Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

# Entwicklung eines 2D Charakter-Animationssystemes für automatische Laufbewegungen

Development of a 2D Character Animation System for Automatic Walking Movements

Daniel Track

Bachelor-Abschlussarbeit

Betreuer: Prof. Dr. Christoph Lürig

Trier, 20.10.2020

# Kurzfassung

Diese Arbeit beschreibt die Entwicklung eines Systems, das die dynamische Animation der Laufbewegungen von zweidimensionalen Charakteren ermöglicht. Dabei wird zuerst ein Charaktermodell als Skelett erstellt und mit einer Textur versehen. Das Programm erstellt dann anhand der Eingaben des Nutzers und der Daten des Skeletts neue Bewegungsabläufe für jeden einzelnen Schritt des Charakters. Die so erstellten Animationen ermöglichen Bewegungen in präzise an die Eingabe des Spielers angepasster Geschwindigkeit und über Untergründe verschiedener Höhen. Dazu werden Hermite Splines für die Hände, Füße und das Becken des Charakters berechnet, denen die entsprechenden Punkte des Körpers dann folgen. Die Splines sind an den Boden unter dem Spieler angepasst sind und variieren die Schrittweite und -form je nach geforderter Bewegungsgeschwindigkeit. Außerdem werden die Splines basierend auf einer Reihe von Spline-Prototypen gebildet, die zuvor vom Nutzer festgelegt werden können. So soll eine große Anpassbarkeit der Animationen an verschiedene Charaktere und Anwendungsfälle ermöglicht werden. Es werden einige Beispiele von Animationen gezeigt, die von dem System erstellt wurden, um die Variation der Bewegungen und den Umgang mit verschiedenen Situationen zu demonstrieren.

This thesis describes the development of a system for the dynamic animation of walking movements for two-dimensional characters. To achieve this, the character model is created as a skeleton and provided with a texture. The application then uses the user's input and the data of the skeleton to generate new animations for every single step of the character. Animations created this way allow for movements with a speed that is fitted precisely to the player's input and that are also able to adapt to differences in ground height. In order to do this, the system creates Hermite splines for the hands, feet and pelvis of the character that the respective body parts follow. These splines are adjusted to the floor below the player and vary the step distance and shape depending on the required walking speed. Furthermore, these splines are based on a set of prototype splines that the user may set beforehand. This allows for customization of the created animations

in order to fit them to different characters and use cases. The thesis shows some examples of animations created with the presented system to showcase different possible variations of movements and the adaptation to various situations.

# Inhaltsverzeichnis

# 1

# Einleitung

Menschen sind überall um uns herum. Die meisten von uns leben und arbeiten täglich mit anderen Mitgliedern der eigenen Spezies und so ist es kein Wunder, dass wir ihren Bewegungsabläufen extrem vertraut sind. Auch in Videospielen kontrolliert der Spieler zumeist Menschen oder zumindest humanoide Gestalten, die stark an sie angelehnt sind. Der enorme Menge an Erfahrungen mit menschlichen Bewegungsabläufen, die die meisten Spieler haben, sorgt dabei aber auch für große Erwartungen an die Repräsentation eben dieser Bewegungen in Spielen, da kleine Ungereimtheiten besonders schnell auffallen können. Das stellt das Feld der Computeranimation vor große Herausforderungen.

Diese Arbeit soll versuchen, durch die Entwicklung eines Animationssystems für zweidimensionale Charaktere einen Beitrag zum Überkommen dieser Herausforderungen zu leisten.

## 1.1 Motivation

Obwohl die enorme Leistungsfähigkeit moderner Computer es erlaubt, große, dreidimensionale Welten in Videospielen immer realistischer darstellen zu können, erscheinen noch immer sehr viele zweidimensionale Spiele, die das große Potenzial dieser Maschinen nicht ausreizen. Dafür mag es eine Reihe von Gründen geben, angefangen bei der Nostalgie vieler Spieler für die 2D-Spiele älterer Konsolengenerationen bis hin zu den meist wesentlich komplexeren mathematischen Verfahren, die zur Programmierung von dreidimensionalen Anwendungen verwendet werden. In vielen Fällen wird es jedoch auch damit zu tun haben, dass die Erstellung von 3D-Modellen und deren Animation besonders kleine Entwicklerstudios vor eine große Herausforderung stellt. 2D-Assets hingegen können recht leicht mit einem der vielen verfügbaren Bildbearbeitungsprogramme erstellt oder sogar per Hand gezeichnet und digitalisiert werden.

Einen entscheidenden Vorteil bieten 3D-Modelle jedoch gegenüber 2D-Sprites: Zur Animation werden 3D-Objekte meist auf ein Skelett gespannt, für dessen Knochen dann Zielstellungen angegeben werden. Das Modell folgt dann der Bewegung der Knochen. Bei 2D-Modellen hingegen wird häufig jeder einzelne Schritt der Animation zuvor erstellt. Auch hier gibt es Verfahren, mit denen Computer den

Grafikdesignern Arbeit abnehmen können; trotzdem wird meist eine beachtliche Anzahl an Animationsframes per Hand erstellt. Viele der 2D-Spiele sind deshalb außerdem auf eine Framerate von 60 Bildern pro Sekunde limitiert. Monitore mit höheren Framerates werden aber zunehmend günstiger und populärer, und 120, 144 oder gar 240 FPS zu unterstützen stellt für die gängigen 2D-Animationssysteme eine schwierige Herausforderung dar, während es bei den meisten 3D-Anwendungen hauptsächlich eine Frage der Hardware ist.

Ein weiterer Vorteil der gängigen 3D-Animationsverfahren ist, dass genaue Zielpunkte für Bewegungen während der Laufzeit des Programms errechnet und Animationen dann so ausgeführt werden können, dass sie diese Punkte exakt treffen (z.B. wenn ein Charakter ein Objekte mit seiner Hand greift). Mit 2D-Animationen, deren Bilder vor der Laufzeit des Programms gezeichnet werden, ist dies nicht möglich.

Eine Übertragung des Verfahrens der skelettbasierten Animation auf 2D-Charaktere könnte dabei helfen, sowohl den Arbeitsaufwand bei der Erstellung von Animationen zu minimieren als auch neue Arten von Animationen zu ermöglichen, die erst zur Laufzeit des Programms gebildet werden.

## 1.2 Zielsetzung

Um die Vorteile der zuvor beschriebenen skelettbasierten Animierung von 2D-Charakteren zu demonstrieren, soll ein System erstellt werden, dass nur einen einzelnen Sprite eines Charakters verwendet und eine Laufanimation darstellt. Um weiterhin zu zeigen, welche Möglichkeiten das Einbeziehen von Laufzeitdaten eröffnet, soll der Charakter auch über Flächen verschiedener Höhen (wie z.B. eine Treppe hinauf) laufen können und dabei sowohl die Position, auf der der Fuß aufgesetzt wird als auch die Bewegung dorthin dynamisch zur Laufzeit errechnen.

Außerdem soll sichergestellt werden, dass die Steuerung des Charakters responsiv ist. Dazu gehört, dass er schnell auf Eingaben reagiert und seine Animation das auch angemessen dem Spieler kommuniziert. Ein weiterer Aspekt ist in diesem Fall aber auch, dass der Charakter sich nicht nur in einer vorbestimmten Geschwindigkeit bewegen kann. Die dynamische Animation eröffnet die Möglichkeit, die Länge und genaue Bewegungsabfolge jedes Schrittes neu zu bestimmen. Von dieser Möglichkeit soll gebrauch gemacht werden, um ein präzise Bestimmung der Laufgeschwindigkeit zu erlauben.

Da für verschiedene Charaktere auch verschiedene Animationsverhalten nötig sind, soll es Möglichkeiten zur Anpassung der Animationen geben. Dies könnte beispielsweise über die Anpassung der Parameter geschehen, die das System zur Generierung neuer Animationen nutzt.

Eine wichtige Limitierung dabei ist, dass ausdrücklich *nicht* Ziel dieser Arbeit ist, realistische bzw. physikalisch korrekte Bewegungen zu zeigen. Auf den Charakter wirkt also keine Gravitation, er muss kein Gleichgewicht halten oder sein

Momentum abbremsen. Die Bewegung soll jedoch glaubhaft[1] aussehen, kann dabei aber durchaus stilisiert sein, wie es in der Computeranimation üblich ist.

## 1.3 Aufbau

Im Folgenden werden zuerst einige der Grundlagen erläutert. Dabei handelt es sich um Techniken und Konzepte, die häufig in der Computeranimation zum Einsatz kommen und auch von dem hier vorgestellten System verwendet werden. Daraufhin legt Kapitel 3 den aktuellen Stand der Forschung in den relevanten Bereichen dar. Dabei wird sowohl auf die prozedurale generation von Laufanimationen als auch auf die Animation von 2D-Charakteren mithilfe von Skeletten eingegangen. Außerdem wird die hier vorliegende Arbeit in den Kontext der bestehenden Forschung eingeordnet.

Das entwickelte Animationssystem und dessen Implementierung wird dann in Kapitel 4 vorgestellt. Dazu werden zuerst konzeptionelle Überlegungen dargelegt, gefolgt von einer kurzen Erläuterung der vom Projekt verwendeten Programmbibliotheken. Abschnitt 4.3 beschreibt dann zunächst den generellen Aufbau des Programms, bevor in 4.4 die Details des entwickelten Animationssystems ausführlich erklärt werden.

Im darauf folgenden Kapitel werden die Ergebnisse besprochen und einige Beispiele von Animationen gezeigt, die mit dem System erstellt wurden. Abschließend wird in Kapitel 6 dann ein Fazit gezogen und Möglichkeiten besprochen, wie die Arbeit noch verbessert werden kann.

[1] Im Sinne von Bates et al.[B+94]: Glaubhaft ist ein Charakter, wenn das Publikum ihn akzeptiert, als wäre er lebendig. Realistisch ist er, wenn er der tatsächlichen Realität entspricht.

# 2

# Grundlagen

## 2.1 Skelettbasierte Animation

Die skelettbasierte Animation ist ein weit verbreitetes Verfahren zur Animation von 3D-Charakteren. Dabei werden für jeden Charakter zwei verschiedene Repräsentationen erstellt: Die erste besteht aus einem Dreiecksnetz (Skin), stellt die Oberfläche des Modells dar und wird zur visuellen Darstellung verwendet. Hinzu kommt eine Menge von Knochen, die nicht sichtbar sind und nur zur Animation des Dreiecksnetzes verwendet werden. Diese Knochen bilden eine hierarchische Datenstruktur und mit Ausnahme des Root-Knochens bestehen sie alle aus den folgenden drei Werten:

1. Ein Parent-Knochen
2. Eine dreidimensionale Transformationsmatrix relativ zum Parent (Bind-Pose Transform)
3. Eine Rotation

Der Root-Knochen hat keinen Parent, seine Transformationsmatrix stellt eine Transformation vom globalen Koordinatensystem ins Koordinatensystem des Knochens dar. Durch die Rotation der Knochen können neue Posen für den Charakter gebildet werden. Dabei wird die Rotation im lokalen Koordinatensystem des jeweiligen Knochens angewendet und beeinflusst so Position aller Knochen weiter unten in der Hierarchie. Jeder Vertex des Skins ist einem (Rigid Binding) oder mehreren Knochen (Smooth Binding) zugeordnet. Zur Bestimmung seiner neuen Position wird der Vertex in das Koordinatensystem der entsprechenden Knochen transformiert, dort rotiert und dann wieder ins Ursprungskoordinatensystem zurück transformiert. Um einen Vertex $v$ in eine beliebige Pose zu transformieren gelten die Formeln:

$$\hat{v} = \prod_{i=0}^{k} C_i (\prod_{i=0}^{k} B_i)^{-1} p = J_k v \tag{2.1a}$$

$$\hat{v} = \sum_{i=0}^{n} w_i J_i v \qquad \text{mit } \sum_{i=0}^{n} w_i = 1 \tag{2.1b}$$

Dabei stellen $B_i^{-1}$ die Inverse Bind-Pose Transformationen der Knochen und $C_i$ die Transformationen für die aktuelle Pose dar. Die Gleichung in 2.1a beschreibt das Verfahren beim Rigid Binding. Beim Smooth Binding (2.1b) werden Joint-Matrizen für alle relevanten Knochen berechnet, die dann mit deinem Gewichtungsfaktor $w_i$ auf den Vertex wirken.

## 2.2 Inverse Kinematics

Bei Inverse Kinematics handelt es sich um mathematische Prozesse zur Berechnung von Gelenkparametern, um das Ende einer kinematischen Kette (wie z.B. eine Hand) an einem bestimmten Punkt zu positionieren. Zur Lösung dieser Problemstellungen gibt es verschiedene analytische und numerische Verfahren. Welches dieser Verfahren gewählt werden sollte, hängt von den Details des vorliegenden Problems ab. Da in dieser Arbeit nur die Hände und Füße des Charakters mittels Inverse Kinematics positioniert werden sollen, wird lediglich ein simples analytisches Verfahren verwendet, das bei der Erläuterung des entsprechenden Programmcodes im Detail erklärt wird.

## 2.3 Hermite Splines

Als kubische Hermitesplines bezeichnet man Splines, die zwischen zwei oder mehr Punkten interpolieren. Sie sind sowohl $C_1$ als auch $C_2$ stetig, was bedeutet, dass die Kurve weich (ohne einen Knick) von Segment zu Segment übergeht. Ein Hermitespline wird dabei durch seine Kontrollpunkte und die Tangenten in den jeweiligen Punkten bestimmt. Der Verlauf des Splines zwischen diesen Punkten ist durch ein Polynom dritten Grades definiert.

# 3

# Forschungsstand

Das folgende Kapitel gibt einen Überblick über bestehende Literatur, die für das hier vorgestellte Animationssystem relevant ist. Zuerst werden dabei die Arbeiten zu prozeduralen Animationssystemen vorgestellt. Da sich die meisten dieser Beiträge mit Bewegungen im dreidimensionalen Raum beschäftigen, können ihre Ergebnisse leider nur beschränkt auf diese Arbeit übertragen werden. Im zweiten Teil wird dann der Forschungsstand zur skelettbasierten Animation von zweidimensionalen Objekten erläutert. Zuletzt wird dann beschrieben, wie sich die hier vorliegende Arbeit in den Kontext der bestehenden Forschung einfügt und welchen Beitrag sie leisten soll.

## 3.1 Prozedurale Laufanimation

Da Laufbewegungen einen der wohl häufigsten Bewegungsabläufe von Menschen darstellen, gibt es eine lange Reihen von Arbeiten zum Problem der Animation von menschlichen Laufbewegungen. In vielen Fällen werden die beschriebenen Animationen dabei einmalig erstellt und dann genau so abgespielt und aufgezeichnet (beispielsweise für computergenerierte Charaktere in einem Film). Da Videospiele ein interaktives Medium sind, ist es zur Anwendung in Spielen jedoch oft wichtig, dass die Animationen noch zur Laufzeit des Programms an das gewünschte Verhalten angepasst werden können. Grob können drei verschiedene Ansätze verwendet werden, um Bewegungen zur Laufzeit zu erstellen: Prozedurale Bewegung (auch algorithmische Bewegung), beispielbasierte Bewegung und dynamisch-simulierte Bewegung [Joh09].

Systeme zur prozeduralen Bewegung verwenden ein Verständnis für die Bewegungsabläufe von Charakteren, um neue Bewegungen spontan generieren zu können. Beispielbasierte Ansätze hingegen schöpfen aus einer Reihe von zuvor erstellten Animationen und kombinieren diese, um neue Bewegungsabläufe zu erstellen. Bisweilen kommen dabei auch noch prozedurale Techniken zum Einsatz, um die kombinierten Animationen weiter an die benötigte Situation anzupassen. Dynamische Simulationen hingegen berechnen ein physikalisches Modell des Charakters und seiner Umgebung und versuchen so, realistische Bewegungen zu erstellen. Als Grundlage können dabei auch Beispielanimationen oder prozedural erstellte Bewegungen verwendet werden.

Einen Ansatz zur prozeduralen Generierung von Laufbewegungen in Echtzeit beschrieben Bruderlin und Calvert[Bru93][BC96]. Sie simulierten die grundlegende Bewegung der Beine als wären sie ein umgekehrtes Pendel und approximierten die Flugbahn des Torsos in der Luft mit einer Parabel. Ihr System bietet dem Nutzer eine große Anzahl von Variablen zur Beeinflussung der Animation und ermöglicht so viele verschiedene, ausdrucksstarke Bewegungen, trotz ihrer rein synthetischen Natur. Es sind aber nur Bewegungen auf ebenen Untergründen möglich.

Ein weiteres Team um Bruderlin[BTC94] entwickelte das *Life Forms* System zur Animation von menschlichen Figuren im dreidimensionalen Raum. Sie verwenden dabei Inverse Kinematics, um die Figuren mit ihren Händen nach Objekten greifen zu lassen. Die Bewegungen der anderen daran beteiligten Körperteile wie Arm und Schulter werden dabei durch die Knochen und die für diese zuvor festgelegte Beschränkungen errechnet. Außerdem bietet ihre Anwendung die Möglichkeit zur prozeduralen Erstellung von Laufbewegungen. Dabei werden als Hauptparameter die Schrittweite und Schrittfrequenz (und daraus resultieren die Laufgeschwindigkeit) vom Nutzer angegeben. Das Programm erstellt dann vier Kontrollpunkte für einen Schritt in der Laufbewegung: Den Anfang, das Ende und dazwischen die beiden Zeitpunkte, an denen die Hüfte am höchsten bzw. am niedrigsten ist. Daraufhin wird eine kubische Spline-Kurve berechnet, die durch diese vier Punkte verläuft. Die Position der Hüfte des Modells wird dann entlang dieser Kurve interpoliert und die Stellung der Beine dazu passend berechnet.

Diesem Problem des unebenen Terrains widmeten sich Chung und Kollegen[CH99]. Die von ihnen entwickelten Algorithmen ermöglichen die automatische Bewegung über Untergründe verschiedener Höhe, während die verwendete Collision Detection dafür sorgt, dass die Beine dabei nicht durch Hindernisse bewegt werden. Außerdem gibt es ein hierarchisches System von Stellschrauben, die zur Justierung der Animation genutzt werden können.

Johansen[Joh09] entwickelte ein System zur Fortbewegungsanimation von 3D-Charakteren, das prozedurale Verfahren mit beispielbasierten Ansätzen verbindet. Er nutzte dabei nur zwei verschiedene Beispielanimationen – bei einer läuft der Charakter geradeaus, bei der anderen bewegt er sich zur Seite – unter der Annahme, dass sich aus diesen beiden Bewegungen alle anderen synthetisieren lassen würden. Die Resultate dieses Schritts werden dann prozedural mittels Inverse Kinematics an die Höhe und den Winkel des Untergrunds angepasst, sodass der Charakter auch über Stufen oder schräge Flächen laufen kann. Die Ergebnisse dieses Systems wirken dabei durchweg glaubhaft.

Auf dieser Arbeit aufbauend versuchte Shapiro[Sha11], variablere, ausdrucksstärkere Animationen mit einem ähnlichen Funktionsspektrum zu erreichen, indem er dem System mit 19 Exemplaren eine größere Menge an Beispielanimationen zur Verfügung stellte, um den prozeduralen Bestandteil und Inverse Kinematics aus dem Verfahren entfernen zu können (und trotzdem noch alle verschiedenen Bewegungsmöglichkeiten abzudecken). Er beschreibt die Resultate seiner Arbeit als realistischer, die Erstellung der vielen Beispielanimationen sorgt aber für einen wesentlich größeren Aufwand, wenn das System in Projekten verwendet werden soll.

## 3.2 Skelettbasierte Animation von 2D-Objekten

Einen der frühen Versuche, skelettbasierte Animation für die Bewegungen zweidimensionaler Charaktere zu verwenden, stellten Burtnyk und Wein[BW76] vor. Bis dahin wurden häufig Keyframes von Animationen gezeichnet und die restlichen Bilder durch Interpolation dieser Keyframes erzeugt. Dieses System erlaubte jedoch nur geradlinige Bewegung in gleich großen Schritten zwischen den Keyframes. Die Autoren führten ein Skelett in das Animationssystem ein, das es erlaubte, mit verhältnismäßig wenig Aufwand eine hohe Anzahl von Keyframes zu erstellen, da nur die simple Strichmännchen-Repräsentation dieser Knochen neu gezeichnet werden musste. Das finale Bild wird dann anhand dieser Knochen in entsprechende Posen verzerrt.

Van Overveld[vO90] beschrieben ein Verfahren zur 2D-Animation vor, bei dem das Skelett vom Rest des Objekts getrennt wurde. Das Skelett bzw. einige seiner Punkte wurden dann prozedural in Echtzeit animiert und die Bewegungen der restlichen Punkte dynamisch simuliert. Die beschränkenden Kräfte auf das Skelett wurden bei dieser Simulation erst nur approximiert und später dann korrigiert, um eine Performance zu erreichen, die schnell genug für eine Darstellung in Echtzeit ist.

Auf diesem Artikel aufbauend entwickelten Aoki et al.[ASTK99] ein Verfahren zur realistischen Animation von Pflanzen, die vom Wind bewegt werden. Sie verwendeten dazu digital erstellte Bilder von Pflanzen und versahen sie per Hand mit Skeletten. Diesen Skeletten wurden dann mechanische Eigenschaften zugewiesen, mit denen die Simulation im letzten Schritt arbeitete. Sie beschreiben ihre Ergebnisse als sehr realistisch, mehr als zwei im Artikel enthaltene Beispielbilder konnten aber nicht gefunden werden.

Ein noch nicht lange zurückliegendes Paper von Pangesti und Kollegen[PUS19] beschreibt die Entwicklung eines Verfahrens zur einfacheren Erstellung von Laufanimationen für 2D-Charaktere mithilfe eines Skeletts und Inverse Kinematics. Außerdem war ein weiteres erklärtes Ziel ihrer Arbeit, eine Animation für das Umkehren während des Laufens zu erzeugen. Durch die Analyse von Laufbewegungen echter Menschen bestimmten sie die Freiheitsgrade der verschiedenen Knochen und wendeten diese auf das virtuelle Skelett an. So wird automatisiert verhindert, dass beispielsweise ein Ellenbogen in die falsche Richtung geknickt wird. Ein Animator kann dann Keyframes festlegen, denen der Körper des Modells folgen soll, ohne dass dadurch Fehlstellungen entstehen können. Die Arbeit demonstriert dieses Verfahren zuletzt dann noch mit einer Animation für das Umkehren des Charakters während der Laufbewegung. Bemerkenswert dabei ist, dass das verwendete Modell dreidimensional ist und dann gerendert wird, um die Frames für die finale, zweidimensionale Animation zu erzeugen.

## 3.3 Einordnung in die bestehende Literatur

Um die Animationen des Charakters präzise an die Gegebenheiten des Untergrunds anzupassen, auf dem der Charakter sich bewegt, ist entweder ein (teilwei-

ser) prozeduraler Ansatz oder eine dynamische Simulation vonnöten. Da dynamische Simulationen aufgrund ihrer physikalischen Modelle meist einen höheren Rechenaufwand mit sich bringen und außerdem nur realistische Bewegungsabläufe generieren können (was nicht Ziel dieser Arbeit ist – die Bewegungen sollen nur glaubhaft aussehen), werden diese Ansätze hier nicht weiter verfolgt. Ideen der beispielbasierten Animation, speziell die Interpolation zwischen mehreren zuvor erstellten Bewegungsabläufen, könnten dabei helfen, Laufbewegungen individueller und ausdrucksstärker erscheinen zu lassen und außerdem die Komplexität des prozeduralen Systems durch ihre Bewegungsvorgaben reduzieren.

Die hier beschriebenen Beiträge zur skelettbasierten Animation von 2D-Charakteren hatten meist nicht zum Ziel, in interaktiven System verwendet zu werden. Sie sollten viel mehr Animatoren als Werkzeug zu dienen, das den Animationsprozess vereinfacht. Das in dieser Arbeit vorgestellte System soll speziell für die Anwendung in Videospielen erstellt werden. Dass die generierten Animationen durch die Eingaben des Spielers maßgeblich beeinflusst werden bedeutet dabei auch, dass die Designer nicht die Möglichkeit haben, Animationen per Hand zu korrigieren, bevor der Spieler sie zu Gesicht bekommt. Somit sollte möglichst gewährleistet werden, dass die Animationen ein vorhersagbares Verhalten zeigen. Sie sollen sich also an die Eingaben des Spielers anpassen, dabei aber nicht zu grob von der von Animator geplanten Bewegung abweichen.

Außerdem soll durch das Erstellen und darauf folgende Animieren eines einzelnen 2D-Sprites der Designaufwand minimiert werden. Pangesti et al.[PUS19] verwendeten zwar skelettbasierte Verfahren, um 2D-Aniamtionen zu erstellen, der dem zugrunde liegende Charakter ist aber ein dreidimensionales Modell, was sowohl die Erstellung des Modells als auch das Animationssystem wesentlich komplizierter macht. Einen der großen Vorteile dabei demonstrierten sie aber auch in ihrer Arbeit: Es kann eine Animation für das Umkehren des Charakters erstellt werden, bei der die Körperteile sich realistisch verdecken. Diese Möglichkeit fällt bei der Verwendung von Sprites weg, da das System keine Informationen über die Tiefenverhältnisse der Körperteile hat. Demnach stellt dies eine Limitierung des hier präsentierten Systems vor.

Da die Idee dieser Arbeit darauf basiert, Techniken aus der 3D Charakter-Animation auf 2D-Charaktere zu übertragen, können viele der Erkenntnisse aus der Forschung zur Bewegungsanimation in 3D häufig übertragen werden. Dazu zählen unter Anderem Collision Detection wie bei Chung et al.[CH99] oder die Kombination von prozeduraler Planung der Schritte mit vorgefertigten Beispielanimationen, wie sie von Johansen[Joh09] demonstriert wurde.

# 4

# Implementierung

Nachdem in den vorherigen Kapiteln theoretische Vorarbeit geleistet wurde, soll nun die Implementierung des geplanten Animationssystems erläutert werden. Begonnen wird dabei mit der konzeptionellen Planung der Software. Da diverse Bibliotheken zur Vereinfachung der Programmierarbeit verwendet werden, werden diese dann kurz beschrieben. Als nächstes folgt eine visuelle und schriftliche Darstellung des Programmaufbaus, bevor dann der Animationsprozess detailliert erklärt wird.

## 4.1 Konzeption

Da der hauptsächliche Anwendungsbereich des in dieser Arbeit beschriebenen Animationssystems Videospiele sind, werden zur Implementierung auch Werkzeuge verwendet, die in der Spieleindustrie gängig sind. Programmiert wird deshalb in C++ unter Windows und mit einem x64-Prozessor als Zielplatform. Um gute Performance des Programms zu gewährleisten soll ein möglichst großer Teil der Berechnungen auf der Grafikkarte ausgeführt werden. Dafür kommt OpenGL als Grafikbibliothek zum Einsatz und es werden Shader in GLSL geschrieben.

Um das Ziel der präzisen Bestimmung der Bewegungsgeschwindigkeit zu erreichen, wird davon ausgegangen, dass die Simulation mit einem Gamepad gesteuert wird. Die Laufgeschwindigkeit soll dann aus den Eingabedaten des Control Sticks errechnet werden.

Außerdem soll zur Synthese von Animationen für die verschiedenen Laufgeschwindigkeiten ein Verfahren zum Einsatz kommen, das aus zwei zuvor festgelegten Animationen eine neue erstellt, die auf die auf die geforderte Situation passt (angelehnt an das Verfahren von [Joh09]).

Um flüssige Bewegungen der Gliedmaßen zwischen vorbestimmten Punkten in der Simulation möglich zu machen, sollen Hermite Splines generiert werden, die die entsprechenden Punkte verbinden. Daraufhin sollen Inverse Kinematics angewendet werden, um die Gliedmaßen entlang der Splines zu bewegen.

## 4.2 Verwendete Bibliotheken

Um viele der grundlegenden Prozesse im Programm zu vereinfachen und schon gut gelöste Probleme wie z.B. das Erstellen eines OpenGL-Kontexts oder das Laden der Modelldaten nicht erneut lösen zu müssen, werden einige Bibliotheken verwendet. Diese werden im Folgenden im Detail erläutert.

### 4.2.1 OpenGL

OpenGL ist eins der meist verwendeten Grafik-APIs und bietet als solches diverse Funktionen, um mit der Grafikkarte zu interagieren. Es bietet außerdem die Shading-Language GLSL, die zur Programmierung der Shader verwendet wird. In dieser Anwendung kommt die Version 3.3 Core zum Einsatz.

### 4.2.2 OpenGL Mathematics (GLM)

Bei GLM handelt es sich um eine Mathematik-Bibliothek für C++, die der Namensgebung und Funktionalität von GLSL entspricht und deren Datenformate so aufgebaut sind, dass sie den von GLSL erwarteten Formaten entsprechen. Die von GLM bereitgestellten Matrix- und Vektorklassen und die Funktionen, die mit ihnen arbeiten, werden für Berechnungen auf der CPU sowie zum Hochladen von Daten in den GPU-Speicher verwendet.

### 4.2.3 Simple DirectMedia Layer (SDL)

SDL ist eine leichtgewichtige C-Bibliothek, die das Erstellen und Verwalten eines Fensters und dazugehörigen OpenGL-Kontexts unter Windows stark vereinfacht. Außerdem bietet sie mit simplen Funktionen Zugang zu Maus-, Tastatur- und Gamepad-Inputs. Des Weiteren wird die dazugehörige Bibliothek SDL_image zum Laden von Bildern im PNG-Format verwendet.

### 4.2.4 Open Asset Import Library (Assimp)

Das Skelett des zu animierenden Charakters und damit zusammenhängend auch das Mesh sowie die UV-Koordinaten und Bone-Weights liegen dem Programm im FBX-Format vor.Assimp ermöglicht es, dieses Dateiformat zu parsen und dabei einige Post-Processing Operationen wie das Vereinigen identischer Vertices durchzuführen.

### 4.2.5 Dear ImGui

Bei Dear ImGui handelt es sich um eine Implementierung des Immediate Mode GUI Paradigmas. Es ermöglicht die Darstellung von grafischen Interfaces im bestehenden OpenGL-Kontext und wird hier verwendet, um ein Debug-Informationen anzuzeigen, die erstellten Tools zu kontrollieren und einige Variablen der Simulation zur Laufzeit ändern zu können.

## 4.3 Aufbau des Programms

Das UML-Klassendiagramm in Abb. 4.1 zeigt den generellen Aufbau der Anwendung. Im Zentrum steht dabei die Klasse `Game`, die auf hoher Ebene den Ablauf koordiniert und die verschiedenen Komponenten der Simulation verwaltet. Ihre Methode `init()` initialisiert alle nötigen Systeme und erstellt das Fenster, sodass das Programm in einem geeigneten Ausgangszustand ist. Die Methode `run()` berechnet den nächsten Simulationsschritt und zeichnet das Ergebnis auf den Bildschirm.

Zum Zeichnen der Grafiken und Verwalten der Shader wird `Renderer` verwendet. Er lädt bei seiner Initialisierung die drei verschiedenen Shader-Programme, die in der Anwendung zum Einsatz kommen und besitzt jeweils ein Objekt der vier entsprechenden Shader-Klassen `DebugShader`, `TexturedShader`, `RiggedShader` und `BoneShader`. Die verschiedenen Shader und ihre Anwendungsgebiete werden in Abschnitt 4.3.1 ausführlicher beschrieben.

Die Klassen `MouseKeyboardInput` und `Gamepad` verwalten die verschiedenen Eingabemöglichkeiten und werden zu Beginn jedes Frames aktualisiert. Bei `Background` handelt es sich um eine sehr einfache Klasse, die nur dafür zuständig ist, eine Textur als Hintergrund zu rendern. Der Hintergrund stellt einen Referenzpunkt für die Bewegungen des Charakters da und erleichtert so die Beurteilung der Animationen.

Da der Charakter über Untergründe verschiedener Höhen laufen können soll, wurde die Klasse `Level` zur Repräsentation der Bodenstruktur erstellt. Der Untergrund besteht dabei aus einer Reihe von Collidern, die jeweils aus Quadraten bestehen und entlang der globalen X- und Y-Achsen ausgerichtet sind. Um den Aufbau des Levels anzupassen gibt es außerdem den `LevelEditor`. Er bietet ein simples User Interface zum Erstellen, Löschen und Verändern der Levelelemente. Weiterhin besteht die Möglichkeit, die Level in einem selbst erstellten Dateiformat zu speichern und wieder zu laden. Beim Start des Programms wird immer die Datei `assets/default.level` geladen, über den Editor können aber auch noch weitere Levels zum testen verschiedener Aspekte der Laufsimulation geladen werden.

`Player` repräsentiert den Spielercharakter und ist eine Unterklasse von `Entity`. Er besitzt ein `Mesh`, das wiederum aus einer Reihe von Vertices und Knochen besteht. Zusammen mit der Textur bietet es die Grundlage für die visuelle Darstellung des Charakters. Kontrolliert wird letzterer hauptsächlich mit den Inputdaten des `Gamepad`.

Der `Animator` steuert die Bewegungen des Spielercharakters und bildet somit den Kern der Anwendung. Er beinhaltet Repräsentationen der vier Gliedmaßen des Spielers in `limbs` und Splines, an denen die Bewegungsabläufe orientiert werden in `spline_prototypes`. Der genaue Ablauf des Animationsprozesses wird in Abschnitt 4.4 ausführlich erklärt.

### 4.3.1 Rendering und Shader

Alle Klassen, die grafische Objekte in der Simulation darstellen, haben eine Methode `void render(const Renderer&)` (eventuell noch mit weiteren Parametern),

mit der sie sich selbst auf den Bildschirm zeichnen. Die Objekte wählen dann einen der vier verschiedenen Shader aus, indem sie auf dem entsprechenden Member-Objekt des Renderers `use()` aufrufen und setzen gegebenenfalls noch Uniform-Variablen. Die zu rendernden Vertexdaten halten die Objekte in einer Instanziierung des Templates `VertexArray<>` mit dem zum Shader passenden Vertextyp als Template-Argument. Sie rufen auf diesem VertexArray `draw(GLenum mode)` auf, um die Vertices im entsprechenden Modus zeichnen zu lassen.

Die vier verfügbaren Shader sind:

1. `DebugShader`
2. `TexturedShader`
3. `RiggedShader`
4. `BoneShader`

Der DebugShader wird hauptsächlich zur Visualisierung von Debug-Informationen verwendet. Es kann eine Farbe für die zu rendernden Primitive in Form einer Uniform-Variable eingestellt werden; Texturen werden nicht verwendet. Das Vertexformat dieses Shaders besteht nur aus einem zweidimensionalen Vektor zur Angabe der Vertexposition. Außerdem gibt es die statische Variable `DEFAULT_VAO`, die ein simples Vertex Array zur Darstellung von quadratischen Objekten anbietet. Diese müssen so nicht ihre eigenen Vertices im Arbeitsspeicher der GPU verwalten, sondern können die beschriebenen Vertices verwenden und mithilfe der Model-Matrix beliebig transformieren.

Bei texturierten Objekten kommt der TexturedShader zum Einsatz. Seine Vertices enthalten deshalb neben der Position noch eine Texturkoordinate; er funktioniert ansonsten aber sehr ähnlich wie der DebugShader.

Soll ein animierter Charakter mit Skelett gerendert werden, wird der RiggedShader verwendet. Er ist der komplexeste der hier verwendeten Shader und transformiert die Vertices anhand der ihnen zugeordneten Knochen. Seine Input-Vertices bestehen deshalb aus vier Komponenten: Neben Position und Texturkoordinate noch die Indizes der zwei Knochen, die diesen Vertex beeinflussen, und die dazugehörigen Gewichtungen. Die Transformationsmatrizen der verschiedenen Knochen werden vor dem Rendering auf der CPU berechnet und als Uniform-Variable auf die GPU hochgeladen, wo der Shader dann auf sie zugreifen kann.

Der BoneShader zeichnet die Knochen des Modells. Seine Input-Vertices sollten jeweils Head- und Tail-Position der Knochen des Modells sein. Die Vertices sollten in der gleichen Reihenfolge wie die Knochen des Modells vorliegen, denn der Shader verwendet die gleichen Bone-Transformationsmatrizen wie der RiggedShader und berechnet den Index des zu verwendenden Knochens aus dem Index des Vertices (geteilt durch Zwei). So können die Vertices wie beim DebugShader lediglich aus einem zweidimensionalen Vektor zur Angabe der Position bestehen.

Nur die Klassen `Mesh`, `Background` und `Spline` verwenden eigene Vertex-Arrays auf der GPU. Nur `Spline` verändert diese Daten nach der Initialisierung noch, die anderen beiden transformieren die Vertices nur noch durch Uniforms.

## 4.4 Animationsprozess

Zur Animation des Charakters bestimmt der `Player` in seiner Update-Methode zunächst die aktuelle Bewegungsgeschwindigkeit anhand der Eingaben des Spielers. Sollte der Charakter gerade in die entgegengesetzte Richtung der Eingabe blicken, wird außerdem das Modell an der Y-Achse gespiegelt. Daraufhin ruft der `Player` die Update-Methode des `Animator`s auf und gibt dabei die Bewegungsgeschwindigkeit – zusammen mit einer Liste der Collider im Level und dem Faktor `delta_time`, der die vergangene Zeit seit dem letzten Simulationsschritt angibt – als Argumente weiter.

Ein Aktivitätsdiagramm der nun folgenden Abläufe zur Animation des Charakters ist in Abb. 4.2 zu sehen. Der `Animator` bestimmt zunächst anhand der Bewegungsgeschwindigkeit und des aktuellen Animationszustands, ob eine neue Animation bestimmt werden muss. Ist dies der Fall, wird der neue Animationszustand in `leg_state` gespeichert und die Methode `set_new_splines` aufgerufen, die daraufhin die nächste Animation berechnet.

Als nächstes wird der Interpolationsfaktor der Animation abhängig von `delta_time` erhöht und die Rotation des Rückens weiter zu ihrer Zielrotation hin interpoliert. Zuletzt berechnet das Programm anhand des Interpolationsfaktors und der jeweiligen Splines die aktuellen Zielpunkte für die Gliedmaßen und sucht mittels Inverse Kinematics nach einer passenden Stellung für die beiden Gelenke des Arms oder Beins. Damit endet die Update-Methode des `Animator`s und der `Player` verschiebt seine Position auf die neue Beckenposition.

### 4.4.1 Generierung neuer Animationen

In der genannten Methode wird dann als erstes anhand der Laufgeschwindigkeit eine Zielrotation für den Rücken des Charakters bestimmt. Diese dient dazu, dass letzterer sich beim Rennen in die Bewegung hinein lehnt. Um flüssige Bewegungsabläufe zu gewährleisten wird dieser Wert nicht direkt als Rotation des entsprechenden Knochens gesetzt, sondern als Zielwert zu dem der Knochen dann hin interpoliert.

Daraufhin wird zwischen zwei verschiedenen Fällen unterschieden, die unterschiedliche Animationen erfordern: Entweder der Charakter steht still, oder er bewegt sich. Da die Animation für den stehenden Charakter (Idle-Animation) aus einer leichten auf und ab Bewegung besteht, muss hier zuerst festgestellt werden, in welche Richtung die Animation als nächstes ausgeführt werden soll. Daraufhin werden für die Arme die entsprechenden Spline-Prototypen ausgewählt und an die richtige Position relativ zum Arm transformiert. Außerdem wird der Startpunkt der Spline an die aktuelle Position der Hand gesetzt. Dieser Schritt findet in fast allen Fällen beim setzen eines neuen Splines für Gliedmaßen statt und gewährleistet, dass es keine Sprünge in der Animation gibt und stattdessen immer weiche Bewegungen von einem Punkt zum nächsten erfolgen.

Für die Beine wird bei der Idle-Animation nach dem Boden direkt unter dem Hüftknochen gesucht und ein Spline direkt vom aktuellen Punkt dort hin verwendet. Das Becken verwendet als Basis seiner Bewegung wie die Arme einen

Spline-Prototypen. Da ganze Körper des Charakters der Bewegung des Beckens folgt muss beim Erstellen eines Splines sichergestellt werden, dass die Beine immer den Boden erreichen können. Die Höhe des Beckens über dem Boden wird durch eine Membervariable in `Animator` bestimmt, deren Wert deshalb immer kleiner sein sollte als die Länge eines Beins. Als Boden unter dem Becken wird außerdem immer der niedrige der beiden Untergründe unter den Beinen gewählt. Zuletzt wird so ermittelte Spline wird dann je nach aktueller Richtung der Idle-Animation noch umgedreht.

Steht der Spieler hingegen nicht still, wird zunächst anhand der gewünschten Bewegungsgeschwindigkeit die Länge des nächsten Schritts ermittelt. Sollte der Charakter gerade erst los laufen, wird diese Schrittweite halbiert, um die Position für den vorderen Fuß nicht so weit nach vorne zu setzen, dass das Bein diese gar nicht mehr erreichen kann. Im Folgenden wird zur Bestimmung der Splines jeweils zwischen zwei Prototypen für Gehen und Rennen interpoliert. Der dabei verwendete Interpolationsfaktor ist wie die Schrittweite von der Laufgeschwindigkeit abhängig.

Die Splines für die Arme werden durch die gerade beschriebene Interpolation berechnet und der Zielpunkt um eine Schrittweite in Bewegungsrichtung verschoben. Das ist nötig um die Position des Punkts relativ zum Körper konstant zu halten, da die Splines in Weltkoordinaten angegeben werden und der Körper des Charakters sich bis zu Ende des Schritts um eben diese Schrittweite bewegt haben wird.

Die beiden Beine führen beim laufen jeweils verschiedene Bewegungen aus und müssen deshalb getrennt betrachtet werden. Das führende Bein berechnet zunächst die Interpolation der beiden Spline-Prototypen, passt dann aber noch sowohl den Ausgangs- als auch den Endpunkt an. Als Ausgangspunkt wird die aktuelle Fußposition gesetzt. Der Fuß soll sich am Ende des Schritts eine halbe Schrittweite vor dem Körper befinden, deshalb wird als X-Koordinate des Endpunkts die des Ausgangspunkts verwendet und um 1,5 Schrittweiten nach vorne verschoben. Das Level wird an dieser Stelle nach geeignetem Untergrund durchsucht und die Höhe von diesem als Y-Koordinate verwendet. Bei diesem Verfahren werden zwar die beiden interpolierten Punkte quasi vollständig missachtet, die Interpolation hat aber trotzdem einen Zweck, da die so bestimmten Tangenten den Stil der Bewegung maßgeblich bestimmen.

Da andere Bein steht während des gesamten Schritts am gleichen Ort auf dem Boden; sein Spline ist deshalb trivial bestimmbar: Beide Punkte entsprechen der aktuellen Fußposition, beide Tangenten haben die Länge Null.[1] Zuletzt wird dann noch die Bewegung des Beckens bestimmt. Dazu werden wieder die interpolierten Spline-Prototypen genutzt, die Startposition auf die aktuelle Beckenposition gesetzt und die X-Koordinate der Zielposition als eine Schrittweite davon entfernt festgelegt. Zur Bestimmung der Y-Koordinate berechnet das Programm die durchschnittliche Höhe der beiden Beine und fügt dem Ergebnis die im `Animator`

[1] Eigentlich könnte diese Position auch einfach direkt festgelegt und gar kein Spline mehr verwendet werden. Es müsste dann aber ein Sonderfall für das stehende Bein im Code eingeführt werden, der das System zwar möglicherweise effizienter, dafür aber auch schwieriger verständlich machen würde.

eingestellte Beckenhöhe hinzu. Als letztes wird dann noch der Faktor zur Interpolation auf den Splines auf Null zurückgesetzt, sodass die Animationen bei ihrem Startpunkt beginnen.

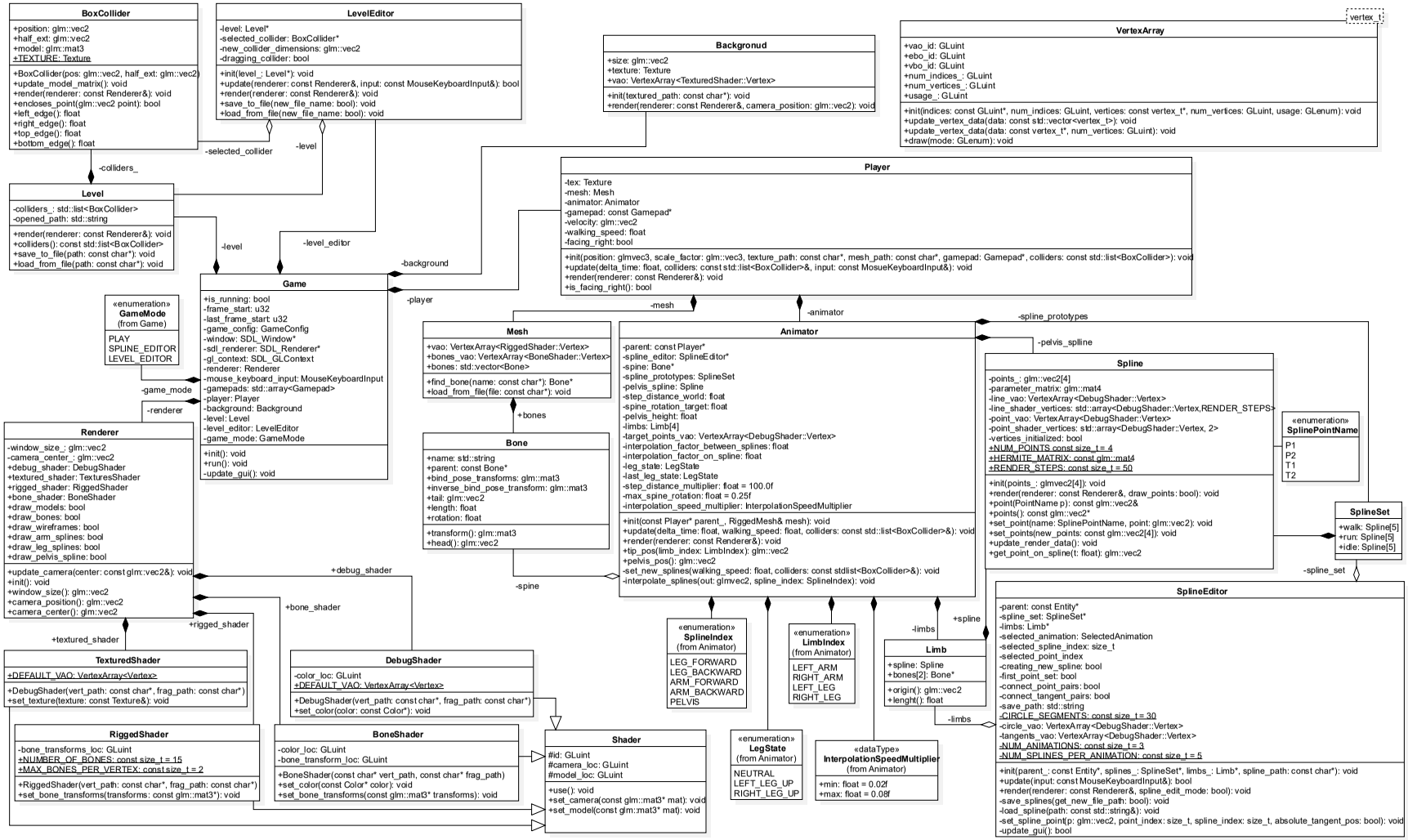



Abb. 4.1: Objektdiagramm des gesamten Programms. Einige Klassen und Assoziationen wurden zur Verbesserung der Übersichtlichkeit ausgelassen.

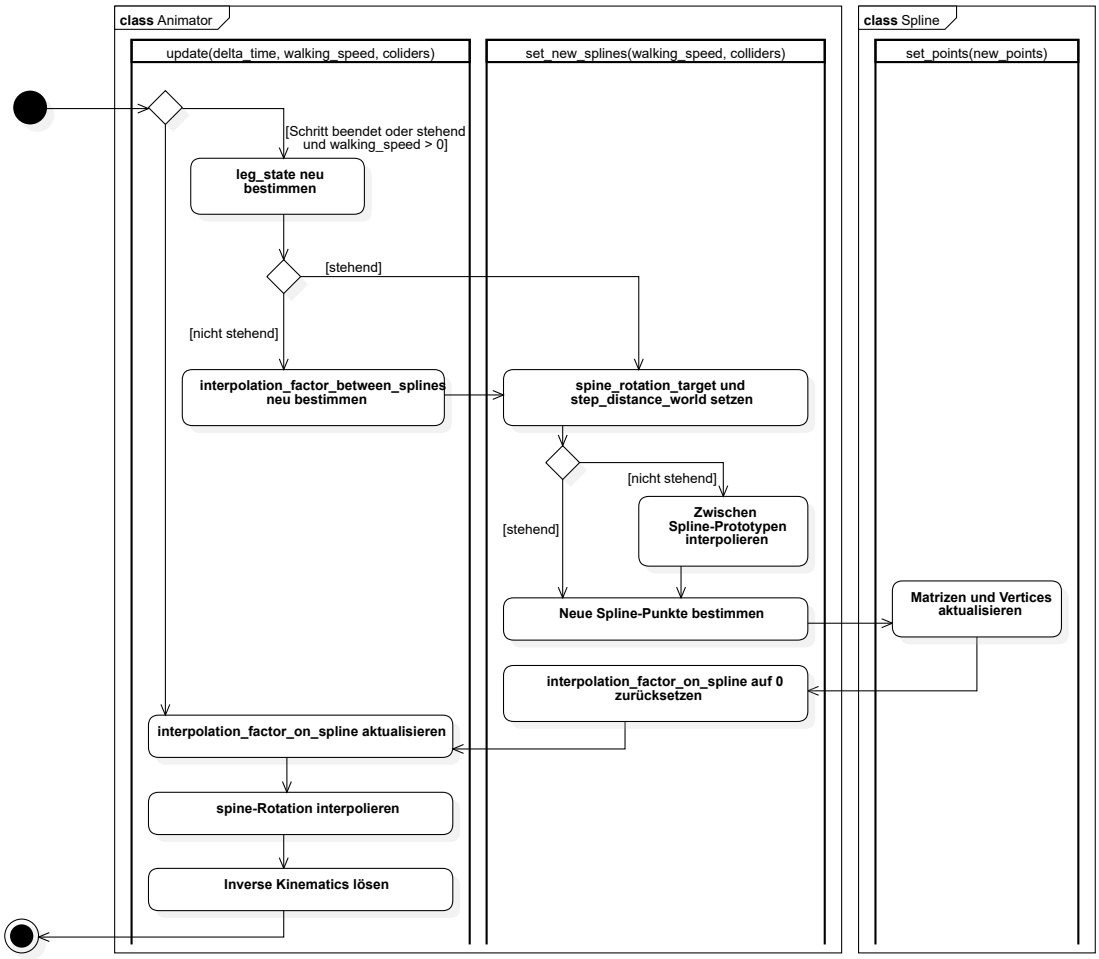



Abb. 4.2: Aktivitätsdiagramm des Animationsprozesses

### 4.4.2 Anpassung der Bewegungen

Um den Stil der generierten Animationen an die Persönlichkeit des animierten Charakters anzupassen, erlaubt das Interface direkten Zugriff auf die Membervariablen `step_distance_multiplier`, `interpolation_speed_multiplier` und `max_spine_rotation` des `Animator`s. Außerdem wurde ein Editor für die Spline-Prototypen implementiert, der direkt mit den Daten `Animator`s arbeitet und diese auch in einem Binärformat speichern und laden kann.

# 5

# Ergebnisse

Allgemein kann festgestellt werden, dass das Versuch, die hauptsächlich in der der 3D-Animation verwendete Technik der skelettbasierten Animation zur dynamischen Animation von 2D-Charakteren zu Nutzen, erfolgreich war. Das entwickelte System verwendet ein Skelett und erstellt anhand einiger zuvor festgelegter Splines für jeden Schritt eine neue Bewegungsabfolge. Wird ein Mesh erstellt und den einzelnen Vertices Knochen zugeteilt, werden die Vertices und die darauf gezeichnete Textur anhand des Skeletts verformt.

Der Charakter verfügt dabei über die Möglichkeit, anhand der Auslenkung des Control-Sicks präzise seine Schrittweite und Bewegungsgeschwindigkeit anzupassen. Beispiele für eine langsame und eine schnelle Bewegungsanimation sind in Abb. 5.1a bzw. Abb. 5.1b zu sehen. Die Interpolation zwischen zwei verschiedenen Splines – einem für das langsame Gehen, einem für das Rennen – sorgt dabei dafür, dass nicht nur die Länge des Schritts wächst, sondern auch die Form der Bewegung sich verändert.

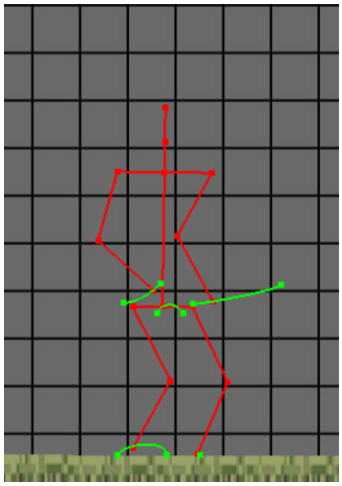
Des Weiteren wurde das Ziel erreicht, Bewegungen über Flächen verschiedene Höhen möglich zu machen. Der Charakter sucht sich dabei an der gewünschten nächsten Fußposition den passenden Boden und berechnet einen Spline mit dem entsprechenden Punkt als Ziel. Eine Animation für das Hinaufsteigen einer Stufe wird in Abb. 5.1c gezeigt, das Hinabsteigen einer solchen ist in Abb. 5.1d zu sehen. Eine entscheidende Limitierung ist dabei aber, dass gerade bei kleinen Schritten mangels Collision Detection für die Splines nicht sichergestellt werden kann, dass die dynamisch erstellten Bewegungen nicht mit dem Level kollidieren, wie in Abb. 5.1e gezeigt.

Um dem Ziel der Responsivität gerecht zu werden, ist es dem Charakter möglich, auch mitten in einem Schritt abrupt stehen zu bleiben. Da ein solches plötzliches Stoppen in der Realität jedoch nicht ohne Weiteres möglich ist, könnte dies zu einer unglaubwürdigen Animation führen könnte. Um mit diesem Problem umzugehen wird die Bewegung des Charakters zwar sofort gestoppt, es wird aber eine Übergangsanimation eingefügt, um die Abruptheit etwas zu verschleiern. Zum stehen kommen kann der Charakter dabei auch auf unebenem Untergrund und

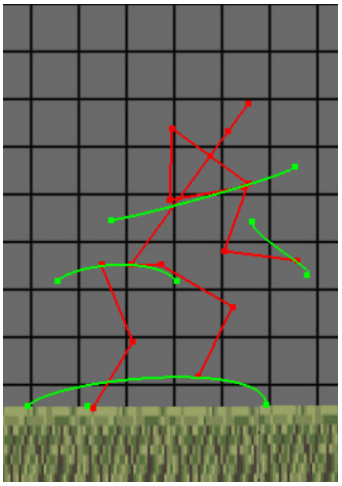
bewegt sich dabei stets so, dass seine Beine sauber auf dem Boden stehen (siehe Abb. 5.1f).[1]

Außerdem werden diverse Möglichkeiten geboten, die Generierung der Animationen anzupassen. Durch die Veränderung von Schrittweite, dem Grad des Einlehnens in die Bewegungsrichtung und der Geschwindigkeit der Interpolation auf den errechneten Splines sind bereits einige Stellschrauben gegeben. Weiterhin wird aber auch noch der Spline-Editor angeboten, der die präzise Anpassung der einzelnen Splines ermöglicht, die wiederum die Grundlage der dynamisch gebildeten Animationen darstellen. Die Kombination dieser Anpassungsmöglichkeiten erlaubt dem Nutzer, eine Vielzahl von verschiedenen Bewegungsmustern für die zu animierenden Charaktere zu erstellen.

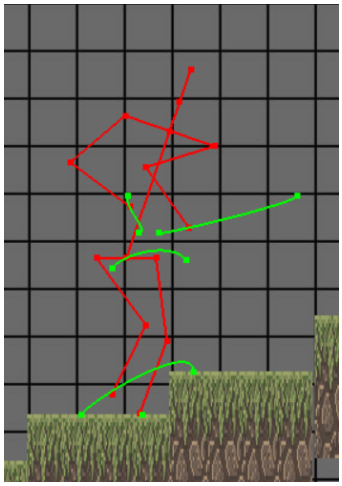
[1] Ein angemessenes Level wird dabei vorausgesetzt. Sind die Unterschiede zwischen benachbarten Untergründen zu groß, um realistisch auf ihnen stehen zu können, entstehen fehlerhafte Animationen.



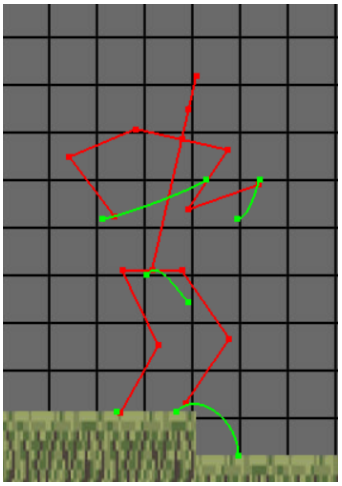
(a) Gehen auf ebenem Untergrund.



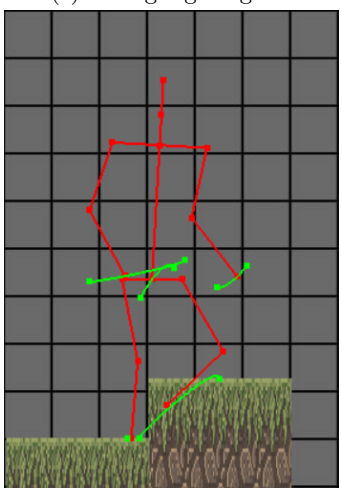
(b) Rennen auf ebenem Untergrund.



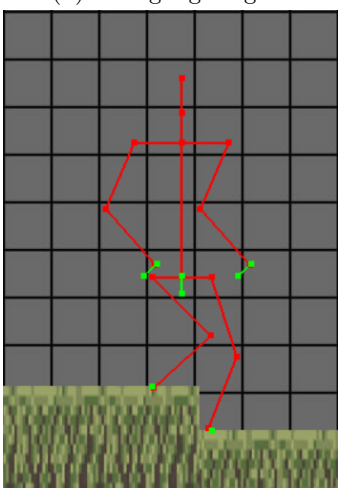
(c) Bewegung bergauf.



(d) Bewegung bergab.



(e) Ein Bein bewegt sich beim aufwärts gehen durch den Boden.



(f) Stehen auf unebenem Untergrund.

Abb. 5.1: Beispiele generierter Animationen. Das Skelett ist in rot dargestellt. Die grünen Linien stellen die Splines dar, an denen sich die entsprechenden Knochen entlang bewegen.

# 6

# Zusammenfassung und Ausblick

In dieser Arbeit wurde versucht, ein System zu entwickeln, das die automatische Laufanimation von zweidimensionalen Charakteren zur Laufzeit des Programms anhand eines Skeletts und eines einzelnen Sprites ermöglicht. Dieses Ziel wurde erreicht; der Charakter kann sich dabei auch über verschieden hohe Untergründe und in beliebiger Geschwindigkeit bewegen. Außerdem reagiert der Charakter dabei meist schnell auf Eingaben und der Rechenaufwand der Simulation ist überschaubar, was die Verwendung in einem Videospiel begünstigt. Ein Editor für die zur Animation verwendeten Splines ermöglicht die Anpassung der Bewegungsabläufe an die Bedürfnisse des Nutzers.

Das System verfügt jedoch auch noch über viele Limitierungen, an denen zukünftige Arbeiten ansetzen könnten. Beispielsweise ist aktuell immer ein Fuß fest an den Boden gebunden, obwohl Menschen beim Rennen in der Regel zeitweise keinen Kontakt mehr zum Bode haben. Ein offensichtlichen Fehler ist außerdem die fehlende Collision Detection beim Bewegen über unebenen Untergrund. So kann es häufig vorkommen, dass sich beispielsweise der Fuß des Charakters durch ein Hindernis bewegt. Durch die Verwendung passender Splines lässt sich das Problem etwas minimieren, was aber die Anpassungsmöglichkeiten der Animation stark einschränkt.

Im Zusammenhang mit dieser Problematik der Kollisionsvermeidung wäre es möglicherweise hilfreich, eine komplexere Art von Splines zu verwenden. Die aktuellen Hermite Splines bestehen immer nur aus zwei Punkten und den zwei dazugehörigen Tangenten, ein dritter Punkt könnte jedoch beispielsweise an den Mittelpunkt des Schritts gesetzt werden, um die Bewegung besser zu definieren und um Hindernisse herum zu navigieren. Ein komplexeres System zur Anpassung der Tangenten in diesem dritten Punkt wäre dann aber vermutlich auch vonnöten, um bei Schritten auf höhere oder niedrigere Ebenen einen gutaussehenden Bewegungsablauf zu wahren.

Außerdem funktioniert der Algorithmus zum Finden der neuen Fußpositionen nur mit Collidern, die entlang der X- und Y-Achsen der Spielwelt ausgerichtet sind. Schräge Flächen sind also nicht möglich. Hier könnte das System leicht erweitert werden, um mehr Variation in den Levels zu erlauben.

# Literaturverzeichnis


ASTK99. Aoki, Masakastsu, Mikio Shinya, Ken Tsutsuguchi und Naoya Kotani: *Dynamic texture: Physically-based 2D animation.* In: *ACM SIGGRAPH 1999 Conference Sketches and Applications*, Band 239, Seiten 311625–312130, 1999.

B+94. Bates, Joseph et al.: *The role of emotion in believable agents.* Communications of the ACM, 37(7):122–125, 1994.

BC96. Bruderlin, Armin und Tom Calvert: *Knowledge-driven, interactive animation of human running.* In: *Graphics Interface*, Band 96, Seiten 213–221, 1996.

Bru93. Bruderlin, Armin: *Interactive animation of personalized human locomotion.* In: *Graphics Interface*, Band 93, Seiten 17–23, 1993.

BTC94. Bruderlin, Armin, Chor Guan Teo und Tom Calvert: *Procedural movement for articulated figure animation.* Computers & Graphics, 18(4):453–461, 1994.

BW76. Burtnyk, Nestor und Marceli Wein: *Interactive skeleton techniques for enhancing motion dynamics in key frame animation.* Communications of the ACM, 19(10):564–569, 1976.

CH99. Chung, Shih-kai und James K Hahn: *Animation of human walking in virtual environments.* In: *Proceedings Computer Animation 1999*, Seiten 4–15. IEEE, 1999.

Joh09. Johansen, Rune Skovbo: *Automated semi-procedural animation for character locomotion.* Aarhus Universitet, Institut for Informations Medievidenskab, 2009.

PUS19. Pangesti, Annisa Rahayu, Ema Utami und Andi Sunyoto: *Analysis of Inverse Kinematics Method for Human Movement In 2D Animation.* In: *2019 1st International Conference on Cybernetics and Intelligent System (ICORIS)*, Band 1, Seiten 189–194. IEEE, 2019.

Sha11. Shapiro, Ari: *Building a character animation system.* In: *International conference on motion in games*, Seiten 98–109. Springer, 2011.

vO90. Overveld, Cornelius WAM van: *A technique for motion specification in computer animation.* The Visual Computer, 6(2):106–116, 1990.

# A

## Erklärung der Kandidatin / des Kandidaten

☐ Die Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

☐ Die Arbeit wurde als Gruppenarbeit angefertigt. Meine eigene Leistung ist ...

Diesen Teil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Namen der Mitverfasser: ...

| | |
|---|---|
| Datum | Unterschrift der Kandidatin / des Kandidaten |