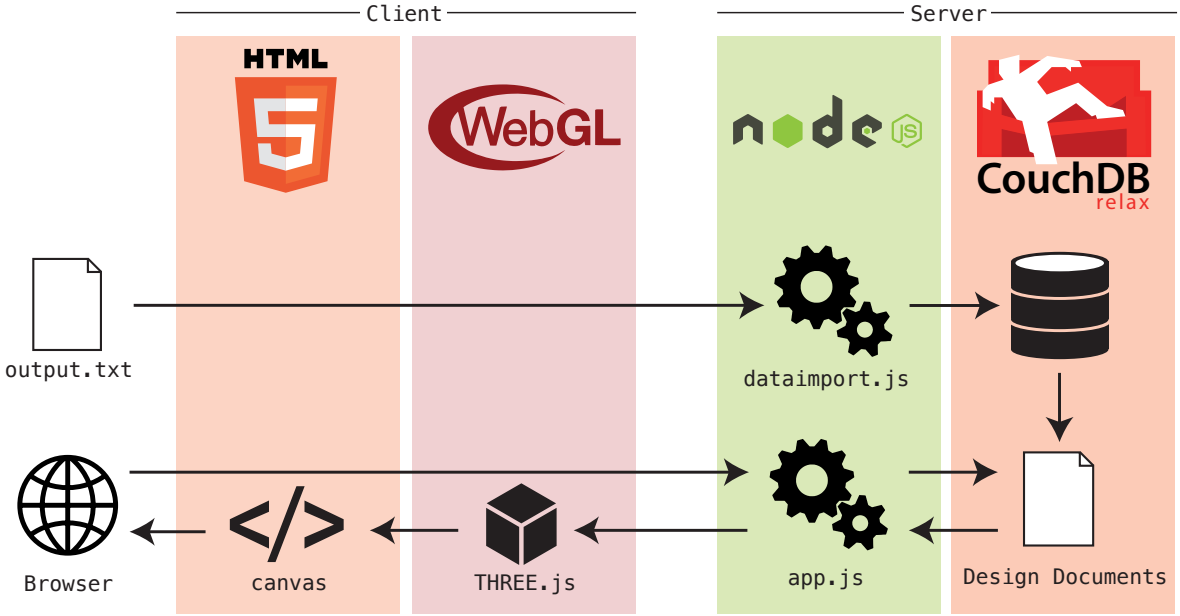


Abbildung 2.3: Struktur der SumoViz3D-Komponenten [Büc12]

SumoViz3D kann Leerzeichen-getrennte Datensätze von Fußgänger-, Geometrie- und Gruppendaten importieren und hält diese in *CouchDB*, einer dokumentenorientierte Datenbank, vor (vgl. Abbildung 2.3). Die Simulationsergebnisse der Fußgängerdaten bestehen aus einzel-

nen Simulationsschritten, denen jeweils ein Zeitstempel zugeordnet ist. Die Geometriedaten bestehen im Wesentlichen aus einfachen Polygonen, die Gebäude, Mauern oder Areale beschreiben und für den gesamten Zeitraum der Simulation unverändert bleiben. Die Daten werden dann von CouchDB transformiert und per HTTP-Schnittstelle (vgl. Abbildung 2.3 `app.js`) an den Browser ausgeliefert. Dort wird in einem HTML5-`canvas`-Element ein WebGL-Kontext erzeugt und die 3D-Visualisierung dargestellt. Die Darstellung erfolgt in Echtzeit und wird nicht vorgerendert. Die Nutzer können sich mit Maus und Tastatur frei durch die geladene Geometrie bewegen, sowie den Winkel und Entfernung der Kamera beeinflussen. Die 3D-Bibliothek `THREE.js` stellt dabei grundlegende 3D-Objekte und Methoden bereit. [Büc12]

Die Schritte der Simulationsergebnisse können dann einzeln dargestellt werden oder fortlaufend nacheinander abgespielt werden. Während der Nutzung kann die Position der Kamera frei verändert werden und es können Standbilder von einzelnen Simulationsschritten abgespeichert werden. Die visuelle Erscheinung der Szene kann mit ein paar einfachen Funktionen angepasst werden. Für die Grundfläche und die Geometrie können verschiedene grafische Texturen ausgewählt werden. Außerdem lassen sich 3D-Modelle von Bäumen laden und in der Szene platzieren. Zur Darstellung der Fußgängerobjekte wird wahlweise ein einfaches 3D-Modell einer Figur genutzt oder eine PNG-Grafik als sogenanntes Partikelsystem. Für große Datensätze ist die Darstellung mit Hilfe der Grafik performanter, da dann jedes Fußgängerobjekt nur aus einem einzelnen Vertex besteht. [Büc12]

Zur Auswertung werden die Fußgängerobjekte entsprechend bestimmter Kennwerte eingefärbt. Die Ausgangsdaten beinhalten einen vorberechneten Dichtewert, der die lokale Dichte eines einzelnen Fußgängerobjekts beschreibt. Zu einigen Datensätzen gibt es Gruppenzugehörigkeiten, die mehrere Fußgänger zu Gruppen zusammenfassen. Diese können so in der jeweils selben Farbe dargestellt werden. Außerdem kann, aus der zum vorigen Schritt zurückgelegten Strecke, dynamisch die Geschwindigkeit eines Fußgängers berechnet werden und entsprechend eingefärbt werden. [Büc12]

Kapitel 3

Implementierung einer Post-Visualisierung für Fußgängersimulationsdaten

3.1 Neuimplementierung von SumoViz3D

Die Unterstützung von WebGL auf Seiten der Browser hat sich nicht wie erwartet verbessert und auf mobilen Plattformen gibt es weiterhin nahezu keine Unterstützung. [CIU] Deshalb soll auf Basis einer vorhandenen 3D-Game-Engine eine Neuimplementierung von SumoViz3D gemacht werden. Die Leistungsfähigkeit der browserbasierten Simulation ist in den Test relativ schnell an eine Leistungsgrenze gestoßen. Etwa 2000 Fußgänger ließen sich als Polygonobjekte flüssig darstellen. Dabei handelte es sich um relativ einfach Modelle mit einer geringen Anzahl an Polygonen. Wurden die Fußgänger als Grafiken dargestellt, ließen sich etwa 10000 Objekte darstellen. [Büc12]

In dieser Arbeit sollen mittels einer Neuimplementierung einige Verbesserungen und Erweiterungen umgesetzt werden.

- **Leistungsfähigere Darstellung** Die Visualisierung ist bei weitem nicht auf dem Niveau, das man von aktuellen Computerspielen gewohnt ist. Mittels der Umsetzung durch eine 3D-Game-Engine lässt sich wahrscheinlich eine wesentlich ansprechendere Visualisierung erzeugen. Komplexere Modelle für Fußgängerobjekte und bessere Shader für die Darstellung von verschiedenen Materialien, Licht und Schatten versprechen eine detailliertere Grafik. Bisher sprang das Fußgängermodell von einem Simulationsschritt zum nächsten. Hier könnte man die Zwischenpositionen interpolieren und so eine flüssige Animation erzeugen. Außerdem können die Gehbewegungen der Fußgänger animiert werden und so deutlich realistischer erscheinen.
- **Schnittstelle für Simulationsdaten** Bisher wurde nur ein bestimmtes Format von Simulationsdaten unterstützt. Hier soll eine abstrakte Schnittstellenbeschreibung ermöglichen, dass beliebige Formate importiert und dargestellt werden können. Die Daten müssen dabei eine Mindestmenge an Informationen, bestehend aus x-y-Koordinaten pro Zeitpunk, enthalten. Dabei soll es egal sein, ob die Daten in einer Datenbank,

strukturierten Dateiformaten, oder einfachen Textdateien vorliegen. Eventuell kann eine Geospatial-Database verwendet werden um Anfragen performant auszuführen.

- **Auswertungswerkzeuge** Bisher bot die Software nur die Einfärbung der Fußgängerobjekte zur Auswertung einiger Kennwerte. In der Neuimplementierung sollen die Werkzeuge zur Analyse deutlich erweitert werden. In Graphen soll die Entwicklung der Geschwindigkeit, Dichte und Anzahl der Fußgänger über die Zeit ausgewertet werden können. Ebenso sollen die Werte für Geschwindigkeit und Dichte nicht nur farblich dargestellt werden, sondern auf reale Werte umgerechnet werden. Zur interaktiven Auswertung kann der Durchfluss an bestimmten Bereichen innerhalb der Szene gemessen werden.

- **Plattformübergreifende Darstellung** Mit der Verwendung einer plattformübergreifenden Engine, wie beispielsweise *Unity3D* ist auch die Nutzung auf Tablets denkbar. Die erweiterten Eingabemöglichkeiten über Touch und Beschleunigungssensoren ermöglicht eine einfachere Bedienung und mobile Nutzung.

3.2 Nutzung von Game-Engines für wissenschaftliche Visualisierungen

3.2.1 Grundlagen zu Game-Engines

Das berechnen von 3D-Grafik zur Erzeugung von virtuellen Welten ist eine komplexe Herausforderung. Früher wurde oft spezielle Hardware genutzt um hochqualitative 3D-Grafik zu berechnen. Heute können selbst aufwändige Computerspiele oder Animationsfilme auf Standard-Hardware berechnet werden. [TS08]

Die beiden Bereiche, mit den größten wirtschaftlichen Interessen an 3D-Grafik und damit auch große Innovationstreiber, sind einerseits die Animationsfilm-Branche und andererseits die Computerspiel-Branche mit 65 Milliarden Dollar Umsatz (2011) [Reu11]. Die Anforderungen und Herangehensweisen dieser beiden Branchen unterscheiden sich in der Interaktivität des Produkts. Während bei Erstellung eines Animationsfilms die Renderzeit für einen Frame oft mehrere Minuten dauert, müssen Computerspiele die Frames in Echtzeit berechnen, da die Frames auf Grund der Interaktion der Nutzer nicht vorberechnet werden können.

Zur Echtzeit-Darstellung von 3D-Inhalten werden üblicherweise so genannte *Engines* verwendet, die bei der Entwicklung grundlegende Aufgaben abstrahieren und eine Bibliothek an Objekten und Funktionen bereitstellen um die Programmierung zu erleichtern. [TS08]

Die meisten und leistungsfähigsten Engines stammen aus der Entwicklung von Computerspielen. Auf Grund des hohen Entwicklungsaufwands werden solche Game-Engines nicht für jedes Spiel neu entwickelt, sondern mehrfach genutzt. Die Engine an sich, wird in vielen Fällen als kostenloses oder open-source-Produkt angeboten. Die Arbeit evaluiert die Nutzbarkeit der Vorteile solcher Game-Engines für die Simulationsvisualisierung von Fußgängerdaten. [FK04]

Definition 1. Seung Seok Noh et al. [NHP06] definieren eine 3D-Game-Engine als

(i) eine Ausgabe-Komponente für Grafik, Musik und Sound-Effekte mit Eingabe-Unterstützung durch verschiedene Eingabegeräte

(ii) die Algorithmen zur Verfügung stellt um den Character/Spieler bewegen zu können,
(iii) die Topografie der Szene sowie die künstliche Intelligenz steuert,
(iv) und Netzwerk-Funktionalität und -überwachung zur Verfügung stellt.

Oft sind noch Physik-Simulation, Kollisionserkennung, grafische Bedienelemente (GUI/HUD) und eine Event-Verarbeitung durch eine Event-Loop Bestandteil von Game-Engines. [FK04] Diese Funktionalität wird in einem Paket zusammengefasst, auf dessen Basis unterschiedlichste Spiele implementiert werden können.

Andersherum werden Spiele immer für eine konkrete Engine implementiert und verwenden oft spezielle Funktionen, die nur die jeweilige Engine zur Verfügung stellt. Das Portieren eines Spiels von einer Engine auf eine andere ist praktisch unmöglich, da das Spiel sehr stark mit seiner Engine verwoben ist. Daher richten sich Engines meist an Spiele eines bestimmten Genres, das gewisse Eigenschaften mit sich bringen: [Gre09]

- **first-person-shooter** Haben eine Kamera aus der Egoperspektive, die vom Spieler durch Bewegen des Characters mit Maus und Tastatur bewegt werden kann. So kann sich frei durch die Szene bewegt werden. Oft haben diese Spiele auch eine hochgradig detaillierte 3D-Grafik und zeigen sich für die Entwicklung neuer 3D-Technologien verantwortlich. [Gre09]
- **Echtzeitstrategie** Üblicherweise mit einer Kamera aus der Vogelperspektive und oft mit isometrischer Darstellung. Gesteuert werden diese Spiele meist durch einzelne Maus-Eingaben des Spielers. [Gre09]
- **platformer/sidescroller** Haben eine Kamera aus der Third-Person-Perspektive, so dass der Character an sich sichtbar ist. Die Steuerung der Kamera übernimmt die Spiellogik. Die Bewegungsfreiheit des Characters ist meist auf zwei Dimensionen limitiert. Die Eingabe erfolgt über die Tastatur. Der Einsatz einer 3D-Engine ist hierfür nicht unbedingt notwendig. [Gre09]

Für die Auswahl einer passenden Game-Engine ist also zunächst einmal entscheidend welchem Genre das Spiel (oder die Visualisierung) am ehesten entspricht. Vor allem die Kameraperspektiven und Eingabemethoden sind sehr genrespezifisch. Die Anforderungen der Fußgängervisualisierung entsprechen am ehesten denen eines first-person-shooters, da sowohl detaillierte 3D-Grafik, als auch eine frei bewegliche Kamera benötigt werden.

Neben der Engine sind weitere Komponenten eines Computerspiels sind die Spiellogik ("game logic"), die die Regeln und das Verhalten des Spiels festlegt und die Topografie ("level data"), die die Landschaft/Umgebung eines Spiels/Levels beschreibt. [LJ02] Im Bezug auf die Simulationsvisualisierung kümmert sich die Spiellogik um die Bewegung der Fußgänger, die Topografie entspricht den Geometriedaten.

3.2.2 Aufbau einer Game-Engine

Eine Game-Engine ist ein komplexes und umfangreiches Softwarepaket und wird in einzelne Schichten gegliedert, die aufeinander aufbauen und Komplexität zur nächsthöheren Schicht abstrahieren. Jason Gregory [Gre09] beschreibt die einzelnen Schichten wie folgt (vgl. Abbildung 3.1):

Das Betriebssystem stellt Grafikfunktionen des Grafikprozessors (*GPU*) in einer Bibliothek zur Verfügung. Dies ist die Basis auf der auch Game-Engines aufbauen. Die zwei am häufigsten genutzten Bibliotheken sind *OpenGL* der Khronos Group und Microsofts *Direct3D*. OpenGL ist plattform- und programmiersprachenunabhängig und wird als offener Standard in einem Konsortium vieler großer Unternehmen (darunter AMD, Intel, NVIDIA und Google) entwickelt. Es läuft unter anderem auf Windows und Mac OS, auf den mobilen Betriebssystemen iOS und Android (*Open GL for Embedded Systems*), der PlayStation 2 und 3 und in einigen Browsern als *WebGL*. [Ebe06] Dem gegenüber steht Microsofts Direct3D für Windows und die XBox, das eine sehr gute objektorientierte und fortschrittliche Programmier-Schnittstelle bietet. Direct3D ist Bestandteil von *DirectX*, einem umfassenden Paket mit vielen Bibliotheken für Grafik, Sound und Eingabe. [LJ02] [Ebe06]

Eine Game-Engine abstrahiert die Grafikbibliothek, so dass nicht direkt mit deren Programmier-Schnittstelle interagiert werden muss. Viele Game-Engines nutzen sowohl OpenGL als auch DirectX um mehrere Plattformen zu unterstützen. [LJ02]

Engines, die auf mehreren Systemen laufen implementieren darüber eine **Plattformunabhängigkeitsschicht**. Diese abstrahiert die Gerätekommunikation für die Engine und standardisiert beispielsweise den Zugriff auf das Dateisystem und die Netzwerk-Transportschicht (UDP/TCP) und implementiert Threading-Modelle plattformspezifisch. [Gre09]

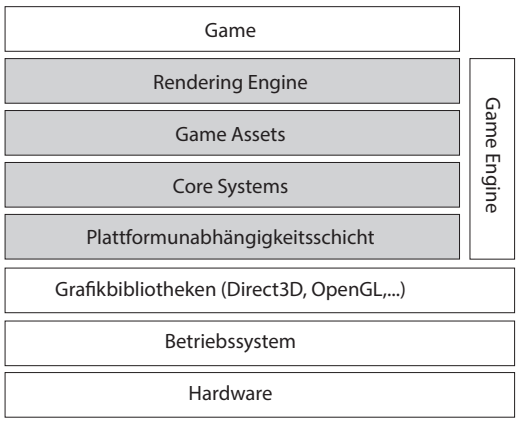


Abbildung 3.1: Schichtenmodell der Architektur einer Game-Engine [Gre09]

Die **Core-Systems-Schicht** ist eine Sammlung verschiedner Datenstrukturen, Werkzeuge und Hilfsroutinen, die die Nutzung der Engine erleichtern sollen. Der Umfang dieser Schicht unterscheidet sich stark zwischen verschiedenen Engines. Bestandteile sind beispielsweise Funktionen zur Speicherverwaltung, Mathematik-Bibliotheken, Parser (XML, JSON, etc.), Werkzeuge zur Lokalisierung, Movie Player, Debugging- und Testing-Funktionen, etc. [Gre09]

Spiele benötigen oft externe Ressourcen (*Assets*) wie 3D-Modelle, Grafiken, Schriften, Texturen, Materialen, etc. für die Darstellung. Diese sind meist in eigenen Dateien gespeichert oder in Archiven zusammengefasst und werden von der **Game-Asset-Schicht** geladen und bereitgestellt. [Gre09]

Wichtigster und größter Bestandteil einer Game-Engine ist die **Rendering-Engine**. Diese erzeugt aus der Beschreibung von 3D-Modellen, Texturen, Positionsinformationen von Lichtern und Kamera ein fertiges Bild zur Darstellung. Die dreidimensionalen Koordinaten aus der Szene werden mit einer Matrixtransformation auf die zweidimensionale Bildebene projiziert. Lichter, Schatten, Materialeigenschaften und Texturen werden dann darauf angewendet und somit die Farbe bestimmt, in der der jeweilige Pixel erscheint. [Ebe06]

Zur Geschwindigkeitsoptimierung sollen nicht alle Objekte gerendert werden, die sich in der Szene befinden, sondern nur Objekte die sichtbar sind. Welche Objekte sichtbar sind bestimmt vor allem die Position und Ausrichtung der virtuellen Kamera. Diese beschreibt einen Pyramidenstumpf (*viewing frustum*) der in der Szene liegt. Alle Objekte die außerhalb davon liegen sind nicht sichtbar und werden garnicht erst berechnet (*frusutm culling*). Selbiges gilt

für Objekte die zu nah oder zu weit entfernt sind (*near-* und *far-plane*). [LJ02] Objekte deren 3D-Modell zwar in einer hohen Auflösung vorhanden ist, aber sehr weit entfernt ist und deshalb klein dargestellt wird, kann in der Komplexität reduziert werden (*Level-of-detail-optimzation*). Es gibt noch einige weitere Techniken zur Optimierung und Reduzierung der zu berechnenden Inhalte. [Ebe06]

Weiterhin kann eine Engine grafische Effekte anbieten, die in der Rendering-Engine erzeugt werden. Licht und Schatten, Kantenglättung, Bildeffekte oder Partikelsysteme, mit dem beispielsweise Feuer- oder Raucheffekte erzeugt werden können, gehören dazu.

Meistens wird das 3D-Bild mit 2D-Grafiken überlagert um *heads-up-displays*, Menüs, oder andere GUI-Elemente einzublenden, die dem Nutzer Informationen und Steuerungsmöglichkeiten bieten, aber nicht Teil der 3D-Szene selbst sind. [Gre09]

Welche Inhalte angezeigt werden, entscheidet das Spiel. Es ist nicht mehr Teil der Engine, sondern ist im Schichtenmodell über der Engine angesiedelt und greift auf deren Funktionen zurück. Hier wird die Spiellogik implementiert. Üblicherweise bieten Game-Engines eine Schnittstelle (*Scripting API*) über die die Engine angesprochen wird. Die Engine kann dann fertig kompiliert vorliegen und muss nicht mehr verändert werden, während das Spiel entwickelt wird. Der Code der Spiellogik wird dann zur Laufzeit interpretiert oder ebenfalls kompiliert. [Gre09]

3.2.3 Vergleich aktueller Game-Engines

Wie in Kapitel 3.2.1 beschrieben ist für die Auswahl einer Engine entscheiden welche Inhalte damit dargestellt werden sollen. Die Anforderungen der Post-Visualisierung von Fußgängerdaten ist dem Genre der First-Person-Shooter relativ ähnlich, da die Darstellung der Simulation ebenfalls aus der Egoperspektive erfolgen soll. Deshalb fokussiert sich der Vergleich auf Game-Engines, die nur oder auch für die Nutzung in First-Person-Shootern entwickelt wurden.

Im Mai 2011 hat das "Game Developers Magazine" eine Umfrage unter Spieleentwicklern gemacht und die meist genutzten Engines untersucht. [DeL11] Das Magazin unterscheidet dabei zwischen "traditionellen" Entwicklern, die oft für Spielekonsolen und mit großen Budgets entwickeln und "casual" Entwicklern, die auf einfachere, mobile und soziale Spiele fokussiert sind. Die genaue Abgrenzung dieser Unterteilung wird nicht weiter beschrieben.

Unter Entwicklern, die eine Game-Engine für ihre Spiele verwenden, ist im traditionellen Segment die *Unreal*-Engine die häufigste Wahl (27,8%), gefolgt von *Trinigy* (19,6%) und *Unity* (11,4%). Für casual-games ist Unity die am meisten genutzte Engine (26,0%), vor *Trinigy* (17,8%). Die Unreal-Engine ist in diesem Segment mit 3,0% auf Platz 4. [DeL11]

Im Folgenden werden die unter Spieleentwicklern am häufigsten genutzten Engines Unreal und Unity betrachtet und ihre Tauglichkeit für die Implementierung der Fußgängervisualisierung verglichen. Trinigy bietet ausschließlich kommerzielle Lizenzen an und kann deshalb nicht für die Implementierung genutzt werden. [Tri] Anhand des Vergleichs zwischen Unity und Unreal wird dann eine geeignete Engine für die Visualisierung ausgewählt.

Unreal Die Unreal Engine ist eine der ältesten und bekanntesten Engines auf dem Markt und wurde 1998 von *Epic Games* zusammen mit dem gleichnamigen Shooter erstmals veröffentlicht. Seitdem ist sie in mehreren Generationen (aktuell 4. Generation) weiterentwickelt worden. Viele der erfolgreichsten und grafisch aufwändigsten Spiele wie "Borderlands 2", "BioShock Infinite" oder "Mass Effect 3" verwenden diese Engine [Unr14]. Sie ist bekannt für ihre realistischen Animationen und Lichtberechnungen. Die Programmierung erfolgt in der eigenentwickelten Skriptsprache *UnrealScript* und bringt eine eigene Entwicklungsumgebung mit. UnrealScript weist allerdings eine hohe Komplexität auf: "Die Einstiegshürde [...] muss im Projekt pro Mitarbeiter mit mehreren Personenwochen gerechnet werden." [Str11] Es ist eine kostenlose Variante der dritten Generation der Engine unter dem Namen "Unreal Development Kit" (UDK) verfügbar. Die aktuelle Generation ist nur kostenpflichtig nutzbar. Diese läuft auf Windows, Mac OS und iOS, entwickelt werden kann jedoch nur unter Windows. Als Grafikbibliotheken wird unter Windows Direct3D, unter Mac OS und iOS OpenGL verwendet. [Unra]

Die weite Verbreitung und lange Verfügbarkeit der Engine sorgt für viele Ressourcen, wie Tutorials, Modell- und Textur-Datenbanken. Die Unreal Engine wurde schon für viele wissenschaftliche Projekte verwendet. Darunter Simulation für Evakuierungsszenarien [WLG03], Visualisierungen für den Immobilienmarkt [Mil99], Landschaftsplanung [HP02] oder virtuelle Lernumgebungen [RBS+04]. Im Jahr 2013 stellte Epic Games zusammen mit Browserhersteller Mozilla eine Portierung der Unreal Engine für *Mozilla Firefox* (ab Version 23.0) vor. Diese Portierung übersetzte über eine Million Zeilen Programmcode der Unreal Engine nach JavaScript und WebGL. Bei dieser Implementierung handelt es sich um einen proof-of-concept und die gezeigte Demonstration "Epic Citadel" ist die einzige Verwendung dieser Technologie. [Moz13]

Unity3D Unity3D ist eine Engine die von *Unity Technologies* entwickelt wird und aktuell in der Version 4.3 vorliegt. Die Engine wird mit zwei verschiedenen Lizenzmodellen angeboten. Einer kostenlosen Variante und einer kostenpflichtigen Pro-Version, die für kommerzielle Projekte benötigt wird und einige erweiterte Funktionen (vor allem im Bezug auf Shader) mit sich bringt. Zur Entwicklung stellt Unity eine eigene Entwicklungsumgebung für Windows und Mac OS zur Verfügung. Die Programmierung basiert auf der Laufzeitumgebung *Mono* kann in JavaScript, C# oder Boo erfolgen. Die Engine steht für eine Vielzahl von Betriebssystemen (Windows, Mac OS, Linux), mobilen Geräten (iOS, Android, etc.) und sogar Spielekonsolen zur Verfügung. Abhängig von der Zielplattform wird OpenGL oder Direct3D verwendet. Als zusätzliche Plattform kann Unity den Webbrowser verwenden. Allerdings handelt es sich dabei nicht um eine native Implementierung sondern es muss ein Plug-In installiert werden. Unity stellt dafür den "Unity Web Player" zur Verfügung, der vom Nutzer zunächst installiert werden muss, aber für alle gängigen Browser angeboten wird. [Str11]

3.2.4 Anforderungsanalyse und Auswahl einer Engine

Zur Auswahl einer geeigneten Engine ist es notwendig die Anforderungen der Simulationsvisualisierung zu bestimmen und entsprechend mit den Möglichkeiten der Game-Engines zu vergleichen. Nicht alle benötigten Funktionen standen zu Beginn der Entwicklung fest, jedoch kann anhand der vorherigen Implementierung von SumoViz3D eine grobe Anforderungsanalyse erstellt werden.

Die Anforderungen für die Neuimplementierung der Visualisierung, nach denen die Game-Enginges verglichen werden sollen, sind:
(i) Ausführung von HTTP-Requests, (ii) Parsen von JSON/XML-Daten, (iii) Darstellen von HUD-Elemeten, (iv) Verfügbarkeit eines Partikelsystems, (v) Verfügbarkeit einer "Flyover"-Kamera und (vi) dynamisches Erstellen von Objekten zur Laufzeit.

(i) HTTP-Requests Die in SumoViz3D verwendete CouchDB und auch viele andere moderne Datenbanksysteme nutzen eine HTTP-Schnittstelle zur Übertragung der gespeicherten Daten. Zum Abrufen der Simulationsergebnisse muss die Engine in der Lage sein HTTP-Requests auszuführen.

Unreal kann mit der `TcpLink`-Klasse beliebige TCP-Sockets aufbauen. HTTP-Request müssen darüber selbst implementiert werden. [Unrb]

Unity3D ermöglicht mit der `WWW`-Klasse asynchrone HTTP-Anfragen. FTP- und lokale Zugriffe sind damit ebenfalls möglich. [Uni]

(ii) Parser Die per HTTP ausgelieferten Daten müssen in eine interne Repräsentation überführt werden. Dazu muss die Engine die abgerufenen Simulationsergebnisse aus ihrem Übertragungsformat (z.B. JSON oder XML) einlesen können.

Unreal bringt einen eigenen JSON-Parser und -Serialisierer. Eine Unterstützung für XML fehlt jedoch komplett. [Unrb]

Unity3D hat keine eigenen Parser für XML- oder JSON-Daten, aber es gibt eine Vielzahl an Parsern für XML und JSON von Drittentwicklern. [Uni]

(iii) HUD-Elemente Zusätzlich zur 3D-Darstellung ist ein zweidimensionales HUD für Auswertungswerkzeuge und zur Steuerung der Simulation notwendig.

Unreal enthält im UDK (Unreal Engine 3) nur eine grundlegende Unterstützung für HUDs. Alle Interface-Elemente müssen selbst implementiert werden. [Unrb]

Unity3D hat eine `GUI`-Klasse die fertige Buttons, Menüs, Fenster, Dialogboxen, etc. anbietet. [Uni]

(iv) Partikelsystem SumoViz3D nutzt ein Partikelsystem für die effiziente Darstellung großer Fußgängerobjektmengen. Dies könnte auch bei der Neuimplementierung genutzt werden. Wichtig ist hierbei, dass die Positionen der Partikel gezielt gesetzt werden können um sie entsprechend der Fußgängerpositionen zu platzieren.

Unreal besitzt ein sehr modulares und gut erweiterbares System für Partikelsysteme. [Unrb]

Unity3D stellt ein eigenes einfache Partikelsystem in der Klasse `ParticleSystem` zur Verfügung. [Uni]

Plattformen Eine hohe Verfügbarkeit der Visualisierung auf mehreren Plattformen (darunter auch mobile Geräte) wäre wünschenswert, ist aber keine Voraussetzung.

Unreal bzw. das UDK unterstützt den Export für Windows, Mac OS und iOS. [Unrb]

Unity3D exportiert für Windows, Mac OS, Linux, die mobilen Plattformen iOS, Andoird, Blackberry 10 und Windows Phone sowie über den Webplayer in diverse Browser. [Uni]

Weiterhin sind folgende Funktionen wichtige Voraussetzungen für die Simulationsvisualisierung, aber so grundlegender Bestandteil, dass sie von jeder Engine erfüllt werden: Eine "Flyover"-Kamera (v), mit der der Betrachter sich frei durch die Visualisierung bewegen kann und so den betrachteten Ausschnitt verändern kann. Dies ist durch Erfüllung von Definition 1(ii) gegeben. Außerdem ist eine Erzeugung von 3D-Objekten zur Laufzeit über das Scripting-Interface notwendig (vi), da die Simulationsdaten erst zur Laufzeit abgerufen werden. Diese Funktionen sind Bestandteil jeder skriptbaren Engine und werden auch von Unreal und Unity bereitgestellt.

Die Wahl einer Game-Engine ist schwer zu formalisieren und bleibt eine subjektive Entscheidung. Beide Engines erfüllen alle Voraussetzungen für die Umsetzung der Simulationsvisualisierung. Unity3D bietet jedoch eine bessere Unterstützung für die Server-Kommunikation und den Datenaustausch. Aktuell ist Unity eine beliebte Wahl bei Spieleentwicklern, da der Einstieg relativ einfach ist. Zusätzlich gibt es auch eine große Menge an Ressourcen für Entwickler. Deshalb wird für die Neuimplementierung der Visualisierung die Unity3D-Engine verwendet.

3.2.5 Voraussetzungen und Anforderungen für die Simulationsvisualisierung

Die Aufgabe der Simulationsvisualisierung ist es Ergebnisdaten einer Simulation nach deren Fertigstellung zu visualisieren ("Post-Visualisierung"). Es sollen folgende Voraussetzungen für die Simulationsvisualisierung gelten:

Definition 2. Voraussetzungen für eine Post-Visualisierung von Simulationsdaten

(i) Zum Startpunkt der Visualisierung liegen bereits alle Simulations-Ergebnisse vor.
(ii) Alle zur Visualisierung benötigten Informationen sind in den Simulations-Ergebnisdaten enthalten oder können daraus eindeutig berechnet werden.
(iii) Es werden ausschließlich die Ergebnisdaten verwendet und keine weiteren Simulationen durchgeführt.
(iv) Das verwendete Simulations-Verfahren ist für die Visualisierung unerheblich und kann beliebig verändert werden, ohne das die Visualisierung davon betroffen ist.

Für die Nutzung zur Simulationsvisualisierung spielen fast alle Bestandteile einer Game-Engine nach Definition 1 eine wichtige Rolle. Jedoch werden Physik-Simulation und künstliche Intelligenz nicht benötigt (vgl. Definition 2iii). Netzwerk-Funktionalität wird nur zum Abrufen der Simulationsergebnisse benötigt. Jedoch nicht für Echtzeit-Übertragung, wie es in Multiplayer-Spielen notwendig ist, sondern in einem initialen Ladevorgang, da diese Daten zum Start bereits komplett vorliegen (vgl. Definition 2i).

3.2.6 Zwei- und dreidimensionale Simulationsvisualisierung

Zunächst ist es wichtig die Dimensionalität der Simulation von der Dimensionalität der Visualisierung zu unterscheiden. Ersteres ist die Grundlage für die Berechnung der Fußgängerdaten und beschreibt in welchen Ebenen beispielsweise Kollisionen berechnet werden. Die Dimensionalität der Visualisierung bezieht sich dann ausschließlich auf die Darstellung und ist unabhängig von der Dimensionalität der Simulation.

Obwohl es sich in der Realität natürlich immer um räumliche Szenarien handelt, werden Fußgänger-Simulationen oft nur zweidimensional ausgeführt. Dies vereinfacht die Komplexität der Modelle und Berechnung, ohne dabei Auswirkungen auf das Ergebnis der Simulation zu haben. Die räumliche Ausdehnung in der Höhe wird vernachlässigt und alle Objekte befinden sich auf der selben Ebene. Dies ist bei ebenerdigen Szenarien, bei denen Höhenunterscheide keine Rolle spielen, problemlos machbar. Hat ein Szenario beispielsweise mehrere Stockwerke können diese in getrennten Szenarien simuliert werden. Treppen die für den Personenzu- und -abfluss der Stockwerke wichtig sind können hierbei als Quellen und Ziele modelliert werden.

Unabhängig davon ob die Simulation zwei- oder dreidimensional durchgeführt wurde kann auch die Visualisierung in 2D oder 3D erfolgen. Bei der zweidimensionalen Darstellung kommt üblicherweise eine Draufsicht zum Einsatz, wie beispielsweise bei SumoViz (vgl. Kapitel 2.5.1). Gerade bei zweidimensional berechneten Simulationen lässt sich dies meist problemlos machen, da sich Objekte nie verdecken können. Trotzdem lassen sich zweidimensional simulierte Daten auch dreidimensional präsentieren. Dazu muss den flachen Fußgänger- und Geometrieobjekten eine Höhe zugewiesen werden. Liegen hierfür keine Daten vor, kommen vordefinierte Standardwerte zum Einsatz, die versuchen den Objekten realistische Höhen zuzuweisen. Da Geometrieobjekte in diesem Fall als zweidimensionale Kantenzüge oder Polygone vorliegen werden sie extrudiert. Für Fußgänger kann ein Standard-3D-Modell einer Person genutzt werden, wie es bei SumoViz3D gemacht wurde (vgl. Kapitel 2.5.2).

Diese Arbeit geht grundsätzlich von einer Darstellung auf 2D-Displays aus, da diese bei üblichen Computern defacto-Standard sind. Wenn von einer 3D-Visualisierungen gesprochen wird, ist dabei ein virtuell dreidimensional berechnetes Modell die Grundlage, das für die Darstellung auf ein zweidimensionales Bild projiziert wurde.

3.2.7 Mehrwert von 3D-Visualisierungen gegenüber zweidimensionaler Darstellung

Die Erzeugung von dreidimensionalen Visualisierungen ist erheblich aufwändiger und so stellt sich die Frage, ob eine 3D-Visualisierung für Fußgängersimulationen überhaupt sinnvoll ist. Bei den dargestellten Daten handelt es sich in der Regel um reale Gebäude oder Bereiche, die bereits existieren oder sich in der Planung befinden. Die physikalische Geometrie der Gebäude ist natürlich dreidimensional auch, wenn die Daten nur in 2D vorliegen. Deshalb ist die Visualisierung als dreidimensionale Szene, auf Grund des Blickwinkels, der Texturen und Licht- und Schattenverhältnisse optisch wesentlich näher an der Realität als eine zweidimensionale Draufsicht. Dies erleichtert Personen, die das reale Szenario kennen die Orientierung innerhalb der Visualisierung.

Für Personen die das Szenario nicht kennen, wird es leichter Größenverhältnisse abzuschätzen. Das 3D-Modell der Fußgänger gibt dem Betrachter einen Referenzpunkt, die Größe von Bauwerken in Relation zu den Personen und anderen Gebäuden zu setzen. Dies hilft dabei das Szenario besser zu verstehen und sich orientieren zu können. Beispielsweise bei der Schulung von Einsatzkräften für Veranstaltungen kann die dreidimensionale Visualisierung helfen eine gute Ortskenntnis zu vermitteln, ohne eine Anwesenheit vor Ort vorauszusetzen.

Räumliche Darstellung hat oft das Problem der Verdeckung. Dabei sind Objekte nicht sichtbar, weil sich ein anderes Objekt näher am Betrachter befindet und das dahinterliegende

Objekt überdeckt. Ein einfacher Lösungsansatz, der auch bei der 3D-Visualisierung gewählt wird, ist die Interaktivität. Wie in der Realität kann der Betrachtungswinkel und -standpunkt verändert werden und so verdeckte Objekte sichtbar machen. Dieser Effekt ("Parallaxenverschiebung") hilft auch bei der räumlichen Einordnung von Objekten und ist somit ein weiterer Vorteil der dreidimensionalen Darstellung. [ET08]

Ein weiterer Vorteil der 3D-Darstellung ist, dass der Raum auch für die Visualisierung weiterer Daten genutzt werden kann, die in einer zweidimensionalen Darstellung Unübersichtlichkeit verursacht hätten. So können mehr Informationen in der Visualisierung untergebracht werden. Beispielsweise könnte die Grundebene der Simulation entsprechend der Dichte eingefärbt werden und gleichzeitig die Fußgängerobjekte entsprechend ihrer Geschwindigkeit.

Die optisch deutlich ansprechendere 3D-Darstellung kann auch dazu genutzt werden um Bild- und Videomaterial zu exportieren. Dieses kann beispielsweise in den Medien verwendet werden, da es auch für Laien intuitiv und schnell verständlich ist.

Wie aufgezeigt bietet die interaktive 3D-Darstellung einig Vorteile gegenüber einer zweidimensionalen Darstellung, weshalb ihre Umsetzung sinnvoll ist und auch bisher schon Anwendung findet.

3.3 Verwandte Arbeiten

3.3.1 Nutzung einer Game-Engine zur Simulation

Wang et al. beschreiben 2003 [WLG03] die Nutzung der Unreal Engine für die Simulation von Robotereinsätzen in Katastrophenszenarien. Das U.S.-amerikanische *NIST* ("National Institute of Standards and Technology") betreibt ein Testgelände für die Erprobung von autonomen fahrbaren Robotern in Evakuierungs- und militärischen Szenarien. Allerdings ist die Nutzung dieses Testgeländes beschränkt und nicht öffentlich zugänglich.

In ihrem Paper bauen die Autoren deshalb das Testgelände als virtuelle Umgebung nach und können so Simulationen und Tests durchführen, ohne das Gelände vor Ort nutzen zu müssen. Die Engine stellt zwei Arten von Einheiten zur Verfügung: normale Spieler ("human playable") und computergesteuerte Einheiten ("Bots"), deren Verhalten über eine Netzwerk-Schnittstelle kontrolliert werden kann. Für die Simulation der Roboter wurden Bots verwendet, die über das Netzwerk die gleichen Anweisungen erhielten, wie die Roboter im realen Einsatz. Dabei kann es sich entweder um autonome Roboter (von einer Software gesteuert) oder um ferngesteuerte Roboter (von Menschen gesteuert) handeln. Die Unreal Engine bietet einen *Spectator*-Modus, in dem die virtuelle Kamera an eine beliebige Einheit gebunden werden kann und ein Bild aus dessen Egoperspektive liefert. Damit wird die Videoübertragung des Roboters simuliert.

Das Testgelände wurde detailliert als 3D-Modell nachgebaut, Fotos des original Geländes als Texturen genutzt und so eine realistisches virtuelles Modell erzeugt. Wichtiger Bestandteil der Simulation ist die Nutzung der Physik-Engine "Karma" um das Verhalten des virtuellen Roboters zu testen. Dabei werden jedem Objekt Werte für die Masse, den Masseschwerpunkt und Beschleunigung zugewiesen, anhand dieser die Physik Engine versucht das Verhalten des Objekts in der realen Welt nachzubilden.

"Die in diesem Paper beschriebene Simulation wurde in weniger als drei Monaten entwickelt und erfüllt alle Anforderungen", [WLG03] schreiben die Autoren abschließend. Die Nutzung der Unreal Game-Engine ermöglicht eine kostengünstige und gute Simulation, die in vielen Fällen die Nutzung des physikalischen Testgeländes ersetzen kann.

3.3.2 3D-Visualisierung von Städten

Kada et al. [KRW+03] untersuchten eine 3D-Visualisierung der Stadt Stuttgart zur Nutzung in der Städteplanung. Datenbasis war ein vorhandenes 3D-Modell der Stadt, Satellitenbilder, Grundrisspläne und Geländekarten für insgesamt 36000 Gebäude der Stadt. Zusätzlich wurden Fotos von einigen Gebäuden in der Innenstadt als Texturen verwendet. Der Nutzer soll sich sowohl aus der Perspektive eines Fußgängers durch die Stadt bewegen können, als auch frei über sie hinwegfliegen können. Die große Anzahl der zu visualisierenden Objekte ist die größte Herausforderung und das Paper beschreibt einige Techniken zur Verbesserung der Geschwindigkeit um eine flüssige Darstellung zu gewährleisten.

Es werden "Impostors" zur Verbesserung der Renderzeit implementiert. Bei dieser Technik werden Render-Ergebnisse als Grafik gespeichert und über den Zeitraum von mehreren Frames verwendet. So müssen die Modelle nicht für jeden Frame neu gerendert werden. Die Impostors werden entsprechend der Bewegung der Kamera transformiert. Diese Technik eignet sich für Objekte die sich bei einer Rotation um ihre senkrechte Mittelachse nur wenig unterscheiden und im Hintergrund der Szene liegen, so dass die Anwendung dieser Technik nicht störend auffällt. So konnte eine Verbesserung der Geschwindigkeit um 350% erreicht werden. Außerdem wurden die Modelle der Gebäude in der Anzahl ihrer Polygonpunkte so weit wie möglich reduziert.

Zur Detailreduktion des Höhenprofils wurde ein "continuous levels of detail"-Ansatz ("C-LOD") nach [HSS98] verwendet. Bei dieser Erweiterung des klassischen "level of detail" gibt es keine diskreten Level unterschiedlicher Auflösung, sondern dynamisch berechnete Reduktionen der Modelle. So werden Sprünge beim Verändern des Detailgrads vermieden. Zusätzlich wurde generell die Komplexität der Modellen auf die Grundformen der Gebäude reduziert.

Kapitel 4

Implementierung der Visualisierung auf Basis einer Game-Engine

In diesem Teil der Arbeit wird nun die Implementierung einer Visualisierung auf Basis einer Game-Engine beschrieben. Bezugnehmend auf die webbasierte Implementierung SumoViz3D aus meiner Bachelorarbeit, trägt die Neuimplementierung den Namen "SumoViz Unity". Dabei kommt die Unity3D-Engine zum Einsatz (vgl. Kapitel 3.2.4) und es sollen einige Erweiterungen umgesetzt werden (vgl. Kapitel 3.1). Die Implementierung kann dabei in die Bereiche Geometrie, Fußgängerobjekte, Steuerung sowie die Analysewerkzeuge untergliedert werden. In den folgenden Kapiteln wird die Struktur von SumoViz Unity werden wichtige Implementierungsdetails und -entscheidungen besprochen, der vollständige Quellcode liegt der Arbeit bei.

4.1 Import der Simulationsergebnisse

Die Simulationsergebnisse liegen als Daten der Form vor, wie sie in Kapitel 2.4.2 definiert wurden. Diese Daten werden jetzt von der Visualisierungssoftware genutzt um die Simulation grafisch darzustellen und die Simulationsergebnisse zu untersuchen.

Zwar ist der Inhalt der Simulationsegebnisse klar definiert, in welcher Form die Daten aber abgespeichert wurden variiert. Die Implementierung soll es ermöglichen Daten aus beliebigen Quellen zu verarbeiten und darzustellen. Je nach Anwendung liegen die Daten in unterschiedlichen Formaten und Speichersystemen vor. Die Speicherformate für die Simulationsergebnisse lassen sich in drei Kategorien unterteilen:

- **Plain text** Oft sind sowohl die Geometriedaten als auch die Fußgängerdaten in einfachen Textdateien hinterlegt, die leerzeichengetrennt Koordinaten enthalten. Dieses Format ist sehr einfach zu generieren und auch problemlos einzulesen. Auch der Austausch solcher Datensätze ist einfach möglich und mit einfachen String-Operationen können die Datensätze transformiert werden.
  Allerdings gibt es für dieses Format keinen fest definierten Aufbau und viele unterschiedliche Implementierungen sind denkbar. Es ist schwierig Beziehungen zwischen Datensätzen darzustellen oder komplexere Datenstrukturen abzubilden. Komplexere

Anfragen können gegen solche Textdateien nicht gestellt werden. Dafür müssten die Daten erst ein anderes Format überführt werden. Ebenso sind Daten im plain-text-Format auch nicht für größere Datensätze geeignet. Bei großen Datensätzen können die Textdateien schnell mehrere hundert Megabyte annehmen und dann nicht mehr effizient abgefragt und bearbeitet werden.
Die Ausgabedateien, wie sie in SumoViz3D verwendet wurden, lagen im plain-text-Format vor. [Büc12]

- **JSON und XML** Strukturierte Serialisierungen von komplexeren Datenstrukturen können mit Formaten wie JSON und XML genutzt werden. Die Formate sind klar definiert und es gibt viele Editoren und Werkzeuge um mit diesen Datenformaten zu arbeiten. Für Unity3D und nahezu jede andere Programmiersprache gibt es Parser zum Einlesen der Dateien und Serializer zum Schreiben der Formate.
  Wie bei einem plain-text-Format können sehr große Datensätze und Anfragen darauf ein Problem für diese Formate werden. Komplexere Datenstrukturen sind mit diesen Formaten aber problemlos abbildbar. In SumoViz3D werden die Anfragen an die Datenbank im JSON-Format an die Applikation übergegen.

- **Datenbanksysteme** sind die aufwändigste Methode der vorgestellten Speicherverfahren, ermöglichen aber auch komplexe Anfragen und Transformationen auf den Datensätzen. Zur Auswertung und Analyse sind Datenbanksysteme daher gut geeigent.
  Geospatial databases mit einer speziellen Unterstützung für raumzeitliche Daten optimieren Anfragen mit Hilfe von räumlichen Datenstrukruten und können damit für große Datenmengen effizient Resultate liefern.
  SumoViz3D verwendet die dokumentenorientierte Datenbank CouchDB zur Transformation der Daten. CouchDB kann die Daten effizient mit MapReduce-Funktionen transformieren und liefert sie dann im JSON-Format aus.

Geometrie- und Fußgängerdaten sowie weitere Zusatzinformationen, wie beispielsweise Gruppendaten, können unabhängig von einander in unterschiedlichen Formaten vorliegen und können auch aus unterschiedlichen Quellen bezogen werden.

4.1.1 Beispielimplementierung eines Imports für Textdateien

In der Implementierung zu dieser Arbeit werden die Daten beispielhaft aus einer Textdatei geladen. SumoViz Unity definiert eine Schnittstelle für die Übergabe der Fußgänger- und Geometriedaten an die Visualisierungssoftware. Diese Schnittstelle kann dann von beliebigen Importern genutzt werden können. So kann beispielsweise auch ein Importer geschrieben werden, der die Daten direkt aus einer Datenbank abfragt, ohne dass die Visualisierung an sich verändert werden muss.

Die Beispielimplementierung nimmt den einfachsten Fall von Leerzeichen-getrennten plain-text-Dateien an, wie sie in Abbildung 4.2 zu dargestellt werden. Diese liegen lokal auf dem Dateisystem vor und die Visualisierungssoftware hat direkten Zugriff auf diese Dateien. Die Implementierung befindet sich in der Klasse `FileLoader`. Diese Klasse ist sowohl für den Import von Geometrie- als auch Fußgängerdaten zuständig und zeigt exemplarisch die Verwendung des `GeometryLoader` und `PedestrianLoader`. Diese beiden Klassen sind für die Erstellung und Verwaltung der entsprechenden Objekte zuständig. Während der `FileLoader` für Daten aus

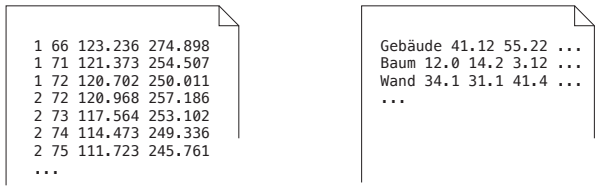


Abbildung 4.1: Beispielhafte Darstellung von plain-text-Daten für Fußgängerpositionen (links) und Geometriedaten (rechts)

einer anderen Quelle neu implementiert werden muss, sind `GeometryLoader` und `PedestrianLoader` fester Bestandteil der Visualisierung, der nicht verändert werden muss. Sie sind die beiden Klassen mit denen der jeweilige Importer kommuniziert um die Fußgänger- und Geometrie-Objekte anzulegen.

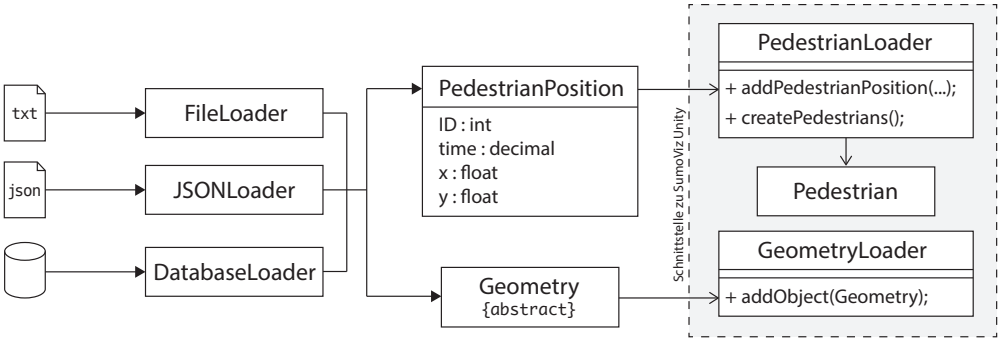


Abbildung 4.2: Schematische Darstellung der Übergabe von Fußgänger- und Geometriedaten an die Visualisierungssoftware

GeometryLoader

Die Geometriedaten, wie in Kapitel 2.4.2 definiert, werden einmalig zum Beginn der Visualisierung geladen und bleiben über den zeitlichen Verlauf unverändert. Die Klasse `GeometryLoader` ist dafür zuständig die Geometrieobjekte zu verwalten und zur Szene hinzuzufügen. Sie stellt die Schnittstelle dar, die die Importer nutzen um ihre Daten an die Visualisierung zu übergeben. Dazu bietet die Klasse eine öffentliche Methode `addObject(Geometry object)`, über die die Daten an die Klasse übergeben werden (vgl. Abbildung 4.2). Alle Geometrieobjekte erben von der abstrakten Klasse `Geometry`. Der `FileLoader` instanziiert das Geometrieobjekt und übergibt es an den `GeometryLoader`.

PedestrianLoader

Analog zum `GeometryLoader` dient der `PedestrianLoader` als Schnittstelle zur Übergabe der Fußgängerdaten an SumoViz Unity. Wie in Kapitel 2.4.2 definiert liegen die Fußgängerdaten zeitdiskretisiert vor. Für die Visualisierung wird dann zwischen den diskreten Zeitschritten linear interpoliert.

Das Layout der Schnittstelle wurde darauf optimiert, dass die Daten im Format $p(\mathrm{ID}, t) \rightarrow (x, y)$ vorliegen. Die logische Verknüpfung der einzelnen Datensätze zu einer Trajektorie eines Fußgängers erfolgt über die ID. Deshalb ist der Aufruf der Schnittstelle so gestaltet, dass die vier Parameter (ID, Zeit, x- und y-Koordinate) übergeben werden können, ohne dass einzelne Datenpunkte bereits zu Trajektorien verknüpft werden müssen. Die vier Parameter werden dafür noch in einem Objekt vom Typ `PedestrianPosition` gekapselt. Listing 4.1 zeigt einen beispielhaften Aufruf, der Datensätze an den `PedestrianLoader` überträgt.

Listing 4.1: Anlegen der Fußgängerobjekte

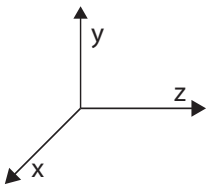
```
foreach (data in file) {                                                    1
    PedestrianPosition p =                                                  2
        new PedestrianPosition(data[id], data[time], data[x], data[y]);
    pedestrianLoader.addPedestrianPosition(p);                              3
}                                                                           4
pedestrianLoader.createPedestrians();                                       5
```

Die Daten können über diesen Aufruf in beliebiger Reihenfolge an SumoViz Unity übergeben werden ohne, dass dabei eine chronologische Sortierung oder Gruppierung nach IDs notwendig ist. Einzelne Fußgänger werden impliziert dadurch erstellt, dass mindestens ein Datensatz mit einer entsprechenden ID an den `PedestrianLoader` übergeben wird.

Anders als bei den Geometrieobjekten werden diese allerdings nicht direkt bei der Übergabe der Daten erstellt. Erst wenn alle Fußgängerdaten an den `PedestrianLoader` übergeben wurden, werden durch den Aufruf der Funktion `createPedestrians()`, die Fußgänger und deren Trajektorien angelegt und können anschließend nicht mehr verändert werden. Die Funktion `createPedestrians()` erstellt bei ihrem Aufruf eine Instanz der `Pedestrian`-Klasse für jede ID. Diesem Objekt werden dann alle Trajketorienpunkte mit der entsprechenden ID zugeordnet und in einer chronologisch sortierten Liste gespeichert (vgl. Abbildung 4.2).

4.2 Darstellung der Geometrie

Unity3D verwendet ein linkshändiges kartesische Koordinatensystem in dem die x- und z-Achse die horizontale Grundebene bilden und die y-Achse die Höhe über dieser Grundebene angibt. Diese Konvention wird auch im Folgenden verwendet.



GameObjects in Unity3D

In Unity3D sind alle Objekte die Bestandteil einer Szene sind ein einem Szenegraph organisiert und erben vom Typ `GameObject`. Wurzel des Szenegraph ist eine Szene. In der Spieleentwicklung entspricht dies meist einem Level oder einer zusammenhängenden Welt. Die Implementierung von SumoViz Unity verwendet nur eine einzige Szene für die Visualisierung. Die GameObjects sind Kinder dieser Wurzel und können in beliebigen Hierachien verzweigt werden. Ein GameObject ist erstmal nur ein leerer Container dem wiederum einzelnen Komponenten zugeordnet werden können, die seine Eigenschaften definieren. Einige dieser Komponenten, die auch für die Erstellung der Geometrieobjekte von Bedeutung sind, werden im Folgenden beschrieben:

- Die **Mesh**-Komponente ("mesh filter") beinhaltet das Drahtgittermodell (engl. "wireframe") des dargestellten Objekts. Das Mesh wiederum besteht aus einzelnen Knotenpunkten ("vertices") die jeweils zu Dreiecken gruppiert werden und so einzelne Flächen in einem Dreiecksnetz erzeugen. Jeder komplexere Körper wird aus einer beliebigen Anzahl dieser Dreiecke zusammen gesetzt. Um die Kosten gering zu halten muss die Anzahl der Vertices niedrig gehalten werden.
- Die **Material**-Komponente definiert wie das Mesh dargestellt wird. Die Farbe, Transparenz, Licht- und Schattenverhalten und grafische Texturen werden in dieser Komponente definiert.
- Die **Script**-Komponente kann einem GameObject ein Skript zuweisen, dass dessen Erscheinung steuert. Diese Skripte können in einer `Start()`-Methode die einmalige Initialisierung des GameObjects vornehmen. Die `Update()`-Methode wird zu jedem Frame aufgerufen und kann so das Verhalten des GameObjects in Echtzeit anpassen.
- Die **Transform**-Komoponente verwendet dreidimensionale Vektoren um Translation, Rotation und Skalierung entlang der drei Achsen zu beschreiben.

Erzeugen der Geometrieobjekte

Die Geometrieobjekte sind in den Ausgangsdaten der Visualisierung definiert und müssen dann als 3D-Objekte erzeugt und dargestellt werden. Welche verschiedenen Geometrie-Typen es gibt hängt einerseits davon ab, wie die Objekte in den Geometriedaten unterschieden werden und andererseits davon, welche Klassen für Geometrieobjekte implementiert und erzeugt wurden. SumoViz Unity definiert eine abstrakte Klasse `Geometry`, von der alle Geometrieobjekte erben müssen. Es können beliebige Geometrie-Klassen angelegt und verwendet werden. Damit ist SumoViz Unity in seinen Möglichkeiten der Darstellung nicht auf eine feste Menge verschiedner Geometrieobjekte festgelegt, sondern es können beliebige neue Objekte definiert werden. SumoViz Unity bringt eine Reihe von Geometrieobjekten mit, die genutzt werden können.

- `AreaGeometry` – Flacher Bereich auf der Grundfläche, der farblich hervorgehoben wird.
- `TreeGeometry` – 3D-Modelle eines Baums.
- `ExtrudeGeometry` – Abstrakte Elternklasse für Objekte die extrudiert werden sollen.
- `ObstacleExtrudeGeometry` – Hindernisse, die in der Höhe extrudiert werden. Ab einer Höhe von 4m wird das Objekt als Gebäude dargestellt.
- `WallExtrudeGeometry` – Wände, die aus einem nicht geschlossenen Kantenzug bestehen und in der Breite und Höhe extrudiert werden.

Für jedes Geometrieobjekt erzeugt der `GeometryLoader` ein GameObject und fügt diesem die benötigten Komponenten hinzu. Der schematische Ablauf bei der Erzeugung eines Geometrieobjekts ist wie folgt:

1. Zerlegung des Polygons in Dreiecke
2. Extrudieren des Polygons (falls `ExtrudeGeometry`)
3. Erzeugen des Dreiecknetzs
4. Anwenden der Textur

Zerlegung in Dreiecke Wie beschrieben, bestehen alle Modelle aus Dreiecksnetzen. Aus der Menge der gegebenen Koordinaten eines Geometrieobjekts müssen nun ebenfalls Dreiecke

für das Mesh bestimmt werden. Dieser Vorgang wird als *Triangulation* bezeichnet. Für dieses Beispiel wird angenommen, dass es sich um zweidimensionale Koordinaten handelt die einen geschlossenen Kantenzug darstellen. Im geometrischen Sinn kann dann von einem Polygon mit n-Ecken gesprochen werden. [Vat92]

Polygone lassen sich in vier verschiedene Kategorien einteilen (vgl. Abbildung 4.3), die unterschiedliche Eigentschaften für die Triangulation mitbringen. Bei konvexen Polygonen ist der Innenwinkel eines Eckpunkts immer kleiner als der Außenwinkel und somit können einfach von einem Punkt Diagonalen zu allen anderen gezogen werden. Bei konkaven Polygonen ist dies nicht gegeben und es muss bei der Triangulation darauf geachtet werden, dass ein erzeugtes Dreieck nicht den Bereich überspannt, der außerhalb des Polygons liegt (vgl. Abbildung 4.3b, roter Bereich). Überschlagene Polygone und Polygone mit Loch können in zwei einzelne Polygone zerlegt werden und dann mit den selben Algorithmen verarbeitet werden. Deshalb werden sie hier nicht gesondert behandelt. [Vat92]

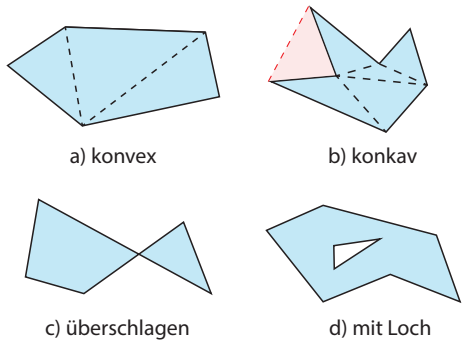


Abbildung 4.3: Kategorisierung von Polygonen [Vat92]

Zur Triangulation gibt es verschiedene Algorithmen mit unterschiedlichen Komplexitätseigenschaften. Da die Erzeugung der Geometrie nur einmalig stattfindet und die Anzahl der zu zerlegenden Polygone üblicherweise auch nicht sehr hoch ist, ist kein hocheffizienter und komplexer Algorithmus notwendig. Die Implementierung verwendet daher den weitverbreiteten "ear clipping"-Algorithmus. [Ebe98]

Beim "ear clipping" werden keine zusätzlichen Vertices erzeugt, sondern nur zwischen den Eckpunkten des Polygons Diagonalen eingezogen. Somit lassen sich alle Polygone in $n - 2$ Dreiecke zerlegen. Der Algorithmus sucht nach Dreiecken, in denen keine anderen Punkte des Polygons liegen. Diese sind sogenannte Ohren und werden als einzelnes Dreieck vom Polygon abgetrennt. Das restliche Polygon wird mit dem gleichen Verfahren weiter zerlegt bis nur noch 3 Polygonpunkte übrig sind, die dann das letztes Dreieck bilden. [Ebe98]

Listing 4.2: Trianglation mittels ear clipping

```
while (Polygon.vertices.count > 3) {
    for (Point p in Polygon) {
        Triangle t = (p.prev(), p, p.next())
        if (t.isEmpty()) {
            Polygon.removePoint(p);
            Triangles.add(t);
        }
    }
}
```

Eine naive Implementierung des Algorithmus wie in Listing 4.2 zu sehen führt zu einer Laufzeit von $O(n^3)$, denn die `while`-Schilfe in Zeile 1, die `for`-Schleife in Zeile 2 und die Funktion `t.isEmpty()` (Zeile 4) haben jeweils eine lineare Komplexität. Jedoch ist mit einer aufwändigeren Implementierung wie sie von David Eberly in "Triangulation by Ear Clipping" beschrieben wird eine quadratische Laufzeit zu erreichen. [Ebe98]

Extrudieren des Polygons Handelt es sich um ein Objekt das extrudiert werden soll, wird ein zweites Polygon berechnet, das die selben Koordinaten wie das ursprüngliche Polygon hat, jedoch entlang der senkrechten Achse verschoben ist. Diese beiden parallelen Polygone bilden somit den "Boden" und das "Dach" des Objekts. Dazwischen müssen nun noch die Wände erzeugt werden. Eine n-eckige Grundfläche benötigt auch n Seitenwände, wobei jede Seitenwand wiederum aus zwei Dreiecken besteht. Dazu wird über alle Eckpunkte des Dach- und Boden-Polygons iteriert und jeweils aus den aktuellen Eckpunkten sowie den des nächsten Nachbarn ein Dreieck gebildet (vgl. Listing 4.3 Zeile 3 und 4).

Listing 4.3: Algorithmus zur Erzeugung der Seitenwände beim Extrudieren

```
for (i in n) {
    int j = (i+1)%n;
    mesh.add(new Triangle(P_dach[i],P_boden[i],P_dach[j]));
    mesh.add(new Triangle(P_boden[i],P_boden[j],P_dach[j]));
}
```

Erzeugen des Dreiecknetzes Die berechneten Dreiecke der Polygon-Zerlegung sowie ggf. die der Seitenwände für extrudiere Objekte werden nun verwendet um ein Dreiecksnetz ("mesh") zu erzeugen. Dieses besteht im Wesentlichen aus einer Liste aller Vertices (wobei mehrfach verwendete Vertices trotzdem nur einmal zum Mesh hinzugefügt werden) und einer Zuordnung, welche drei Vertices jeweils ein Dreieck bilden.

Bei der Zuordnung der Vertices zu den Dreiecken spielt die Reihenfolge der Zuordnung eine wichtige Rolle. Jedes Dreieck erzeugt einen Normalenvektor, der senkrecht auf dem Dreieck steht. Dieser Vektor wird für die Berechnung der Farbe, des Lichts und des Schattens verwendet. Beispielsweise kann mit Normalenvektor und Blickrichtung der Kamera der Blickwinkel und damit der Ausfallswinkel des Lichts berechnet werden. Dies hat aber auch zur Folge, dass Dreiecke deren Normalenvektor von der Kamera wegzeigt nicht dargestellt werden. Die Dreiecke haben also keine Rückseite und sind somit nur von vorne sichtbar.

In welche Richtung der Normalenvektor zeigt wird durch die Reihenfolge der Vertices des Dreiecks festgelegt. Diese werden im mathematischen Drehsinn angegeben, der Normalenvektor zeigt dann nach oben. Deshalb müssen alle Dreiecke so erzeugt werden, dass für Dächer und Flächen der Normalenvektor nach oben zeigt und für die Seitenwände nach außen. Dies hat zugleich den Effekt, dass Objekte von unten nicht sichtbar sind. Befindet sich die Kamera innerhalb eines Objekts sind auch die Seitenwände nicht zusehen. Oft wird in der 3D-Grafik versucht diesen Effekt zu verhindern, in dem die Dreiecke beispielsweise doppelt angelegt werden, oder spezielle doppelseitige Shader verwendet werden. [Wat93]

Das Dach- und Boden-Polygon sowie die $n - 2$ Seitenwände werden dann zu einem Mesh zusammengefügt und erzeugen das extrudierte Objekt.

Anwendern der Texturen Für die Darstellung muss dann noch ein Material auf das Mesh angewendet werden. Grafische Texturen stellen eine Bilddatei auf dem 3D-Objekt dar. Dazu wird an jedem Vertex definiert welche Stelle der Grafik dargestellt wird. Dafür wird ein UV-Vektor pro Vertex definiert, der die Position innerhalb der Grafik bezeichnet, die am entsprechend Vertex dargestellt wird. [Wat93] Der ausgewählte Darstellungsmodus (vgl. Kapitel 4.2.1) entscheidet welche Grafik für welches Geometrieobjekt geladen wird.

4.2.1 Darstellungsmodi der Geometriedaten

Um die Visualisierung an verschiedene Szenarien anpassbar zu gestalten, wurde auch die Darstellung modular entworfen. So kann ein vorhandener Darstellungsmodus ausgewählt, oder falls kein passender vorhanden ist ein neuer entwickelt werden, der die Darstellung der Szene passend umsetzt.

Ein Darstellungsmodus definiert die Szenerie in der die Visualisierung stattfindet und das Aussehen von Objekten innerhalb der Szene. Denkbar sind beispielsweise unterschiedliche Darstellungsmodi für Szenarien in der Stadt, in der Natur, Indoor, auf Konzert- und Festivalflächen, etc. (vgl. Abbildung 4.4).

Abbildung 4.4: Darstellung der selben Szene in zwei unterschiedlichen Darstellungsmodi

Zur Erstellung eines Darstellungsmodus muss nur eine Klasse implementiert werden, die von der abstrakten Elternklasse `ThemingMode` erbt und deren abstrakte Methoden implementiert. Zusätzlich muss noch ein Terrain in Unity angelegt werden, dass die Darstellung der Grundfläche und Umgebung definiert. Die Implementierung des Darstellungsmodus gibt für bestimmte Geometrietypen passende Materialien zurück. So muss beispielsweise eine Methode `getWallsMaterial()` implementiert werden, die das Material für die Darstellung von Wänden zurück gibt.

In der Implementierung von SumoViz Unity sind bereits zwei verschiedene Darstellungsmodi angelegt. `NatureThemingMode` und `CityThemingMode` definieren unterschiedliche Darstellungen in einer Stadt und in der freien Natur. Der gewünschte Darstellungsmodus wird als Objekt erzeugt und dem `GeometryLoader` übergeben:

```
geometryLoader.setTheme (new CityThemingMode ());
```

4.3 Steuerung des Visualisierung

4.3.1 Wiedergabe der Simulationsdaten

Die Dauer der Visualisierung ist solange wie Fußgängerdaten vorliegen. Damit definiert sich die Länge der Visualisierung `total_time` durch den spätesten Trajektorienpunkt. Dieser Zeitpunkt wird direkt beim Einlesen der Fußgängerdaten bestimmt. Den aktuellen Zeitpunkt der

Visualiserung gibt `current_time` an und wird mit 0 initialisiert. Ein Wahrheitswert `playing` gibt an, ob die Visualisierung läuft oder angehalten ist.

Während die Visualisierung abgespielt wird, wird bei der Berechnung jedes Frames die Zeitdifferenz zum vorherigen Frame auf die `current_time` addiert. Zu beachten ist hierbei, dass mit hoher Wahrscheinlichkeit nicht die exakten Zeitpunkte der Trajektorien ausgewählt werden sondern immer zwischen zwei Trajektorienpunkten interpoliert werden muss (vgl. Kapitel 4.4.2).

Die einzelnen Trajektorienpunkte eines Fußgängers sind in einer sortierten Liste chronologisch gespeichert. Ausgehend vom aktuellen Trajektorienpunkt könnte somit der nächste Trajektorienpunkt in konstanter Zeit bestimmt werden. Dazu kann einfach der folgende Eintrag der Liste genutzt werden. Jedoch hat der Nutzer die Möglichkeit die Abspielposition der Visualisierung zu jedem Zeitpunkt beliebig zu verändern. Da ich der aktuelle Zeitpunkt der Darstellung jederzeit beliebig verändern kann, kann nicht einfach der chronologisch folgende Trajektorienpunkt ausgewählt werden. Der darzustellende Trajektorienpunkt muss mit dem entsprechenden Zeitpunkt als Anfrageprädikat innerhalb der sortierten Liste gesucht werden (vgl. Kapitel 4.4.2).

4.3.2 Virtuelle Kamera

Die virtuelle Kamera entscheidet welcher Ausscchnitt der 3D-Szene sichtbar ist. Die Kamera ist, wie alle anderen Objekte auch als GameObject im Szenegraph angelegt und kann ebenso manipuliert werden. Sie hat eine Position im dreidimensionalen Raum und einen Rotationsvektor der die Neigung der Kamera um die jeweiligen Achsen beschreibt.

Bei einer perspektivischen Projektion, wie sie auch in der Realität vorkommt, wird der Bildausschnitt duruch eine Pyramide definiert. Die Spitze liegt an der Position der Kamera. Wie spitz die Pyramide ist, wird über das "field of view" definiert, das der Brennweite physikalischen Kameras entspricht. Werte für "near plane" und "far plane" definieren eine minimale und eine maximale Entfernung, innerhalb der Objekte lieben müssen um darstellt zu werden. Damit wird die Pyramide auf einen Pyramidenstumpf begrenzt, der als "frustum" bezeichnet wird. Alle Objekte die mindestens teilweise innerhalb des frustums liegen werden für die Berechnung des finalen Bilds genutzt. Alle anderen Objekte werden für die Berechnung des aktuellen Frames ignoriert ("frustum culling"). [Wat93]

Bewegung der Kamera

Der Betrachter kann sich mit der virtuellen Kamera frei durch die Szene bewegen. Dabei können Position und Rotation der Kamera beliebig verändert werden.

Die Position der Kamera ist ausschlaggebend für die Berechnung der Darstellung und muss daher als erster Schritt bei der Berechnung des aktuellen Frames ausgeführt werden. Dabei wird geprüft ob eine oder mehrere Tasten zur Translation der Kameraposition gedrückt sind. Zusätzlich wird der Distanzverktor der Mausbewegung seit dem letzten Frame berechnet. Die Veränderung der Mausposition wird in die Rotation der Kamera übersetzt, gedrückte Tasten in die Positionsveränderung der Kamera.

`W`, `A`, `S` und `D` verschiebt die Kamera nach vorne, hinten, links und rechts. Die Verschiebung hängt dabei von der Blickrichtung der Kamera ab. Blickt die Kamera beispielsweise gerade nach unten auf die Grundfläche, so entspricht eine Bewegung nach vorne einer Bewegung in Richtung der Grundebene. Zusätzlich kann die Kamera mit den Tasten `Q` und `E` senkrecht zur Blickrichtung nach oben bzw. unten bewegt werden.

Verwendung einer Skybox

Je nach Ausrichtung der Kamera ist nicht nur die Grundfläche mit den Geometrieobjekten zu sehen, sondern auch der Horizont und damit der Blick ins Unendliche. Um die Darstellung realistischer zu gestalten sollte über dem Horizont ein Himmel dargestellt werden. Damit wirkt die Szene fotorealistischer und größer. Videospiele verwenden für die Darstellung eines Himmels eine "Skybox". Die Kamera befindet sich hierbei im Zentrum eines Würfels der die gesamte Szene umspannt. Dieser Würfel bewegt sich bei der Translation der Kamera mit dieser mit, bleibt jedoch bei der Rotation der Kamera statisch. So kann die Kamera die Skybox nicht verlassen und durch die gleichbleibende Entfernung wirkt die Skybox in unendlicher Entfernung. [Gau05]

Auf die Innenseiten des Würfels, also die sichtbaren Flächen der Skybox, wird eine grafische Textur angewendet. Insgesamt werden sechs verschiedene Grafiken für vorne, links, hinten, rechts, oben und unten benötigt. Bei der Erstellung der Grafiken muss darauf geachtet werden, dass der Übergang an den Kanten nicht sichtbar ist. [Gau05]

Im Gegensatz zur Berechnung des Himmels mittles 3D-Objekten für Wolken, Sonne, etc. ist die Verwendung einer Skybox wesentlich ressourcenschonender. Die Darstellung von Objekten in unendlicher Entfernung wäre außerdem problematisch, da die Tiefenauflösung ("depth buffer") gegebenenfalls nicht hoch genug ist. Dies kann zur Folge haben, dass durch Rundungsfehler Objekte hinter dem Himmel verschwinden würden. [Gau05]

4.4 Darstellung der Fußgänger

4.4.1 Lokomotionsebene der Fußgänger

Aufgabe der Post-Visualisierung ist die Darstellung der lokomotorischen Bewegungen (vgl. Kapitel 2.1.4) der Fußgänger. Dazu ist es zunächst wichtig den menschlichen Gang und seine Eigenschaften zu betrachten um diese dann in der Visualisierung umsetzen zu können.

Ganganalyse

Gehen besteht zunächst aus einer Abfolge von Schritten. Ein Einzelschritt bezeichnet den Abschnitt vom Aufsetzen der Ferse des einen Beins, über das Schwingen und Aufsetzen des anderen Fußes. Einzelschritte definieren ein Stand- und ein Schwungbein. Ein Doppelschritt bezeichnet zwei auf einander folgende Einzelschritte, so dass die Anfangs- und die Endposition identisch sind. Der menschliche Gang ist ein sich wiederholender zyklischer Bewegungsablauf bestehend aus Doppelschritten ("Gangzyklus"). [Run98]

Beim Gehen ist, anders als beim Laufen, immer mindestens ein Bein auf dem Boden und es gibt einen Moment im Gangzyklus in dem beide Beine Bodenkontakt haben. [Run98] Die Geschwindigkeit des Gehens kann über zwei Parameter gesteuert werden: Die Schrittlänge l_s [m] und die Schrittfrquenz $v_{F,h}$ [m/s], bzw. deren Kehrwert, die Schrittdauer. Weidmann stellt dazu folgenden linearen Zusammenhang fest: $l_s = 0,235\text{m} + 0,302 \cdot v_{F,h}$ [Wei93]

Die durchschnittlichen und maximale Schrittlänge und -frequenz ist abhängig von Faktoren wie dem Geschlecht, Bewegungszweck, Alter, Größe, etc. Als Durchschnittswert verschiedener Literaturangaben nennt Weidmann $1,34\text{m/s}$ $(4,83\text{km/h})$. Als Richtwert für die Schrittlänge gibt er $0,65\text{m}$ bei einer Frequenz von $2,05\text{Hz}$ an. [Wei93]

Animation des Gangzyklus

Jeder Fußgänger wird mit einem 3D-Modell einer Person dargestellt und hat eine zugehörige Bewegungstrajektorie. Das Modell wird über den Ablauf der Visualisierung hinweg entlang dieser Trajektorie bewegt. Ohne die Animation des Gangzyklus würde das Modell bewegungslos über die Grundfläche gleiten. Deshalb wird das Modell mit einer Animation versehen, die den Gangzyklus des Fußgängers animiert.

Animation können in Unity3D als sogenannte Keyframe-Animationen umgesetzt werden. Hierfür wird der Zustand des 3D-Modells zu bestimmten Zeitpunkten ("Keyframes") definiert. Die Zustände zwischen den Keyframes werden interpoliert. [CC00] Zur Interpolation werden quadratische oder kubische Algorithmen verwendet, damit die sich die Bewegungsrichtungen nicht abrupt ändern. Wir sehen menschliche Bewegungen jeden Tag und kennen den natürlichen Bewegungsablauf daher sehr gut. Auch kleine Fehler im animierten Bewegungsablauf fallen oft schon auf und wirken unrealistisch. Dabei lässt sich der Fehler in vielen Fällen gar nicht konkret benennen. [MFCGD99]

Als Fußgängermodell wird in SumoViz Unity ein relativ komplexes 3D-Modell mit über 3000 Vertices verwendet. Damit nicht alle 3000 Vertices einzeln animiert werden müssen, werden Skelette verwendet. Das Modell verfügt über ein einfaches unsichtbares Skelett aus einzelnen Gelenkpunkten (engl. "joints") die hierarchisch organisiert sind. Beispielsweise ist das Kniegelenk ein Kind des Hüftgelenks und das Fußgelenk wiederum ein Kind des Knies. Eine Veränderung des Hüftgelenks bewegt also auch das Knie und den Fuß mit. Die Vertices des 3D-Modells werden als "skin" bezeichnet und werden dann den Gelenkpunkten zugeordnet und mit diesen mitbewegt. In der Animation werden dann nur die Skelettpunkte animiert und das Modell bewegt sich entsprechend mit. Für die Animation von menschlichen Modellen wird das Skelett oft ähnlich zum biologischem Skelett modelliert und es gibt für jedes menschliche Gelenk eine Entsprechung.[MMG06]

Um möglichst realistische Gehbewegungen zu erzeugen können motion-tracking-Technologien eingesetzt werden, die echte Menschen mit mehreren Kameras erfassen und die Bewegungen auf das virtuelle Skelett übertragen. [MFCGD99]

Aus dem Gangzyklus wie in Abbildung 4.5 zu sehen, werden einzelne Keyframes generiert, die dann auf dem virtuellen Skelett umgesetzt werden. Im Folgenden werden die Phasen eines Doppelschritts, wie in Abbildung 4.5 gezeigt, beschrieben: [Heg00]

1. Ausgehend vom vorhergehenden Schritt berührt das Schwungbein mit der Ferse zu erst den Boden und leitet die Standphase (auch als "Zweibeinstand" bezeichnet), in der

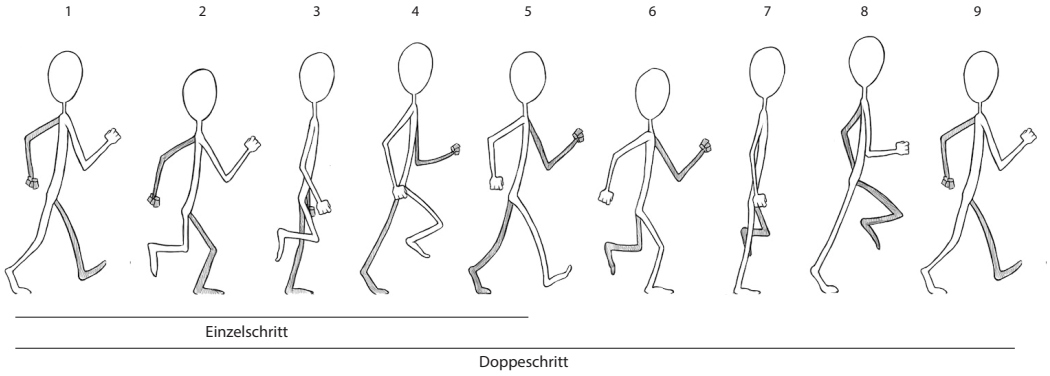


Abbildung 4.5: Ablauf eines Doppelschritts in neun Keyframes [Eve12]

beide Beine Bodenkontakt haben, ein. Das Schwungbein bleibt weitestgehend gestreckt und das Knie federt das Auftreten nur leicht ab. Der gesamte Körper verlagert nun den Schwerpunkt nach vorne und verlagert so das Gewicht auf das andere Bein. Diese Phase ist mit Abstand die längste und benötigt etwa 60 % der Schrittdauer während die Phasen 2-4 jeweils etwa 13 % der Schrittbauer in Anspruch nehmen. [BFL00]

2. Das Gewicht ist komplett auf das linke Bein verlagert, während das rechte Bein vom Knie her abgewinkelt wird und so in Schwungphase überleitet. Das Schwungbein befindet sich dabei hinter dem Schwerpunkt des Körpers. [NTH85] Der gegenüberliegende ("kontralaterale") Arm schwingt parallel mit dem Bein nach vorne und der Ellbogen ist dabei gestreckt.
3. Mit Hilfe der Hüftmuskulatur wird das angewinkelte Schwungbein nach vorne bewegt. Dabei schwingt der Fuß beim entspannten Gang nur wenige Millimeter über dem Boden hinweg. Das Sprunggelenk ist angespannt, damit die Fußspitze nicht den Boden berührt. Der kontralaterale Arm beugt sich am Ellbogen. [BFL00]
4. Das Beenden der Schwungphase wird eingeleitet. Das Bein wird durchgestreckt und das Knie angespannt und für das Auftreffen stabilisiert. Das Spunggelenk wird wieder entspannt und der Fuß geht zurück in Normalposition. [NTH85] [BFL00]
5. Es ist wieder eine Standphase erreicht. Der kontralaterale Arm schwingt in die Ruheposition zurück. Die Haltung ist die selbe wie in Schritt 1, allerdings ist nun das rechte Bein vorne. Der Einzelschrittzyklus ist an dieser Stelle abgeschlossen. [BFL00]

5.-9. Wiederholung des Einzelschritts wie in den Schritten 1 - 5, jedoch gespiegelt mit dem linken Bein als Schwungbein.

Es ist zu erkennen, dass sich Kopf und Hüfte relativ zum Boden sinusförmig auf und ab bewegen. Die Armbewegungen variiren mit der Geschwindigkeit. Bei hohen Geschwindigkeiten sind sie wesentlich ausgeprägter. Auch der Oberkörper ist nicht immer gerade, sondern lehnt bei der Gewichtsverlagerung auf das andere Bein nach vorne. [NTH85]

Die Geschwindigkeit der Animation des Gangzyklus wird über den zeitlichen Abstand der Keyframes definiert. Für SumoViz Unity wurden die Keyframes exakt so gewählt wie in Abbildung 4.5 zu sehen. Die Informationen zum zeitlichen Ablauf der Keyframes wurden aus [Heg00] entnommen. Als Schrittdauer für die Animation wurde die durchschnittliche Schrittdauer nach Weidmann verwendet. Es wird eine Schrittfrequenz von $2,06$Hz angegeben, was einer Schrittdauer von $0,49$s für einen Einzelschritt entspricht. Selbiges gilt für die Schrittlänge von $0,63$m. [Wei93].

Daraus ergibt sich eine durchschnittliche Geschwindigkeit von $1,32\text{m/s}$ [Wei93], mit der die Animation bei einfacher Geschwindigkeit abgespielt wird.

Schrittverhalten in Menschenmassen

Jelic et al. [JARLP12] untersuchten 2012 das Gehverhalten von Fußgängern in eindimensionaler Richtung. Zunächst stellten sie fest, dass es erhebliche Unterschiede im menschlechen Gang bei freim Fluss oder in einer Menschenmasse gibt. Dazu machten sie einen Experimentaufbau, bei dem Fußgänger hintereinander in einem Kreis laufen mussten. Ihre Ergebnisse beziehen sich daher auf das Verhalten in Menschenmassen. Radius und Dichte wurden dabei über verschiedene Experimente variirt. Über motion-tracking-Systeme wurden die Daten der Probanden aufgezeichnet und anschließend analysiert. Untersucht wurde die Schrittlänge, die Schrittdauer und Synchronisationsphänomene.

So wurde gezeigt, dass die Schrittlänge sich direkt proportional zur Bewegungsgeschwindigkeit verhält und damit indirekt proportional zur Dichte. Die Schrittlänge nimmt demnach ab, wenn sich die Geschwindigkeit verringert. Dies hat zur Folge, dass selbst bei fast komplettem Stillstand Fußgänger trotzdem noch ihr Gewicht von einem auf den anderen Fuß verlagern und so "auf der Stelle laufen". Die Schrittdauer dagegen verlängert sich kaum bei zunehmender Dichte. Ebenfalls bei hohen Dichten beginnen die Fußgänger ihre Schritte zu synchronisieren um so Kollisionen zu vermeiden. [JARLP12]

4.4.2 Bewegung des Fußgängers entlang einer Trajektorie

Das gesamte Fußgänger-Modell wird nun entlang seiner Trajektorie bewegt. Hier wird linear zwischen den Trajektorienpunkten interpoliert und das Modell entsprechend der Zeit verschoben. $T_{current}$ bezeichnet den aktuellen Zeitpunkt der Simulation. Solange die Wiedergabe nicht angehalten ist verändert sich $T_{current}$ mit jedem Frame um die Zeitdifferenz zum vorherigen Frame.

Für jedes Fußgängerobjekt ist T die Menge aller Zeitpunkte zu denen Daten vorliegen. Nun müssen die beiden Trajektorienpunkte T_{start} und T_{end} ermittelt werden, die am nächsten vor bzw. nach dem aktuellen Zeitpunkt liegen und so den aktuellen Trajektorienabschnitt definieren. Es muss also gelten:

$$\begin{aligned} T_{start} \in T \mid \forall\, T_i \in T \;:\; |T_{current} - T_i| > |T_{current} - T_{start}| \vee T_{start} \leq T_{current} \\ T_{end} \in T \mid \forall\, T_i \in T \;:\; |T_{current} - T_i| > |T_{current} - T_{end}| \vee T_{end} \geq T_{current} \end{aligned} \tag{4.1}$$

T_{start} und T_{end} sind dann die Punkte zwischen denen das Modell bewegt werden muss. Der prozentuale Fortschritt der Bewegung zwischen den beiden Punkten zum Zeitpunkt $T_{current}$ ergibt sich damit zu:

$$p(T_{current}) = \frac{T_{current} - T_{start}}{T_{end} - T_{start}} \tag{4.2}$$

So kann dann der Standort $\vec{P}_{current}$ des Modells zum entsprechenden Zeitpunkt zwischen $\vec{P}_{start}$ und $\vec{P}_{end}$ interpoliert werden:

$$\vec{P}_{current} = \vec{P}_{start} + (\vec{P}_{end} - \vec{P}_{start}) \cdot p(T_{current}) \tag{4.3}$$

Das Modell wird dabei immer so rotiert, dass es mit Blickrichtung nach $\vec{P}_{end}$ ausgerichtet ist. Damit werden die Fußgänger entlang ihrer Trajektorien bewegt, gleiten aber immernoch regungslos über die Grundfläche. Deshalb wird parallel zu dieser Translation die Animation des Gangzyklus in einer Endlosschleife abgespielt um die Gehbewegungen darzustellen.

Hier ist es wichtig zu beachten, dass die Bewegungsgeschwindigkeit des Fußgänger mit der Animation übereinstimmt. Die Geschwindigkeit v eines Fußgängers berechnet sich aus der Differenz des letzten und des nächsten Trajektorienpunkts dividiert durch deren zeitliche Differenz und ist entlang eines Trajektorienabschnitts unverändert:

$$v = \frac{|\vec{P}_{end} - \vec{P}_{start}|}{T_{end} - T_{start}} \tag{4.4}$$

Die Geschwindigkeit wird nun verwendet um die Abspielgeschwindigkeit der Schrittanimation anzupassen. Der Geschwindigkeitsfaktor der Animation kann einfach mit $\frac{v}{1{,}32\mathrm{m/s}}$ berechnet werden (vgl. Kapitel 4.4.1).

Effiziente Bestimmung des Trajektorienabschnitts

Die in Formel 4.1 beschriebene Bestimmung der Anfangs- und Endpunkte des Trajektorienabschnitts. Die Bestimmung von T_{start} und T_{end} für den aktuellen Zeitpunkt muss in jedem Frame, für jedes Fußgängerobjekt, ausgeführt werden und muss daher möglichst effizient sein.

Oft sind die Zeiten der Trajektorienpunkte zwar in einem einheitlichen Rhythmus angelegt. Zum Beispiel, dass zu jeder vollen Sekunde ein neuer Trajektorienpunkt für alle Fußgängerobjekte angelegt wurde, allerdings ist dies keine Voraussetzung. Nach der Definition aus Kapitel 2.4.2 können die Daten, für jedes Objekt einzeln, zu beliebigen Zeitpunkten angelegt werden. Daher ist eine Speicherung, die einen Zugriff in konstanter Zeit auf die Daten ermöglicht (beispielsweise in einem Array) nicht möglich.

Damit der aktuelle Trajektorienabschnitt, wie in Formel 4.1 angegeben, in Abhängigkeit von $T_{current}$ effizient ermittelt werden kann, werden die raumzeitlichen Daten in einer Liste pro Fußgänger sortiert nach der Zeitkomponente gespeichert. Da sich die Trajektorienabschnitte nicht verändern muss der Aufwand für die Sortierung nur einmalig beim Start aufgebracht werden und ist üblicherweise in $O(n \log n)$ durchführbar. Mit dem Anfrageparameter $T_{current}$ können dann über eine binäre Suche in der sortierten Liste die beiden Zeitpunkte T_{start} und T_{end} ermittelt werden, die den aktuellen Trajektorienabschnitt eingrenzen. Die Laufzeitkomplexität der Suche ist dabei $O(\log n)$. Wie in Kapitel 4.3.1 beschrieben kann der Nutzer die Abspielposition der Visualisierung beliebig verändern. Da sich die Abspielposition dadurch zu jedem Zeitpunkt beliebig verändern kann, muss für jeden Frame erneut eine Suche mit der aktuellen Abspielposition ausgeführt werden.

4.5 Implementierung der Analysewerkzeuge

Die Werte nach denen die Simulation analysiert werden kann wurden in Kapitel 2.4.3 aufgezählt. Dabei lässt sich unterscheiden, ob sich die Messwerte auf einen einzelnen Fußgänger (z.B. Geschwindigkeit) beziehen, auf die gesamte Szene (z.B. Population) oder einen Ausschnitt der Szene, wie beispielsweise eine Messlinie (z.B. Fluss) oder einen Messbereich (z.B. Dichte).

Daraus ergibt sich folgende Definition für die Kategorisierung von Messwerten in Fußgängerdaten:

- **individuelle Messwerte** bezeichnen Messwerte die pro Fußgänger berechnet werden können. Jeder Fußgänger kann dabei einen anderen, individuellen Wert haben.
- **lokale Messwerte** bezeichnen einen Messwert der für einen definierten Messbereich oder eine Messlinie gültig ist.
- **globale Messwerte** bezeichnen einen Messwert der sich auf den gesamten Datensatz bezieht.

Für die meisten Messwerte lässt sich dann die Veränderung über den zeitlichen Verlauf hinweg betrachten, oder ein Mittelwert darüber bilden. Im Nachfolgenden werden die einzelnen Messwerte, die in SumoViz Unity zur Verfügung stehen, besprochen und Implementierungsdetails erläutert.

4.5.1 Messung der Geschwindigkeit

Die Berechnung der Geschwindigkeit eines Fußgängers lässt sich mit geringem Aufwand bewerkstelligen. Jeder Trajaktorienabschnitt eines Fußgängers hat einen Anfangs- und Endpunkt sowie eine Anfangs- und Endzeit. Zwischen den Punkten bewegt sich der Fußgänger linear. Damit lässt sich die Geschwindigkeit über den Abstand der Punkte und die dafür benötigte Zeit berechnen, wie in Formel 4.4 definiert. Da die Zeit in Sekunden angegeben wird und die Koordinaten der Punkte Metern entsprechen, wird die Geschwindigkeit in m/s angegeben. Der Wertebereich der Geschwindigkeiten liegt zwischen 0m/s und $1,69$m/s [Wei93] und kann in der Visualisierung pro Fußgänger farblich visualisiert werden. Die niedrigste Geschwindigkeit wird rot, die höchste blau dargestellt. Die Berechnung der Geschwindigkeit benötigt keinen zusätzlichen Rechenaufwand, da die Trajektorienpunkte bereits für jeden Fußgänger bestimmt wurden.

4.5.2 Messung der Dichte

Einer der wichtigsten Kennwerte für die Analyse von Fußgängerströmen ist die Dichte. Übernommen wurde sie aus der Physik, die die Dichte eines Körpers als Quotient seiner Masse und des Volumens definiert ($\varrho = \frac{m}{V}$). Speziell die Fluiddynamik wird oft als Analogie zur Fußgängerforschung verwendet. Dabei entsprechen die einzelnen Moleküle/Atome den Fußgängern und deren Anzahl in einem bestimmten Bereich definiert die Dichte. Bei der Fußgängerforschung gibt es keine einheitliche Definition für die Berechnung der Dichte, sondern eine Reihe verschiedener Berechnungsverfahren.

Standarddefinition der Dichte

Die häufigste Definition der Fußgängerdichte k eines Bereichs A ist wie folgt definiert und wird als Standarddefinition bezeichnet: [SS10]

$$k_s = \frac{N}{|A|} \tag{4.5}$$

Dabei bezeichnet N die Anzahl der Fußgänger im Bereich und $|A|$ die Fläche des Bereichs. Daraus ergibt sich als Einheit für die Dichte $\frac{\mathrm{P}}{\mathrm{m}^2}$.

Bedingt durch diese Definition kann eine Dichte immer nur für einen bestimmten Bereich angegeben werden (lokaler Messwert). Üblicherweise werden dabei rechteckige Bereiche verwendet, dies ist aber nicht Voraussetzung. Damit die Werte für die Dichte zuverlässig und vergleichbar sind, sollte der Messbereich nicht zu klein gewählt werden, denn sehr kleine Messbereiche können auf Grund der maximalen Dichte nur eine einzige Person beinhalten. Um zuverlässige Werte zu messen sollte ein Mittelwert über einen gewissen Zeitraum gebildet werden und die Größe und Positionierung des Messbereichs gut gewählt werden. [SS10]

Die Berechnung der Dichte ist effizient möglich, da sich der Messbereich nicht verändert und bei jeder Bewegung der Fußgänger ein einfacher Test ergibt, ob der Fußgänger in den Messbereich eingetreten ist ($N = N + 1$), oder diesen verlassen hat ($N = N - 1$).

Berechnung individueller Dichtewerte

Mit der Definition der Dichte nach Formel 4.5 ist die Messung immer nur für einen Bereich möglich, nicht aber für einen einzelnen Fußgänger (individuelle Messung). Wünschenswert ist jedoch, individuell für jeden Fußgänger angeben zu können, wie dicht seine Umgebung ist. So können beispielsweise einzelne Fußgänger in der Visualisierung entsprechend ihrer Dichte eingefärbt werden und auf Grenzwertüberschreitungen geprüft werden. Die Berechnung der Dichte muss dafür mit einem anderen Verfahren bestimmt werden.

Voronoi-Diagramme Steffen und Seyfried beschreiben die Nutzung von Voronoi-Diagrammen zur Dichteberechnung. [SS10] Ein Voronoi-Diagramm ist ein Raumzerlegungsverfahren, bei dem der vorhandene Raum in N Zellen zerlegt wird, wobei in jeder Zelle genau ein Fußgänger liegt. Die Ausbreitung einer Zelle ist definiert durch den Bereich um den Fußgänger herum, der dem jeweiligen Fußgänger näher ist als allen anderen Fußgängern. [Aur91] Dies hat zur Folge, dass die Zellen kleiner werden, je enger die Fußgänger positioniert sind. Daraus lässt sich eine Dichtefunktion für den jeweiligen Fußgänger i aufstellen, die sich indirekt proportional zur Fläche der Voronoizelle A_i verhält:

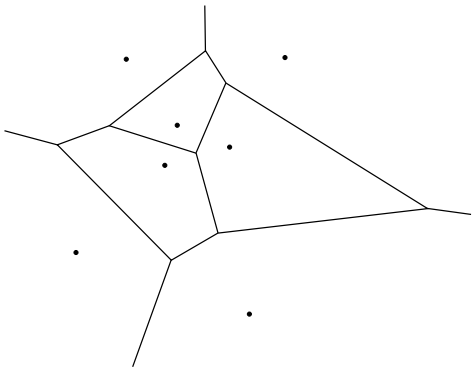


Abbildung 4.6: Voronoi-Diagramm [Mat09]

$$k_i = \frac{1}{|A_i|} \tag{4.6}$$

Problematisch bei dieser Definition der Dichte ist das Verhalten am Rand der Fußgängergruppe, denn hier ist die Voronoizelle nicht begrenzt und, wenn der Raum nicht durch Wände oder ähnliches begrenzt wird, unendlich groß und die Dichte damit 0.

Die Berechnung des Voronoi-Diagramms muss nicht für jeden Fußgänger einzeln erfolgen, sondern jeweils für den gesamten Raum zu einem bestimmten Zeitpunkt. Fortune stellte 1987 einen Sweep-Line-Algorithmus vor, der Voronoi-Diagramme mit einer Laufzeit von $O(n \log n)$ berechnen kann. [For87] Anschließen muss noch die Fläche jeder Voronoizelle für jeden Fußgänger berechnet werden, was nochmals lineare Zeit benötigt.

Standarddefinition mit fester Fläche Ein simpler Ansatz, der auf dem selben Verständnis wie die Standarddefinition der Dichte basiert, ist eine Umkreissuche mit festem Radius. Dabei wird ein Kreis mit einem vorgegebenen Radius r um den jeweiligen Fußgänger gebildet und die Anzahl der Fußgänger in diesem Kreis gezählt. Somit kann die gleiche Formel wie in 4.5 verwendet werden. Die Fläche entspricht konstant der Fläche des Kreises mit dem jeweiligen Fußgänger im Zentrum. Gezählt werden dann alle Fußgänger in diesem Kreis.

Dafür muss für jeden Fußgänger getestet werden, ob jeder andere Fußgänger innerhalb oder außerhalb des Kreises liegt ("naiver Algorithmus"). Solch eine Berechnung lässt sich ohne die Verwendung spezieller räumlicher Datenstrukturen nicht sehr effizient durchführen. Die Laufzeit für den naiven Ansatz ist quadratisch ($O(n^2)$). Die Implementierung verwendet trotzdem diesen naiven Algorithmus, da die Implementierung von Fortunes Algorithmus für das Voronoi-Diagramm sehr aufwändig ist. Außerdem sind die Geschwindigkeitsunterschiede erst bei sehr hohen Fußgängerzahlen merklich.

Ein entscheidender Parameter für die Dichtemessung mit festem Radius ist die Wahl eines passenden Radius. Wird der Radius sehr groß gewählt, enthält jeder Kreis alle Fußgänger und somit haben alle Objekte die selbe Dichte. Dadurch wird der Wert für die Dichte zwar nicht falsch, ist aber wenig aussagekräftig um verschiedene Personen innerhalb einer Simulation zu vergleichen. Zum Vergleich verschiedener Szenarien könnte dieser Wert trotzdem noch genutzt werden. Wenn der Radius zu klein gewählt und der Kreis immer nur das Objekt selbst enthält, nimmt die Dichte auch für alle Personen den selben Wert von $\frac{1}{r^2\pi}$ an, da im Kreis kein Platz für weitere Personen ist.

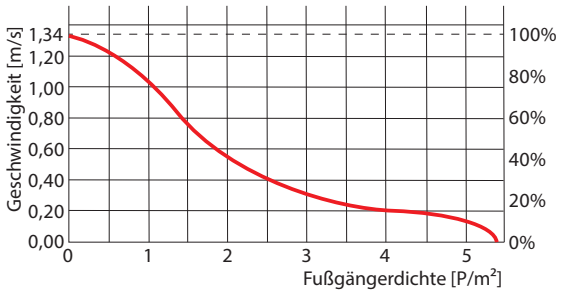


Abbildung 4.7: Fundamentaldiagramm: Abhängigkeit der Fußgängergeschwindigkeit von der Fußgängerdichte [Wei93]

Nach Weidmann ist "bereits bei einer bescheidenen Fußgängerdichte von $0,5\mathrm{P/m}^2$ [...] ein spürbarer Rückgang der Geschwindigkeit um nahezu 10 % zu erwarten." [Wei93]. Demnach ist bereits eine Dichtenveränderung auf einer Fläche von $2\mathrm{m}^2$ relevant für den Fußgängerfluss. Dies entspricht einem Kreis um den Fußgänger mit einem Radius von ungefähr $0,8\mathrm{m}$ dessen Dichte für die Geschwindigkeit des Fußgängers relevant ist. Diese Annahme ist sicherlich nicht komplett korrekt, da die Dichte vor einem Fußgänger wesentlich entscheidender ist, als die Dichte hinter dem Fußgänger. Als Annäherung ist die

kreisförmige Dichtebestimmung aber ein guter Ansatz. Abbildung 4.7 zeigt den Einfluss der Dichte auf die Geschwindigkeit und nennt einen Dichtewert von etwa $5\mathrm{P/m}^2$ bei dem der Verkehrsfluss zum erliegen kommt. [Wei93]

Abhängig vom Verwendungszweck der Daten, kann sich mit dieser Methode der Dichteberechnung gegebenenfalls ein Problem ergeben. Die Anzahl der Nachbarn die sich im Umfeld des Fußgängers befinden ist immer eine Ganzzahl und liegt für einen Kreis mit Radius $r = 1,0\mathrm{m}$ erfahrungsgemäß in einem Bereich von 0 bis 5. Somit ergibt sich jedoch keine kontinuierliche Dichteverteilung, sondern es werden immer wieder die gleichen Dichtewerte berechnet. Diese Werte sind nicht falsch, trotzdem kann eine solche Diskretisierung in manchen Fällen unerwünscht sein (vgl. Kapitel 4.5.4).

Standarddefinition mit fester Personenzahl Um die Diskretisierung der Dichte zu vermeiden kann auch ein anderer Ansatz zur Dichteberechnung gewählt werden, der auch auf der Standarddefinition beruht, aber kontinuierliche Dichtwerte erzeugt. Oben wurde beschrieben, wie die Dichte mit für eine Person mit fester Fläche berechnet werden kann. In diesem Ansatz wird nun kein fester Radius definiert, sondern Größe des Umkreises über eine feste Personenzahl definiert, die sich innerhalb des Kreises befinden muss. Dieser Ansatz wird kNN-Distanz genannt und verwendet die Distanz zum k-ten Nachbarn als Radius für die Dichteberechnung. [GRHS04] Hierbei ist die Wahl von k ein entscheidender Faktor. Außerdem zeigt diese Variante der Dichteberechnung gerade bei geringen Dichten Probleme auf. Wenn der k-te Nachbar soweit entfernt ist, dass er nicht mehr für die individuelle Dichte des Fußgängers relevant ist, sind die Werte wenig aussagekräftig.

Laufzeitoptimierung für die Dichteberechnung einzelner Fußgänger Zur Verbesserung der Laufzeit wird der naive Algorithmus um eine Optimierung erweitert: Die Dichteveränderungen zwischen zwei aufeinanderfolgenden Frames sind üblicherweise nicht sehr groß, da die Fußgänger sich nur wenig bewegt haben und sich oft sogar in der selben Richtung bewegen. Gerade für die Visualisierung mittels Farben, sind minimale Abweichungen vom tatsächlichen Dichtewert nicht zu erkennen. Daher muss die Dichte nicht in jedem Frame neu berechnet werden, sondern kann alle f Frames neu berechnet werden. So kann in jedem Frame nur ein Teil (N/f) der Dichtewerte aktualisiert werden und im nächsten Frame werden die Dichtewerte von den Objekten aktualisiert, die am längsten nicht mehr aktualisiert wurden.

4.5.3 Analyse mit Hilfe einer Messlinie

Manche Messgrößen beziehen sich auf eine Messlinie, die von den Trajaktorien der Fußgänger zu bestimmten Zeitpunkten gekreuzt wird. SumoViz Unity kann eine Messlinie bestehend aus einem Anfangs- und Endpunkt auf der Grundebene definieren und zur Analyse nutzen.

Zum einzeichnen der Messlinie wird die Maus genutzt. Durch zwei Klicks werden Punkte definiert, die die Endpunkte der Linien darstellen. Die Herausforderung hierbei ist die Bestimmung dieser Punkte auf der Grundebene. Die 2D-Position der Maus, die beim Klick bestimmt wird, bezieht sich auf den Bildschirm und muss dann in den 3D-Raum übersetzt werden, damit der Schnittpunkt mit der Grundebene bestimmt werden kann. Für die Berechnung eines Frames wird aus der Kameraposition eine Projektionsmatrix bestimmt, die die

dreidimensionalen Koordinaten des Raums auf die 2D-Bildschirmebene transformiert. Diese Projektionsmatrix lässt sich auch rückwärts auf die 2D-Koordinaten anwenden und so der Punkt des Mausklicks auf der Projektionsebene im 3D-Raum berechnen. Dann wird mittels "Raycasting" ein Strahl erzeugt der von der Kamera durch die Position der Maus geschossen wird und solange verfolgt wird, bis er die Grundebene schneidet. Der Schnittpunkt ist dann der Punkt auf der Grundebene an dem die Messlinie beginnt bzw. endet. [Wat93]

Anhand der Messlinie werden die folgenden Messwerte bestimmt. Abgesehen von der Länge der Linie basieren alle Messwerte auf dem Überschreiten der Messlinie durch einen Fußgänger.

- **Länge der Linie** Kann verwendet werden um Bereiche der Simulation zu vermessen und ist eine wichtige Grundlage für die Berechnung weiterer Messwerte. Die Berechnung der Länge ist mit gegebenem Start- und Endpunkt trivial.

- **Anzahl der Überquerungen** Wieviele Fußgänger die Linie überqueren ist ein wichtiger Messwert und Grundlage für die Berechnung des Flusses. In jedem Frame wird die Position aller Fußgänger schrittweise verändert (vgl. Kapitel 4.3). Wenn das Liniensegment zwischen alter und neuer Position die Messlinie schneidet wird eine Überschreitung der Messlinie gezählt. Hierbei ist eine effiziente Berechnung notwendig, da der Schnitt in jedem Frame für jeden Fußgänger geprüft werden muss. Zur Optimierung muss nicht berechnet werden wo der Schnittpunkt liegt, sondern nur, ob sich die Linien schneiden oder nicht. Hier kommt Franklin Antonios Algorithmus "Faster Line Segment Intersection" zum Einsatz, der in konstanter Zeit und mit sehr wenigen Operationen bestimmen kann, ob ein Schnitt vorliegt oder nicht. [Ant92]
  Liegt ein Schnitt vor, wird ein einfacher Zähler inkrementiert. Dabei ist es egal, in welcher Richtung die Fußgänger die Messlinie kreuzen und ob ein Fußgänger die Linie schon einmal gekreuzt hat oder nicht.

- **Geschwindigkeit** In dem Moment, in dem ein Fußgänger die Linie kreuzt wird die aktuelle Geschwindigkeit des Fußgängers berechnet. Aus allen Geschwindigkeiten seit dem Erstellen der Linie wird eine Durchschnittsgeschwindigkeit berechnet und angezeigt. Der Mittelwert wird dabei inkrementell bestimmt und ist so effizient berechenbar, wobei N die Anzahl der Linienüberquerungen bezeichnet und v die Geschwindigkeit des aktuellen Fußgängers:
  $$\bar{v}_{neu} = \frac{\bar{v}_{alt} \cdot (N-1) + v}{N} \quad \text{in } [\frac{m}{s}] \tag{4.7}$$

- **Fluss** Aus der Anzahl der Überquerungen pro Zeit und der Länge der Messlinie lässt sich der Fluss über die Messlinie bestimmen. Würde man hier die schon berechnete Anzahl der Überquerungen nutzen könnte man lediglich einen Mittelwert über die gesamte Zeit der Zählung berechnen. Um einen genaueren Wert für den Fluss zu berechnen muss das Zeitfenster kleiner gewählt werden. SumoViz Unity verwendet für die Berechnung des Flusses ein fließendes Zeitfenster ("sliding window") von einer Sekunde. Der Fluss berechnet sich damit aus der Anzahl der Überquerungen in der letzten Sekunde und der Länge der Messlinie.
  Für die Berechnung der Überquerungen der letzten Sekunde wird eine einfache Liste verwendet. Neue Überquerungen werden am Anfang der Liste eingefügt, während Überquerungen, die nicht mehr relevant sind (älter als eine Sekunde) von hinten aus der Liste entfernt werden.

Der Fluss q berechnet sich somit aus der Größe der Liste L der Überquerungen der letzen Sekunde und der Länge der Messline l bei festen Größe des Zeitfensters von einer Sekunde: [Kne13]

$$q = \frac{|L|}{|l| \cdot 1\mathrm{s}} \quad \text{in } [\frac{\mathrm{P}}{\mathrm{m\,s}}] \tag{4.8}$$

Sobald eine Messlinie in die Visualisierung eingezeichnet wurde, werden die oben genannten Messwerte berechnet. Um die kumulierten Werte wie Anzahl oder Durchschnittsgeschwindigkeit zurückzusetzen muss die Linie entfernt und neu eingezeichnet werden. Solange die Messlinie nicht eingezeichnet ist werden auch keine Messwerte berechnet.

4.5.4 Fundamentaldiagramm

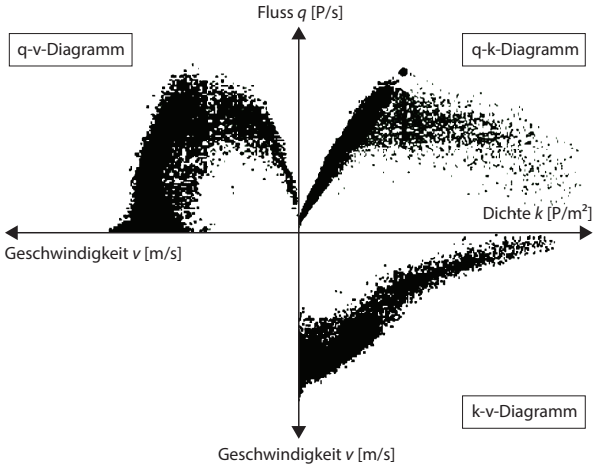


Abbildung 4.8: Fundamentaldiagramm [Küh04]

Das Fundamentaldiagramm betrachtet die wichtigsten makroskopischen Kenngrößen für die Verkehrsbeobachtung. Oft wird es im Zusammenhang mit der Analyse von Fahrzeugen auf Straßen verwendet, doch die Definitionen lassen sich auch auf Fußgängerverkehr anwenden und so verwendet zu Beispiel auch Weidmann [Wei93] in seinem Werk das Fundamentaldiagramm um den Zusammenhang von Verkehrsstarke, Verkehrsdichte und Geschwindigkeit zu beschreiben. [KKR+08] In Abbildung 4.8 sind alle möglichen zweidimensionalen Projektionen des Fundamentaldiagramms zu sehen.

Die Implementierung von SumoViz Unity erlaubt es ein Fundamentaldiagramm zu erstellen. Hierbei wird das k-v-Diagramm generiert, welches die Geschwindigkeit gegen die Dichte anträgt. Diese Darstellung wird auch in vielen Publikationen zur Fußgängeranalyse verwendet. Die Verkehrsstärke bzw. der Fluss wird dabei außer Betracht gelassen. Dadurch ergibt sich der Vorteil, dass sich das Diagramm nicht auf eine konkrete Messlinie bezieht. Sowohl Dichte als auch Geschwindigkeit sind Werte, die unabhängig eines Bereichs oder einer Messlinie für jeden Fußgänger ermittelt werden können. Abbildung 4.7 zeigt solch ein k-v-Fundamentaldiagramm nach Weidmann.

In SumoViz Unity erzeugt das Fundamentaldiagramm ein Streudiagramm (engl. "scatter plot") in dem für jeden Fußgänger ein Punkt eingezeichnet wird. An der y-Achse wird die Geschwindigkeit angetragen, an der x-Achse die Dichte. [KKR+08]

In der Implementierung kann das Fundamentaldiagramm zu einem bestimmten Zeitpunkt des Datensatzes angefordert werden. Dabei kann beobachtet werden, dass sich sowohl die Dichte- als auch die Geschwindigkeitswerte nicht kontinuierlich verteilten, sondern zu bestimmten Werten häufen.

Die Diskretisierung der Dichtewerte erklärt sich durch die verwendete Berechnungsmethode. In SumoViz Unity kommt eine Dichteberechnung mit mittels Umkreissuche mit festem Radius zum Einsatz. Der Dichtewert bestimmt sich dann durch die Anzahl der Personen und die

Fläche des Umkreises. Da für alle Fußgänger der selbe Umkreis verwendet wird, unterscheiden sich die Dichtewerte nur nach der Anzahl der Fußgänger im Umkreis. Dadurch kann die Dichte nur bestimmte diskrete Werte annehmen.

Die Diskretisierung der Geschwindigkeitswerte ist kein Problem der Berechnungsmethode, sondern ein Phänomen mancher Datensätze. Abhängig von der Simulationsmethode kann es sein, dass für die Geschwindigkeiten der Fußgänger nur bestimmte Werte auftreten. Beispielsweise bei der Verwendung von zellulären Automaten ist dieses Phänomen oft zu beobachten.

Um die Diskretisierung der Dichte zu vermeiden wurde das Berechnungsverfahren für das Fundamentaldiagramm testweise verändert. Die Dichte eines Fußgängers wurde über alle Trajektorienpunkte hinweg als Dichtemittelwert implementiert. Dadurch ergab sich jedoch auch nicht die übliche Verteilung, sondern eine Häufung der Werte bei mittleren Dichten. Deshalb wurde entschieden die Standarddefinition der Dichte mit festem Radius beizubehalten. Der Radius wurde allerdings auf zwei Meter vergrößert um ein größeres Spektrum verschiedener Dichtewerte zu unterscheiden. Zusätzlich wird noch eine Trendlinie in das Fundamentaldiagramm eingezeichnet, die in den meisten Fällen der aus Abbildung 4.7 sehr nahe kommt.

Kapitel 5

Ausblick und Fazit

5.1 Vergleich zur Bachelorarbeit

Abbildung 5.1: Vergleich einer Visualisierung der gleichen Simulationsergebnisse in SumoViz3D (links) und SumoViz Unity (rechts)

Der wichtigste Unterschied zwischen der Implementierung aus der Bachelorarbeit und der Implementierung aus dieser Arbeit ist der Technologiewechsel von WebGL zu einer klassischen Game-Engine. Im Vergleich mit der webbasiserten Lösung aus der Bachelorarbeit ist durch die Nutzung der Game-Engine Unity eine deutliche Steigerung in der 3D-Leistung erkennbar. Während bei der Bachelorarbeit ein Fußgängermodell mit wenigen Polygonen zum Einsatz kam um eine flüssige Darstellung zu gewährleisten konnte bei der Implementierung in Unity ein wesentlich aufwändigeres Modell mit über 3000 Vertices genutzt werden. Abbildung 5.1 zeigt die Darstellung der selben Szene in beiden Visualisierungsprogrammen. Zusätzlich werden in SumoViz Unity die Fußgänger animiert dargestellt und die Positionen zwischen zwei Trajektorienpunkten interpoliert.

Den entscheidenten Unterschied bei der Entwicklung machten die 3D-Bibliotheken. Während die verwendete WebGL-Bibliothek *Three.js* nur grundlegende Funktionen zur Verfügung stellte, ist *Unity3D* eine umfangreiche und ausgereifte Engine, die auch für die Erstellung von anspruchsvollen Anwendungen geeignet ist. Grundlegende Funktionalitäten wie Kameras, Lichter, Vektoren, etc. werden sowohl von Unity als auch Three.js bereit gestellt. Unity bietet darüber hinaus eine Menge weiterer Funktionen für Texturen, Animationen, Laden von

externen Modellen und die Verarbeitung von Nutzer-Eingaben.

Ein wichtiges Argument für die Nutzung von WebGL war die plattformübergreifende Unterstützung des Standards. In diesem Punkt kann Unity, mit der Unterstützung aller relevanten Plattformen, problemlos mit der WebGL-Lösung mithalten. Bei der browserbasierten Lösung muss für die Nutzung keine zusätzliche Software installiert werden, da alle Programmdaten aus dem Internet abgefragt werden können. Dafür ist der Betrieb eines Servers notwendig, der die Daten bereitstellt. Bei SumoViz Unity muss die Visualisungssoftware lokal auf dem Rechner installiert werden und beinhaltet dann aber alle benötigten Daten.

5.2 Möglichkeiten der Weiterentwicklung

Obwohl einige Probleme gegenüber der Implementierung aus der Bachelorarbeit verbessert werden konnten, haben sich während der Erstellung der Software noch einige Bereiche ergeben, die Raum für Verbesserung offen lassen. Ob die möglichen Verbesserungen sinnvoll sind und den teilweise hohen Aufwand für die Implementierung rechtfertigen muss im Einzelfall geprüft werden.

5.2.1 3D-Modell der Fußgänger

Die Implementierung verwendet nur das 3D-Modell eines Fußgängers auf dessen Textur unterschiedliche Farben multipliziert werden um verschiedene Personen zu erzeugen. Das liegt daran, dass die Gangzyklus-Animation auf das jeweilige Modell angepasst werden muss. Mit der Verwendung eines einzelnen Modells musste nur eine Gangzyklus-Animation erstellt werden. Für eine realistischere Darstellung sollten jedoch Modelle verschiedener Geschlechter, Größen und Körperbaus verwendet werden, auf die eine Vielzahl unterschiedlicher Texturen angewendet wird.

In den Fußgängerdaten liegt nicht vor, ob es sich bei den entsprechenden Fußgängern um Männer, Frauen oder Kinder handelt. Wie Weidmann aber aufzeigt, unterscheidet sich der Gang, nicht nur an Hand dieser Merkmale, erheblich zwischen verschiedenen Personengruppen. [Wei93] Daher ist es ohne genauer Daten problematisch passende Modelle zu wählen. Trotzdem könnte mit der Verwendung mehrerer verschiedener Fußgängermodelle die Darstellung verbessert werden. Die Modelle sollten dann Durchschnittswerten für den Körperbau entsprechen, oder dynamisch nach Verteilungsfunktionen erzeugt werden.

5.2.2 Gangzyklus-Animation

Die Animation des Gangzyklus wird in ihrer Geschwindigkeit auf den Fußgänger angepasst. Das entspricht jedoch nicht dem realen Verhalten. Wie in Kapitel 4.4.1 beschrieben wird nicht die Zyklusdauer verändert, sondern die Schrittlänge. Muss ein Fußgänger langsamer gehen, macht er kleinere Schritte und nicht wie in der aktuellen Implementierung langsamere.

Die Geschwindigkeit der Gang-Animation lässt sich jedoch sehr einfach verändern, während eine Veränderung der der Schrittlänge nur sehr aufwändig um zusetzen wäre. Deshalb wurde in der Implementierung der einfachere Weg gewählt.

5.2.3 Mess- und Analysewerkzeuge

SumoViz Unity bietet bereits einige Mess- und Analysewerkzeuge. Weitere können nach Bedarf implementiert werden. Neben dem bereits implementierten Einzeichnen einer Messlinie, könnten beispielsweise durch das Einzeichnen eines zweidimensionalen Messbereichs lokale Messwerte erhoben werden. Wie auch schon in anderer Visualisungssoftware gesehen, wird hierfür ein rechteckiger Messbereich auf der Grundfläche aufgespannt. Dort können die selben Messwerte im Bezug auf den Bereich erhoben werden, die auch global gemessen werden. Zusätzlich können auch Werte wie der Fluss, die in SumoViz Unity anhand der Messlinie ermittelt werden, auf Grundlage eines rechteckigen Messbereichs gemessen werden.

Die detaillierte Analyse und Verarbeitung der Messwerte ist nicht im Bereich der Post-Visualisierung angesiedelt. Daher könnte für diesen Zweck eine Exportfunktion die Messwerte für andere Programme zur Verfügung stellen.

5.2.4 Interpolation zwischen Trajektorienpunkten

Die Fußgängerdaten bestehen aus Trajektorien, zwischen deren Punkten linear interpoliert wird. Dies hat zur Folge, dass Ecken entstehen, an denen die Fußgänger aprupt ihre Richtung ändern. Meist ist der Winkel der Richtungsänderung relativ gering und fällt daher nicht stark auf. Wurden die Trajektorienpunkte jedoch für alle Fußgänger immer zum selben Zeitpunkt erstellt, ändern alle Fußgänger zum gleichen Zeitpunkt ihre Richtung. Dies ist dann als deutliches Ruckeln erkennbar.

Eine Möglichkeit dieses Problem zu korrigieren, wäre es demnach die Daten genauer zu erfassen. Je öfter Trajektorienpunkte erzeugt werden, desto genauer sind die Trajektorien an den ursprünglichen Wegen der Fußgänger. Zusätzlich könnten die Trajektorienpunkte nicht für alle Fußgänger zum gleichen Zeitpunkt erzeugt werden. Dadurch würden immer nur wenige Fußgänger gleichzeitig ihre Richtung ändern und die Richtungsänderungen unauffälliger.

Aber auch mit unveränderten Simulationsergebnissen könnten Veränderungen an der Darstellung implementiert werden. Würde zwischen den Trajektorienpunkten nicht linear interpoliert werden, könnten die Eckpunkte abgerundet werden. Mit einer Spline-Interpolation oder Bézierkurven zwischen den Trajektorienpunkten würden die Bewegungen der Fußgänger wesentlich realistischer wirken. Jedoch wäre die Darstellung somit nicht mehr entsprechend der Originaldaten und es muss abgewogen werden, welcher Aspekt wichtiger ist.

Auch die Veränderung der Geschwindigkeit ist durch die Trajektorienpunkte definiert und berechnet sich aus dem Abstand der beiden aktuellen Trajektorienpunkte (vgl. Kapitel 4.4.2). Wie für die Veränderung der Richtung könnte auch für die Veränderung der Geschwindigkeit ein Interpolationsverfahren genutzt werden, das abrupte Veränderungen vermeidet.

5.3 Fazit

Abschließend betrachtet lässt sich sagen, dass die Visualisierung mittels einer Game-Engine einen deutlichen technologischen Sprung gegenüber der browserbasierten Lösung ermöglicht hat. Für grafisch anspruchsvolle Visualisierungen stellen Game-Engines eine gute Basis dar.

Trotzdem kann die webbasierte Umsetzung weiterhin Anwendungsfälle finden. Gerade für einfachere und schematischere Darstellungen sind moderne Webtechnologien sicherlich auch gut geeignet. Die Entwicklung von Browser-APIs und WebGL-Frameworks ist in einer sehr aktiven Phase und schafft viele neue Möglichkeiten für eine browserbasierte Visualisierung.

Die Praxis hat gezeigt, dass Visualisierungssoftware auch oft dazu verwendet wird detaillierte Bilder und Videosequenzen von Simulationen zu genereieren, die beispielsweise in der Presse- und Öffentlichkeitsarbeit zum Einsatz kommen. Hierbei ist eine grafisch ansprechende Darstellung wünschenswert, weshalb sich für diesen Anwendungsfall eine Game-Engine-basierte Lösung wesentlich besser eignet.

Speziell Unity, aber auch andere Game-Engines haben den Markt außerhalb der klassischen Videospiele entdeckt und bieten Unterstützung für Anforderungen von "serious games", wie beispielsweise der Simulationsvisualisierung, Architektur und des Inginieurswesens. Die US Air Force setzt zum Beispiel eine Unity basierte Lösung für das Training von Piloten ein. [Uni14] Bisher waren Videospiele zwar die treibende Kraft am Markt, doch die zunehmend Nutzung von Game-Engines für andere Inhalte ist ein Trend und so ist vielleicht in der Zukunft auch der Begriff "Game-Engine" nicht mehr passend.

Anhang A

Klassenübersicht SumoViz Unity

Im Folgenden werden einzelnen Klassen, die Bestandteil der entwickelten Simulationsvisualisierung "SumoViz Unity" sind, aufgelistet und erläutert. Diese Auflistung beschränkt sich auf die wichtigsten Klassen für das Verständnis der Struktur und ist nicht vollständig. Zudem werden in Abschnitt A.5 Komponenten von Drittentwicklern, die nicht Teil von Unity sind, entsprechend ausgewiesen.

Bei einigen Klassen wird zusätzlich eine Übersicht über die wichtigsten öffentlichen Methoden oder Attribute und deren Funktionalität gegeben.

A.1 Allgemein

PlaybackControl
Kontrolliert die Wiedergabe des Datensatzes. Auch die Steuerung der Simulation durch den Nutzer mit den Buttons wird in dieser Klasse verarbeitet.

Die Klasse definiert unter anderem folgende Attribute:
`bool playing` – Play/Pause-Status der Visualisierung
`decimal current_time` – aktueller Abspielzeitpunkt
`decimal total_time` – Gesamtdauer der Visualisierung

FundamentalDiagram
`drawFundamentalDiagram(Rect position)`
Erzeugt das Fundamentaldiagramm an der angegebenen Position.

InfoText
Die Berechnung und Anzeige der Messwerte, entsprechend der ausgewählten Werkzeuge, werden in dieser Klasse gesteuert. Das Zeichnen der Diagramme für einzelnen Messwerte ist ebenfalls in dieser Klasse implementiert.

FileLoader
Implementierung eines beispielhaften Import von Daten aus Text-Dateien.

Groundplane
Grundebene, auf der die Fußgänger- und Geometrieobjekte dargestellt werden. Das Einzeichnen der Messlinie ist in dieser Klasse implementiert.

A.2 Fußgänger

PedestrianPosition
Beschreibt einen Trjaketorienpunkt eines Fußgängers.
`PedestrianPosition(int id, decimal timestamp, float x, float y)`

PedestrianLoader
Schnittstelle zur Erstellung der `Pedestrian`-Objekte.

`void addPedestrianPosition(PedestrianPosition p)`
Über diese Methode werden alle Trajektorienpunkte einzeln an den PedestrianLoader übergeben. Dabei ist die Reihenfolge der Aufrufe dieser Methode egal.

`void createPedestrians()`
Sobald alle Trajektorienpunkte an den PedestrianLoader übergeben wurden, wird diese Methode aufgerufen um Fußgängerobjekte zu erzeugen.

Pedestrian
Instanz eines Fußgängers. Beinhaltet eine Referenz auf alle Trajektorienpunkte und das 3D-Modell des Fußgängers. Die Farbe eines Fußgängers wird zufällig bestimmt.

`void showTrajectory()`
Blendet die Trajektorie des Fußgängers auf der Grundebene ein. Die Trajektorie hat die Farbe des Fußgängers. Mit `hideTrajectory()` kann die Trajektorie wieder ausgeblendet werden.

`float getDensity()`
Gibt die Dichte des Fußgängers zum aktuellen Zeitpunkt in P/m^2 zurück. Zur Berechnung wird ein Umkreis mit Radius 2m verwendet.

`float getSpeed()`
Gibt die Geschwindigkeit des Fußgängers zum aktuellen Zeitpunkt in m/s zurück.

A.3 Darstellungsmodi

ThemingMode
Abstrakte Basisklasse zur Implementierung verschiedener Darstellungsmodi.

`abstract string getTerrainName()`
Gibt den Namen des in Unity angelegten Terrains für den entsprechenden Darstellungsmodus zurück.

`abstract Material getWallsMaterial()`
Gibt ein Material zurück, das als Textur auf alle Wände angewendet wird. Es existieren weitere Methoden für die Materialien von Häusern, Dächern, etc. die von jedem Darstellungsmodus implementiert werden müssen.

A.4 Geometrie

GeometryLoader
Schnittstelle zum Erstellen der Geometrieobjekte und zur Einstellung des Darstellungsmodus.

`void addObject(Geometry obj)`
Zur Darstellung eines Geometrieobjekts muss eine Instanz der entsprechenden Unterklasse von `Geometry` erzeugt werden und an den `GeometryLoader` übergeben werden.

`void setTheme(ThemingMode mode)`
Um den Darstellungsmodus festzulegen wird eine Unterklasse von `ThemingMode` instanziiert an den `GeometryLoader` übergeben.

Geometry
Abstrakte Basisklasse für die Implementierung von Geometrieobjekten.

AreaGeometry
`void create (string name, List<Vector2> verticesList)`
Unterklasse von `Geometry`, die die Darstellung von Geometrieobjekten als zweidimensionale Fläche implementiert. Die übergebene Liste definiert die Fläche als Polygon. Diese Darstellung wird beispielsweise für Quell- und Zielbereiche verwendet.

TreeGeometry

`void create (string name, Vector2 vertices)`

Unterklasse von `Geometry`, die ein 3D-Modell eines Baums an den gegebenen Koordinaten darstellt. Die Größe und Ausrichtung des Baums wird dabei zufällig festgelegt.

ObstacleExtrudeGeometry

`void create (string name, List<Vector2> verticesList, float height)`

Unterklasse von `Geometry`, die ein Hindernis oder oder Gebäude darstellt. Als Grundfläche wird ein Polygon übergeben, das dann entsprechend der Höhe extrudiert wird.

WallExtrudeGeometry

`void create (string name, List<Vector2> verticesList, float height, float width)`

Unterklasse von `Geometry`, die einen Linienzug als Wand darstellt. Dazu wird der Linienzug um die angegebenen Werte in der Höhe und Breite extrudiert.

A.5 Verwendete Komponenten von Drittanbietern

Vectrosity von Starscene Software

Bibliothek zum einfachen Zeichnen von Vektorlinien. Wird verwendet um Trajektorien und Diagramme zu zeichnen.

`http://www.starscenesoftware.com/vectrosity.html`

Triangulator von Rune Skovbo Johansen

Skript zur Zerlegung von Polygonen in einzelne Dreiecke. Wird zur Erzeugung der Geometrie benötigt.

`http://wiki.unity3d.com/index.php?title=Triangulator`

TangentSolver von Kyle Gibbar

Berechnet die Tangenten eines Mesh, was von einigen Shadern zur Darstellung benötigt wird.

`http://answers.unity3d.com/questions/7789/calculating-tangents-vector4.html`

FlyCam Extended von Desi Quintans

Kamera-Skript, dass eine freibewegliche und steuerbare Kamera ermöglicht.

`http://wiki.unity3d.com/index.php/FlyCam_Extended`

HUDFPS von Aras Pranckevicius

Anzeige der "frames per second" (FPS).

`http://wiki.unity3d.com/index.php?title=FramesPerSecond`

MaleCharacterPack Carl von Mixamo Software

Nicht-animiertes 3D-Modell einer männlichen Person inklusive Texturen. Wird als Modell für alle Fußgänger verwendet. Die Animation des Gangzyklus wurde stälbstständig auf diesem Modell implementiert.

`http://u3d.as/content/mixamo/male-character-pack`

Anhang B

Quellcode zu SumoViz Unity

Auf der beigefügten CD befindet sich folgender Inhalt:

- Unity-C#-Quellcode für SumoViz Unity
- Beispieldatensätze für Fußgänger- und Geometriedaten
- diese Arbeit im PDF-Format

Zusätzlich kann der Quellcode der Arbeit auch unter folgender Adresse abgerufen werden: `https://github.com/danielbuechele/SumoVizUnity`

Abbildungsverzeichnis

Literaturverzeichnis

[AHR+93] Klaus Aerni, Edith Höfliger, Ruth Kalbermatten Rieder, Urs Kaufmann, and Ueli Seewer. Fussgänger als wichtigste Verkehrsteilnehmer: Eine Fussgängeruntersuchung in der Berner Innenstadt. *disP-The Planning Review*, 29(113):3–10, 1993.

[Ant92] Franklin Antonio. Faster line segment intersection. In *Graphics Gems III*, pages 199–202. Academic Press Professional, Inc., 1992.

[Aur91] Franz Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

[Ber05] Robert Berggren. *Simulating crowd behaviour in computer games*. PhD thesis, BSc dissertation. Luleå University of Technology, 2005.

[BFL00] Paul Brinckmann, Wolfgang Frobin, and Gunnar Leivseth. *Orthopädische Biomechanik*. Georg Thieme Verlag, 2000.

[BG95] Bruce M Blumberg and Tinsley A Galyean. Multi-level direction of autonomous creatures for real-time virtual environments. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 47–54. ACM, 1995.

[BKSZ01] Carsten Burstedde, Kai Klauck, Andreas Schadschneider, and Johannes Zittartz. Simulation of pedestrian dynamics using a two-dimensional cellular automaton. *Physica A: Statistical Mechanics and its Applications*, 295(3):507–525, 2001.

[BM10] Hans Jochen Blätte and Detlef Mamrot. Loveparade Duisburg 2010. *Zeitschrift für Forschung, Technik und Management im Brandschutz*, 59(4):193–195, 2010.

[Büc12] Daniel Büchele. Webbasierte 3D-Postvisualisierung von Fußgängersimulationsdaten. Master's thesis, Technische Universität München, 2012.

[CC00] Nigel P Chapman and Jenny Chapman. *Digital multimedia*, volume 4. Wiley, 2000.

[CIU] Can I use WebGL?
`http://caniuse.com/webgl`.

[CJ03] Thomas J Cova and Justin P Johnson. A network flow model for lane-based evacuation routing. *Transportation Research Part A: Policy and Practice*, 37(7):579–604, 2003.

[CJ12] Diego Cantor and Brandon Jones. *WebGL Beginner's Guide*. Packt Publishing Ltd, 2012.

[DeL11] Mark DeLoura. Game engine survey 2011. *Game Developer Magazine*, pages 7–12, May 2011.

[DFWB13] Toby P Davies, Hannah M Fry, Alan G Wilson, and Steven R Bishop. A mathematical model of the London riots and their policing. *Scientific reports*, 3, 2013.

[DH03] Winnie Daamen and Serge P Hoogendoorn. Experimental research of pedestrian walking behavior. *Transportation Research Record: Journal of the Transportation Research Board*, 1828(1):20–30, 2003.

[Ebe98] David Eberly. Triangulation by ear clipping. *Geometric Tools, LLC. http://www. geometrictools. com*, 1998.

[Ebe06] David H Eberly. *3D game engine design: a practical approach to real-time computer graphics*. CRC Press, 2006.

[ET08] Niklas Elmqvist and Philippas Tsigas. A taxonomy of 3d occlusion management for visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 14(5):1095–1109, 2008.

[Eve12] Joey Everett. Maya Topology. `http://joeyeverettgad.blogspot.de/2012_02_01_archive.html`, 2012.

[FK04] Dieter Fritsch and Martin Kada. Visualisation using game engines. In *ISPRS commission*, volume 5, pages 621–625, 2004.

[For87] Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2(1-4):153–174, 1987.

[Fry] Hannah Fry. The London riots of 2011. `http://hannahfry.co.uk/2013/02/21/the-london-riots-of-2011/`.

[Fry14] Hannah Fry. I predict a riot. In *re:publica 2014*, 2014.

[Gau05] Jean-Marc Gauthier. *Building Interactive Worlds in 3D: Virtual Sets and Pre-Visualization for Games, Film, and the Web*. Taylor & Francis, 2005.

[GKL08] Matthias Gather, Andreas Kagermeier, and Martin Lanzendorf. *Geografische Mobilitäts- und Verkehrsforschung*. 2008.

[GM85] P.G. Gipps and B. Marksjö. Cellular automata microsimulation for modeling bi-directional pedestrian walkways. *Mathematics and Computers in Simulation*, 27(2-3):95–105, April 1985.

[Gre09] Jason Gregory. *Game engine architecture*. CRC Press, 2009.

[GRHS04] Jacob Goldberger, Sam Roweis, Geoff Hinton, and Ruslan Salakhutdinov. Neighbourhood components analysis. 2004.

[Han12] Kein Ende der Ermittlungen in Sicht. `http://www.handelsblatt.com/politik/deutschland/loveparade-katastrophe-kein-ende-der-ermittlungen-in-sicht/6194648.html`, 2 2012.

[HB04] Serge P Hoogendoorn and Piet HL Bovy. Pedestrian route-choice and activity scheduling theory and models. *Transportation Research Part B: Methodological*, 38(2):169–190, 2004.

[HD05] Serge P Hoogendoorn and Winnie Daamen. Pedestrian behavior at bottlenecks. *Transportation Science*, 39(2):147–159, 2005.

[HDB03] Serge P Hoogendoorn, Winnie Daamen, and Piet HL Bovy. Extracting microscopic pedestrian characteristics from video data. In *Transportation Research Board 2003 Annual Meeting, CD-ROM, Paper*, number 477, 2003.

[Heg00] Günther Hegewald. *Ganganalytische Bestimmung und Bewertung der Druckverteilung unterm Fuß und von Gelenkwinkelverläufen*. PhD thesis, Humboldt-Universität zu Berlin, Philosophische Fakultät IV, 2000.

[Hen74] LF Henderson. On the fluid mechanics of human crowd motion. *Transportation research*, 8(6):509–515, 1974.

[HFMV02] Dirk Helbing, I Farkas, Péter Molnár, and Tamás Vicsek. Simulation von Fußgängermengen in normalen Situationen und im Evakuierungsfall. *Vortrag bei der AGE Zentralveranstaltung*, 2002.

[HM95] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[Hoo01] Serge Paul Hoogendoorn. *Normative Pedestrian Flow Behavior, Theory and Applications*. Delft University of Technology, Faculty of Civil Engineering and Geosciences, Transportation and Traffic Engineering section, 2001.

[HP02] Adrian Herwig and Philip Paar. Game engines: tools for landscape visualization and planning. *Trends in GIS and Virtualization in Environmental Planning and Design*, pages 161–172, 2002.

[HSS98] Wolfgang Heidrich, Philipp Slusallek, and Hans-Peter Seidel. Real-time generation of continuous levels of detail for height fields. In *Proc. WSCG'98*, pages 315–322, 1998.

[HW58] BD Hankin and RA Wright. Passenger flow in subways. *OR*, pages 81–88, 1958.

[Isi12] Mustafa K. Isik. SumoViz - HTML5-based Visualization of Pedestrian Simulation Data. Master's thesis, Technische Universität München, 2012.

[ISS09] Karim Ismail, Tarek Sayed, and Nicolas Saunier. Automated collection of pedestrian data using computer vision techniques. In *Transportation Research Board Annual Meeting Compendium of Papers, Washington, DC*, 2009.

[JARLP12] Asja Jelić, Cécile Appert-Rolland, Samuel Lemercier, and Julien Pettré. Properties of pedestrians walking in line. II. Stepping behavior. *Physical Review E*, 86(4):046111, 2012.

[JSC06] Kyungkoo Jun, Meeyoung Sung, and Byoungjo Choi. Steering behavior model of visitor NPCs in virtual exhibition. In *Advances in Artificial Reality and Tele-Existence*, pages 113–121. Springer, 2006.

[JZWL10] Yan-qun Jiang, Peng Zhang, SC Wong, and Ru-xun Liu. A higher-order macroscopic model for pedestrian flows. *Physica A: Statistical Mechanics and its Applications*, 389(21):4623–4635, 2010.

[KHW01] Jon Kerridge, Julian Hine, and Marcus Wigan. Agent-based modelling of pedestrian movements: the questions that need to be asked and answered. *Environment and Planning B*, 28(3):327–342, 2001.

[Kir02] Ansgar Kirchner. *Modellierung und statistische Physik biologischer und sozialer Systeme*. PhD thesis, PhD Thesis (in german), 2002.

[KKR+08] Hubert Klüpfel, Tobias Kretz, Christian Rogsch, Andreas Schadschneider, and Armin Seyfried. ped-net.org.
`http://www.ped-net.org`, 2008.

[KKS] Andreas Keßel, Hubert Klüpfel, and Michael Schreckenberg. Simulation der Bewegung von Fußgängern, Menschenmengen und Evakuierungsprozessen.

[Kne13] Angelika Kneidl. *Methoden zur Abbildung menschlichen Navigationsverhaltens bei der Modellierung von Fußgängerströmen*. PhD thesis, Technische Universität München, 2013.

[KR03] Christoph Kinkeldey and Martin Rose. Fußgängersimulation auf der Basis sechseckiger zellularer Automaten. In *Forum Bauinformatik*, 2003.

[KRW+03] Martin Kada, Stefan Roettger, Karsten Weiss, Thomas Ertl, and Dieter Fritsch. Real-time visualisation of urban landscapes using open-source software. In *Proceedings of ACRS*, 2003.

[Küh04] R Kühne. Das Fundamentaldiagramm–Grundlagen und Anwendungen. *FGSV Merkblatt (Entwurf)*, page 59, 2004.

[LJ02] Michael Lewis and Jeffrey Jacobson. Game engines. *Communications of the ACM*, 45(1):27, 2002.

[LKF05] Taras I Lakoba, David J Kaup, and Neal M Finkelstein. Modifications of the Helbing-Molnar-Farkas-Vicsek social force model for pedestrian evolution. *Simulation*, 81(5):339–352, 2005.

[LL08] Zhen Liu and Yu-Sheng Lu. A motivation model for virtual characters. In *Machine Learning and Cybernetics, 2008 International Conference on*, volume 5, pages 2712–2717. IEEE, 2008.

[Løv94] Gunnar G Løvås. Modeling and simulation of pedestrian traffic flow. *Transportation Research Part B: Methodological*, 28(6):429–443, 1994.

[Mat09] Markus Matern. Voronoi diagram.
`http://commons.wikimedia.org/wiki/File:Voronoi_diagram.svg#mediaviewer/Datei:Voronoi_diagram.svg`, 11 2009.

[MFCGD99] Franck Multon, Laure France, Marie-Paule Cani-Gascuel, and Gilles Debunne. Computer animation of human walking: a survey. *The journal of visualization and computer animation*, 10(1):39–54, 1999.

[Mil99] Vito Miliano. Unreality: application of a 3D game engine to enhance the design, visualization and presentation of commercial real estate. In *Proceedings of 1999 International Conference on Virtual Systems and MultiMedia (VSMM'99)*, pages 508–513, 1999.

[MIN99] Masakuni Muramatsu, Tunemasa Irie, and Takashi Nagatani. Jamming transition in pedestrian counter flow. *Physica A: Statistical Mechanics and its Applications*, 267(3):487–498, 1999.

[MMG06] Bruce Merry, Patrick Marais, and James Gain. Animation space: A truly linear framework for character animation. *ACM Transactions on Graphics (TOG)*, 25(4):1400–1423, 2006.

[Moz13] 'Epic Citadel' Demo Shows the Power of the Web as a Platform for Gaming. `https://blog.mozilla.org/futurereleases/2013/05/02/epic-citadel`, 2013.

[MPG+10] Mehdi Moussaïd, Niriaska Perozo, Simon Garnier, Dirk Helbing, and Guy Theraulaz. The walking behaviour of pedestrian social groups and its impact on crowd dynamics. *PloS one*, 5(4):e10047, 2010.

[Ngu12] Minh-Quang Nguyen. Erweiterung einer web-basierten Postvisualisierung zur Darstellung und Analyse von Fußgängersimulationsergebnissen. Master's thesis, Technische Universität München, 2012.

[NHP06] Seung Seok Noh, Sung Dea Hong, and Jin Wan Park. Using a game engine technique to produce 3D Entertainment contents. In *Artificial Reality and Telexistence–Workshops, 2006. ICAT'06. 16th International Conference on*, pages 246–251. IEEE, 2006.

[NTH85] Johnny Nilsson, Alf Thorstensson, and JÑ HALBERTSMA. Changes in leg movements and muscle activity with speed of locomotion and mode of progression in humans. *Acta Physiologica Scandinavica*, 123(4):457–475, 1985.

[RBS+04] Albert A Rizzo, Todd Bowerly, Cyrus Shahabi, J Galen Buckwalter, Dean Klimchuk, and Roman Mitura. Diagnosing attention disorders in a virtual classroom. *Computer*, 37(6):87–89, 2004.

[Reu11] Reuters. Factbox: A look at the 65 billion dollar video games industry. `http://uk.reuters.com/article/2011/06/06/us-videogames-factbox-idUKTRE75552I20110606`, 2011.

[Rey99] Craig W Reynolds. Steering behaviors for autonomous characters. In *Game developers conference*, volume 1999, pages 763–782, 1999.

[RR02] Klaus Richter and Jan-Michael Rost. *Komplexe Systeme: Chaos, Selbstorganisation, zelluläre Automaten, Spiel des Lebens, granulare Systeme, Musterbildung, mesoskopische Quantensysteme, Fraktale, Phasenübergänge, Skaleninvarianz, logische Tiefe.* Fischer-Taschenbuch-Verlag, 2002.

[Run98] Martin Runge. Ganganalyse. 1998.

[Sha06] Wei Shao. *Animating autonomous pedestrians*. PhD thesis, New York University, 2006.

[SKK+09] Andreas Schadschneider, Wolfram Klingsch, Hubert Klüpfel, Tobias Kretz, Christian Rogsch, and Armin Seyfried. Evacuation dynamics: Empirical results, modeling and applications. In *Encyclopedia of complexity and systems science*, pages 3142–3176. Springer, 2009.

[SOHTG99] Thorsten Schelhorn, David O'Sullivan, Mordechay Haklay, and Mark Thurstain-Goodwin. STREETS: An agent-based pedestrian model. 1999.

[SS10] Bernhard Steffen and Armin Seyfried. Methods for measuring pedestrian density, flow, speed and direction with minimal scatter. *Physica A: Statistical mechanics and its applications*, 389(9):1902–1910, 2010.

[Str11] David Strippgen. Game Engines Vergleich am Beispiel UDK und Unity. `http://home.htw-berlin.de/~strippg/?q=node/10`, 2011.

[Tri] Trinigy.net. `http://www.trinigy.net`.

[TS08] David Trenholme and Shamus P Smith. Computer game engines for developing first-person virtual environments. *Virtual reality*, 12(3):181–187, 2008.

[TTI00] Kardi Teknomo, Yasushi Takeyama, and Hajime Inamura. Review on microscopic pedestrian simulation model. In *Proceedings Japan Society of Civil Engineering Conference*. Citeseer, 2000.

[Uni] Unity Documentation. `https://docs.unity3d.com`.

[Uni14] Unity Sim Viz. `http://unity3d.com/industries/sim`, 2014.

[Unra] About Unreal Engine 4. `https://www.unrealengine.com/products/unreal-engine-4`.

[Unrb] Unreal Engine 4 Documentation. `https://docs.unrealengine.com/latest/INT/`.

[Unr14] Unreal Engine Showcase. `https://www.unrealengine.com/showcase`, 3 2014.

[Unt06] Rembert Unterstell. Die Dynamik der Panik. *forschung*, 31(2):17–19, 2006.

[Vat92] Bala R Vatti. A generic solution to polygon clipping. *Communications of the ACM*, 35(7):56–63, 1992.

[Wat93] Alan Watt. *3D computer graphics*. Addison-Wesley Longman Publishing Co., Inc., 1993.

[Wei92] Ulrich Weidmann. Transporttechnik der fußgänger. 1992.

[Wei93] Ulrich Weidmann. *Transporttechnik der Fussgänger: Transporttechnische Eigenschaften des Fussgängerverkehrs (Literaturauswertung)*. ETH, IVT, 1993.

[WLG03] Jijun Wang, Michael Lewis, and Jeffrey Gennari. A game engine based simulation of the NIST urban search and rescue arenas. In *Simulation Conference, 2003. Proceedings of the 2003 Winter*, volume 1, pages 1039–1045. IEEE, 2003.