Techniken der 2D-Skelettanimation

Studiengang Informatik

Bachelorarbeit

vorgelegt von

Benedikt Jensen
geb. in Langenfeld(Rheinland))

durchgeführt an:
Technische Hochschule Mittelhessen (THM), Gießen

Referent der Arbeit:	Prof. Dr. Soundso Mustermann
Korreferent der Arbeit:	Prof. Dr. Herr Mann
Betreuer an der THM:	Prof. Dr. Andreas Gogol-Döring

Gießen, 2022

Danksagung

Normalerweise haben eine ganze Reihe von Personen mehr oder wenig Anteil am Gelingen der Bachelorarbeit, denen man hier dankt: In der Regel zunächst den Referenten und Betreuern der Arbeit. Aber natürlich auch Personen, Firmen und Institutionen, die die Arbeit tatkräftig unterstützt haben. Sei es durch die Bereitstellung von spezieller Hard- oder Software oder nur durch ein gewissenhaftes Korrekturlesen.

5

Selbstständigkeitserklärung

Ich erkläre, dass ich die eingereichte Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Gießen, August 2022

Benedikt Jensen

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Animation ist die Technik, durch Darstellen einer Sequenz von Einzelbildern, die Illusion von flüssiger Bewegung zu schaffen. Ursprünglich wurden Animationen Bild für Bild erstellt. Mit der Entwicklung der Technik haben sich auch die Methoden zur Animationserstellung verändert. Heutzutage werden Animationen größtenteils mit Hilfe von Computern erstellt. Eine bewährte Technik der Computeranimation ist die Skelettanimation.

Im folgenden wird zwischen 3D- und 2D-Animation unterschieden. Als 3D-Animation versteht sich eine Animation, welche mit einem 3D-Modell als Basis erstellt wurde. Bei 2D-Animationen ist dies nicht der Fall. Skelettanimation ist sowohl für 3D-Animation als auch 2D-Animation möglich. Sie wird in den meisten Anwendungsbereichen vorwiegend für 3D-Animationen verwendet z.B. für das Erstellen von 3D-Animationsfilmen. In 2D hat die Skelettanimation einige Nachteile. Da auch 2D-Animationen meist Charaktere in einem 3D-Raum repräsentieren ist zur exakten Abbildung dieser Charaktere auch ein 3D-Modell notwendig. Das ist der Grund dafür, dass 2D Skelettanimationen oft etwas unnatürlich wirken.

Dennoch ist die 2D-Skelettanimation in der Entwicklung von 2D-Videospielen weit verbreitet, da sie hier im Vergleich zur Bild-zu-Bild-Animation einige Vorteile mit sich bringt. Da die Skelettanimation weniger arbeitsintensiv als die Bild-zu-Bild-Animation und ermöglicht effiziente Abänderungen der Animation im Nachhinein und sogar zur Laufzeit. Des Weiteren ermöglicht die Skelettanimation eine größere Interaktion des animierten Charakters mit der Spielwelt. Die vorliegende Arbeit ist eine Untersuchung verbreiteter Techniken zur 2D-Skelettanimation und deren Anwendung in Videospielen.

Kapitel 2

Funktionsweise

2.1 Skin und Skelett

Damit sich ein Charakter animieren lässt, müssen zunächst folgende Bestandteile erstellt werden. Der Skin (engl.: Haut) ist die Erscheinung der zu animierenden Figur. Sie besteht aus einer oder mehreren Bilddateien, welche zusammen die gesamte Figur abbilden. Das Skelett ist eine Ansammlung von transformierbaren Knochen. Grundlegende Transformationen sind Rotation, Verschiebung und Skalierung. Es ist auch möglich weitere Merkmale wie z.B. Einfärbung, Weichzeichnung, Skew (engl.: Schiefstellung) miteinzubeziehen.

2.1.1 Hierarchische Knochenstruktur

Um eine realitätsnahe Animation zu ermöglichen, werden die Knochen des Skeletts hierarchisch angeordnet. Knochen stehen zueinander in einer Eltern-Kind-Beziehung. Der Knochen, welcher einem anderen in dieser Hierarchie übergeordnet ist, wird als Eltern-Knochen bezeichnet. Entsprechend wird ein untergeordneter Knochen Kind-Knochen genannt. Kind-Knochen ererben die Transformationswerte des Eltern-Knochen, zu welchen ihre eigenen Transformationswerte hinzugefügt werden. Mit dieser Aufstellung wird die Illusion geschaffen, dass Kind-Knochen an den Eltern-Knochen befestigt sind, sowie eine Hand an einem Arm. Die Berechnung der Position und Orientierung des letzten Elements in einer kinematischen Kette auf diese Weise wird als „Forward Kinematics“ bezeichnet.

2.1.2 Manipulation von Skins

Die Beziehung zwischen Skelett und Skin muss definiert werden. Die simpelste Art das zu tun, ist jeder Bilddatei des Skins einen Knochen des Skeletts zuzuweisen. Die Bilddatei übernimmt Position, Rotation und Skalierung des zugehörigen Knochens.

2.2 Keyframes

Bei der Skelettanimation wird nicht jedes Einzelbild von Hand erstellt. Stattdessen werden Keyframes (engl.: Schlüsselbilder) erstellt, aufgrund welcher die Gesamtheit aller Einzelbilder

programmatisch generiert wird.

2.3 Interpolierung

Um aus Keyframes eine flüssige Animation zu machen wird die Technik der Interpolierung genutzt. Die Tranformation der Knochen über den Zeitraum zwischen zwei Keyframes hinweg wird als Funktion verstanden. Im Folgenden wird dies am Beispiel der Interpolierung der Rotation eines Knochens veranschaulicht. Folgende Angaben sind bekannt:

t_0	der Zeitpunkt des 1. Keyframes
t_1	der Zeitpunkt des 2. Keyframes
t	Zeitpunkt der Betrachtung
x	$(t - t_0) \div (t_1 - t_0)$ (Anteil der verstrichenen Zeit)
a	Rotation des Knochens zu t_0
b	Rotation des Knochens zu t_1

Gesucht ist $f(t) =$ Rotation des Knochens zum Zeitpunkt t.
Die Interpolierungsfunktion kann je nach gewünschtem Ergebnis definiert werden. Das simpelste Beispiel ist die lineare Interpolierung:

$$lerp(a, b, x) = a \cdot x + b \cdot (1 - x) \tag{2.1}$$

Zur Interpolierung von Skalierung und Translation eines Knochens genügt es die Rotation durch die entsprechenden Werte zu ersetzen.

Kapitel 3

Weiterführenede Techniken

3.1 Variieren der Interpolierungsfunktion

Die oben aufgezeigte Interpolierungsfunktion ist in den meisten Anwendungsfällen nicht geeignet. Betrachten wir eine Figur, welche ihren Arm hebt. Zwei Keyframes wurden definiert: Die Figur mit gesenktem Arm und mit gehobenem Arm. Spielen wir dieser Animation als Schleife ab, hebt und senkt die Figur ihren Arm wiederholt. Jedoch kommt es dabei zu abrupten Richtungswechseln, welche unnatürlich wirken. Das liegt daran, dass sich physikalische Objekte nach dem Gesetz der Trägheit bewegen. Um eine Veränderung der Bewegungsrichtung zu erreichen, muss das Objekt beschleunigt werden.

Um diese Eigenschaft physikalischer Objekte zu simulieren, bedarf es einer Interpolierungsfunktion, wessen Ableitung keine Sprünge macht, weder zwischen zwei Keyframes noch über angrenzende Keyframes hinweg.

Die wie folgt beschriebene Funktion $ease\_in\_out$ hat diese Eigenschaften. Sie lässt die Animation langsam beginnen, dann schneller werden und zum Ende wieder langsamer werden. (vgl. [Feb18]). Die Funktion $lerp$ wird unverändert aus Formel 2.1 übernommen:

$$ease\_in(x) = x^2 \tag{3.1}$$

$$ease\_out(x) = (1 - x)^2 \tag{3.2}$$

$$ease\_in\_out(x) = lerp(ease\_in(x), ease\_out(x), x) \tag{3.3}$$

Das Ergebnis von $ease\_in\_out(x)$ setzen wir statt x in $lerp$ ein

$$lerp(a, b, ease\_in\_out(x)) \tag{3.4}$$

Und bekommen somit für jeden Zeitpunkt t den richtigen Transformations-Wert.

3.2 Animation Layering

Besonders in interaktiven Anwendungen ist kann es hilfreich sein, mehrere Animationen zu kombinieren. Erstellt man eine Animation für einen bestimmten Bereich, dann kann man diesen Bereich unabhängig vom Rest des Körpers animieren. Die Transformations-Werte aller betroffenen Gelenke werden je Keyframe abgespeichert. Diese werden interpoliert, um für jeden Frame wie weiter oben geschildert mit Forward Kinematics die globalen Transformations-Werte zu berechnen. Beim Animation Layering ersezt eine Ebene für den betroffen Bereich alle berechneten Werte darunterliegender Ebenen.

Betrachten wir folgendes Beispiel:
Zu animierende Figur hat 2 Animationen: Gehen und Winken. Die Gehen-Animation betrifft alle Gelenke der Figur. Die Winken-Animation hingegen betrifft nur die Gelenke des rechten Arms. Um beide Animationen zu kombinieren, werden die Rotationen der Armgelenke der Gehen-Animation durch die Rotationen der Winken-Animationen ersetzt.

3.3 Additive Animation Blending

Neben dem Animation Layering gibt es auch die Technik des "Additive Animation Blending". Zwei oder mehr Animationen werden zu einer neuen Animation zusammengefügt. Die neue Animation besteht zu vom Animator festgelegten Gewichtungen aus bereits vorhandenen Animationen. So ließe sich z.B. eine Gehen-Animation und eine Rennen-Animation kombinieren.

3.4 Meshes

Ein Mesh(engl.: Netz) ist eine Ansammlung von Punkten und Dreiecken, welche sich aus den Verbindungslinien jeweils 3 dieser Punkte bilden. Ein solches Mesh kann als ein Array von Vertices(Punkte im Koordinatensystem) und ein Array von Indices definiert werden. Drei dieser Indices, welche auf je einen Punkt des ersten Arrays verweisen, bilden jeweils ein Dreieck. Mithilfe eines solchen Meshs können wir nun eine Textur aufspannen. Dazu definieren wir zusätlich ein Array von UVs(Bildkoordinaten). Je Punkt wird ein UV festgelegt. Unabhängig von den Koordinaten eines Punkts können wir ihm jetzt eine Position in der Textur-Bilddatei zuweisen. Mit den gegebenen Informationen kann nun die Farbe jedes Pixels innerhalb eines Dreiecks bestimmt werden. Dazu wenden wir den ¡¡Triangle rendering algorithm¿¿ an.

Die Verwendung von Meshes zum Rendern von Grafiken bringt einige Vorteile mit sich, darunter:
1) Bilder können nicht nur in ihrer ursprünlichen Erscheinung sondern auch in verzerrter Form angezeigt werden.
2) Transparente Bereiche von Bilddateien mit Alpha-Kanal können beim anzeichnen ignoriert werden, was die Effizienz erhöht.

3.4.1 Mesh Generierung

Je nach Verwendungszweck unterscheidet sich die Natur des am besten geeigneten Meshs. Es gilt die Größe der vom Mesh bedeckten Fläche zu minimieren, um die Effizienz zu maximieren. Das bedeutet, das Mesh sollte alle sichtbaren Bereiche bedecken, damit diese vollständig angezeigt werden, aber vollständig transparente Bereiche sollten möglichst ausgelassen werden.

Die Dichte von Dreiecken in einem Bereich des Bildes und die Form und Lage der Dreiecke beeinflusst die Art und Weise, auf welche sich die angezeigte Grafik bei Verzerrung des Meshs verändert. Übermäßig große Dreiecke, besonders in Bereichen, welche stark verzerrt werden, führen meist zu einem unnatürlich erscheinendem Ergebnis.

Marching Squares Algorithmus

Um ein enganliegendes Mesh zu generieren, berechnen wir erst einmal eine Kontur für unsere Bilddatei, in unserem Fall ein Bild im PNG-Format. Um aus einer Bilddatei eine Kontur zu generieren, kann der Marching Squares Algorithmus (engl.: wandernde Quadrate) angewandt werden. Die Kontur sollte etwas weiter als die tatsächliche Kontur des abgebildeten Objekts sein. Das dient dazu, dass alle sichtbaren Teile beim Rendern später vollständig gezeichnet werden. Zudem ermöglicht es das aussortieren überflüssiger Eckpunkte, die durch den Marching Squares Algorithmus entstehen. (vgl. [Map03])

Unser Ziel ist es, eine Kontur um alle sichtbaren Teile der Bilddatei zu ziehen. Als sichtbar sehen wir alles an, was einen festgelegten Alpha-Schwellwert überschreitet. Dazu generieren wir aus dem Alpha-Kanal des Bildes ein Distanzfeld. Das Distanzfeld ist eine Grauwertmatrix der selben Auflösung des Bildes. Für jeden Pixel der Bilddatei wird hier die Distanz zum nächstgelegenen sichtbaren Pixel berechnet und festgehalten. Anhand eines weiteren Schwellwerts generieren wir ein Raster, dessen Zellen jeweils entweder $true$ für sichtbar oder $false$ für unsichtbar sind. Auf dieses Raster wird nun ein Konturraster gelegt, welches in x- und y-Richtung jeweils eine Zelle kleiner ist. Die Eckpunkte des Konturrasters liegen jeweils im Zentrum einer der Binärzellen. Jeder Zelle des Konturrasters wird ein Wert zugewiesen. Dieser ergibt sich daraus, ob die Werte der vier Ecken dieser Zelle $true$ oder $false$ sind. Den Ecken werden die Werte 2^0, 2^1, 2^3 und 2^4 zugewiesen. (siehe Abb.: 3.1) [Wik22] Für jede der 15 möglichen Kombinationen wird jeweils ein entsprechendes Konturstück festgelegt. Diese 15 Fälle können in einer Lookup-Tabelle gespeichert werden. Fügt man all diese Stücke zusammen ergibt sich die gesuchte Kontur. Abb. 3.3 und 3.4 zeigen die Vorgangsweise an einem Beispiel. Wir generieren nun aus den Zellen einen ungerichteten Graphen.

3.4.2 Mesh Deformierung

Die Deformierung des Meshs erfolgt durch die Verschiebung der dazugehörigen Punkte. Um eine solche Deformierung durch das Transformieren der Skelett-Knochen zu erreichen, legen

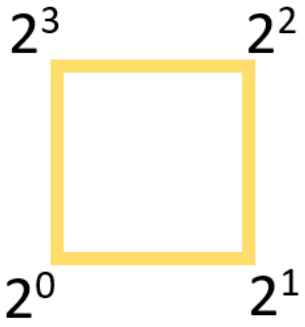

Abbildung 3.1: Konturzelle

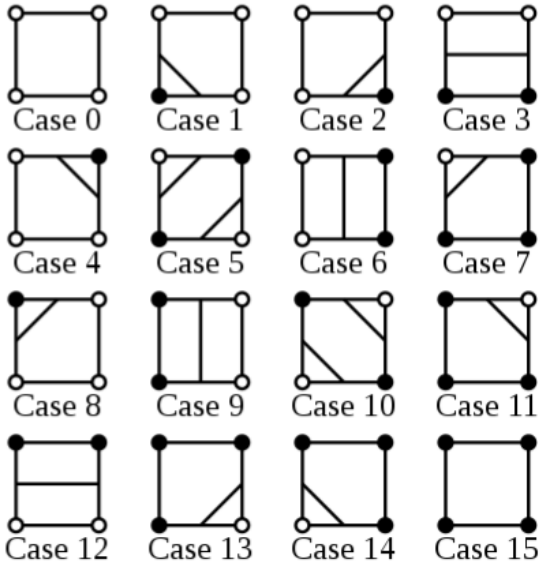

Abbildung 3.2: Mögliche Fälle (vgl. [**?**])

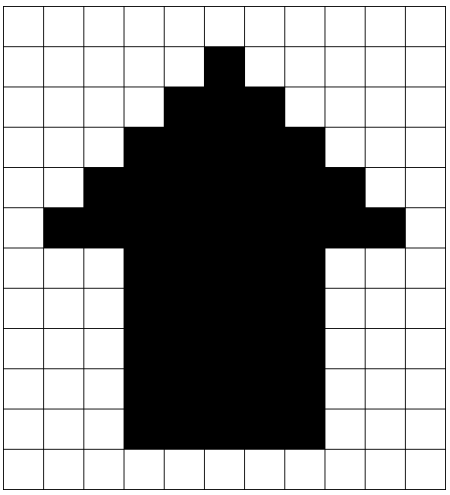

Abbildung 3.3: Ausgangsposition

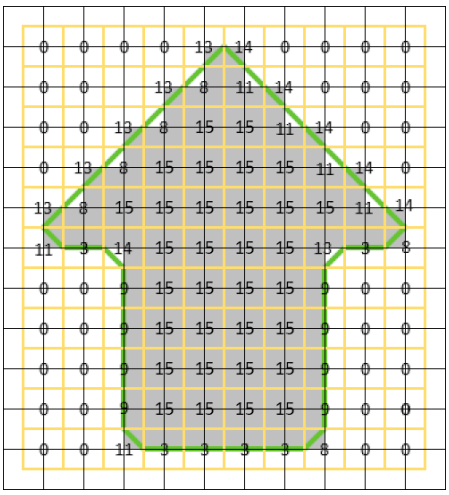

Abbildung 3.4: Zielposition

wir den Einflussgrad jedes Knochens auf jeden Mesh-Punkt fest. Zusätzlich legen wir die Lage jedes Punktes relativ zur Lage, Rotation und Skalierung jedes auf ihn Einfluss nehmenden Knochens fest. Für jeden Mesh-Punkt ergeben sich nun bei beliebiger Skelettpose eine Anzahl an globalen Koordinaten. Die endgültige Punktkoordinate ergibt sich zu festgelegten Anteielen aus diesen global Koordinaten. Das angepasste Mesh geben wir an die Graphics Library weiter, welche sich um das Rendern kümmert.

3.5 Prozedurale Animation

Oft ist es von Vorteil, wenn Teile einer Animation nicht direkt vom Animator gestaltet werden müssen, sondern prozedural erstellt werden. Soll zum Beispiel eine humanoide Figur mit der Hand nach einem Objekt greifen, ist es mühselig und zeitaufwendig jeden Abschnitt des Arms einzeln zu animieren. Inverse Kinematics machen es möglich mittels der Position und Orientierung der Hand auf die Position und Rotation aller Armabschnitte zu schließen. Dies Bedarf keiner weiteren Eingabe des Animators.

Ein weiteres Beispiel für prozedurale Animation ist Physik-Simulation. Die automatische Animierung von Objekten, welche sich nach grundlegenden physikalischen Gesetzen verhalten, spart Zeit. Darüber hinaus ist sie meist genauer, als die Animierung von Hand. Ein Anwendungsfall für Physik-Simulation ist ein herabhängendes Stück Stoff, das von einer Person getragen wird.

Ein weiterer Vorteil von prozeduraler Animation ist, dass nicht alle Details zur Erstellung der Animation bekannt sein müssen. So ist es z.B. möglich, dass eine animierte Figur mit der Umgebung interagiert, obwohl diese erst zu Laufzeit des Programms bekannt ist und Nutzereingaben können mit in die Animation einspielen.

3.6 Inverse Kinematics

Der Begriff „Inverse Kinematics“ (engl.: inverse Kinematik) kommt aus der Robotik. Um den Endeffektor eines Roboterarms mit n Gelenken und Verbindungsstücken an gewünschte kartesische Koordinaten mit gewünschter Orientierung zu bewegen, wird ein Algorithmus benötigt. Ziel dieses Algorithmus ist die Errechnung der passenden Einstellung jedes beteiligten Gelenks.

Eine Kette von Gelenken und Knochen in der Skelettanimation hat die gleiche Struktur wie ein Roboterarm mit Gelenken und Verbindungsstücken. Die gleichen Techniken, welche zur Lösung von Inverse Kinematics in der Robotik eingesetzt werden, sind daher auch in der Skelettanimation anwendbar.

3.6.1 Algebraische Methode für 2 Gelenke

Wir betrachten einen Roboterarm (siehe Abbildung 3.5). Der Arm hat 2 Gelenke A und B. A ist fixiert. Am Ende des Arms ist der Endeffektor C befestigt. Nun soll C zur Zielposition C' bewegt werden. Dieses Problem kann mit dem Gesetz des Kosinus gelöst werden. Abbildung 3.6 zeigt den Roboterarm in der Zielstellung. a, b und c bilden zusammen ein Dreieck. Mit dem Gesetz des Kosinus(Gleichung 3.1) ermitteln wir die Winkel γ und α.

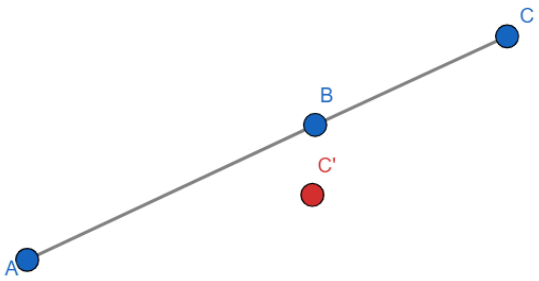

Abbildung 3.5: Ausgangsposition

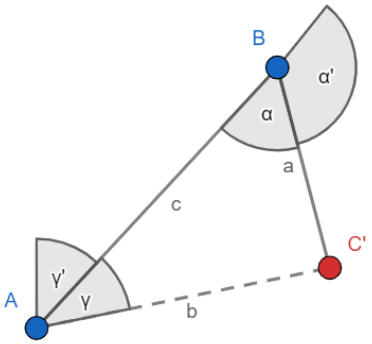

Abbildung 3.6: Zielposition

$$
\begin{aligned}
c^2 &= a^2 + b^2 - 2ab \cdot cos(\theta) \\
\theta &= cos^{-1}\left(\frac{a^2 + b^2 - c^2}{2ab}\right) \\
\gamma &= cos^{-1}\left(\frac{a^2 + b^2 - c^2}{2ab}\right) \\
\alpha &= cos^{-1}\left(\frac{c^2 + b^2 - a^2}{2cb}\right)
\end{aligned}
\tag{3.5}
$$

Die Winkel, welche wir eigentlich brauchen sind γ' und β'. Diese sind die benötigten Winkeleinstellungen für die jeweiligen Gelenke[1]. Wir berechnen die beiden gesuchten Winkel wie folgt:

$$
\begin{aligned}
\alpha' &= \pi - \alpha \\
\gamma' &= tan^{-1}\left(\frac{\vec{AC}_x}{\vec{AC}_y}\right)
\end{aligned}
$$

3.6.2 Cyclic Coordinate Descent

Cyclic Coordinate Descent (CCD) ist eine iterative Methode zur Lösung von Inverse Kinematics. Alle Gelenke welche Teil der betroffenen kinematischen Kette sind, werden nacheinander rotiert, um den Abstand zur Zielposition zu minimieren. Dabei werden alle Gelenke von der tiefsten bis zur höchsten Ebene in der Hierarchie abgearbeitet. Dieser Vorgang wird wiederholt bis der Endeffektor annäherungsweise die gewünschte Position erreicht hat.

Abbildung 3.7 zeigt einen Roboterarm, welcher aus drei Knochen und drei Gelenken besteht. A ist fixiert, D ist der Endeffektor und soll zu D_Ziel bewegt werden. Wir beginnen mit dem Knochen auf der untersten Ebene in der Hierarchie und rotieren diesen um den Punkt C, sodass D auf $\vec{CD}_{Ziel}$, also dem Vektor vom betroffenen Gelenk zum Zielpunkt, liegt (siehe

[1]Wir definieren senkrecht nach oben als den Nullwinkel und bewegen uns von hieraus im1 Uhrzeigersinn

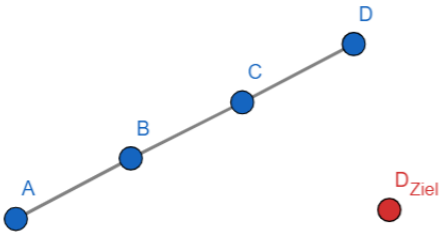

Abbildung 3.7: Ausgangsposition

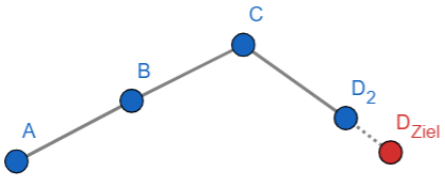

Abbildung 3.8

Abbildung 3.8). Eine Ebene weiter oben rotieren wir um den Punkt B (siehe Abbildung 3.9), dann um den Punkt A (siehe Abbildung 3.10). Wiederholen wir diese Schritte, nähert sich D mit jeder Iteration D_{Ziel} an. Der Algorithmus endet, sobald der Abstand von D zu D_{Ziel} einen vorher festgelegten Minimalabstand unterschreitet. Je nach Aufstellung kann es sein,

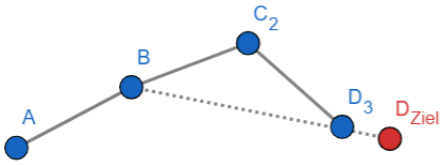

Abbildung 3.9

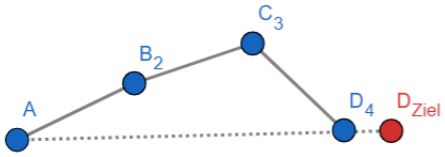

Abbildung 3.10: Zielposition

dass viele Iterationen benötigt werden bis der Algorithmus auf diese Weise endet. Jedoch sind die Berechnungen und Transformationen nicht rechenintensiv, wodurch sich CCD für interaktive Echtzeitanwendungen eignet. (vgl. [FÔ3]: 207-208)

3.6.3 Stoff-Simulation

Eine Stück Stoff lässt sich mit Hilfe des Verlet-Algorithmus simulieren. (vgl. [Cou]). Der Verlet-Algorithmus bestimmt die Position x eines Körpers zum Zeitpunkt t. v ist die Geschwindigkeit und a die Beschleunigung des Körpers. Zur Anwendung des Verlet-Algorithmus benötigen wir zwei Positionen, x_n und x_{n-1}. Ist die Position x_0 bekannt, muss zuerst die Position x_1 bestimmt werden. Dazu wenden wir folgende Formel an:

$$\vec{x}_1 = \vec{x}_0 + \vec{v}_0\Delta t + frac12\vec{a}_0\Delta t^2 \tag{3.6}$$

Durch Iteration des Verlet-Algorithmus können nun alle Positionen $\vec{x}_n$ bestimmt werden.

$$\vec{x}_{n+1} = 2\vec{x}_n - \vec{x}_{n-1} + \vec{a}_n\Delta t^2 \tag{3.7}$$

Es ist anzumerken, dass der Verlet-Algorithmus die Geschwindigkeitsberechnung aus der Formel für gleichmäßig beschleunigte Bewegung eliminiert. Das kommt uns zu Nutzen, da es die Stoffsimulation vereinfacht.

Folgender Pseudo-Code implementiert den Verlet-Algorithmus einer Punktmasse:

```
// Compute velocity from current position - lastPosition
velocity = position - last_position;

// Save position for next frame;
last_position = position;

// Find next position
position = position + velocity *  + (1/2 * acceleration * deltaTime);

// Set acceleration to gravity
acceleration = vector(0., -9.81, 0.);
```

vgl. [Wiv]

Wir definieren unser Stück Stoff als eine Ansammlung von Punktmassen, welche mit Gliedern verbunden sind. Eine Punktmasse ist ein Objekt ohne Ausdehnung. Es hat lediglich eine Masse und eine Position. Die Glieder werden in Beziehung zu den jeweils zwei dazugehörigen Punktmassen definiert. Die Länge eines Glieds ist der Abstand der beiden Punktmassen zueinander. Wir legen nun Constrainsts[2] für unser Stoffstück fest: Jedes Glied hat eine Soll-Länge, welche angestrebt wird.

Durch den Einfluss von Schwerkraft und anderen Kräften verändert sich der Abstand zwischen den Punktmassen. Um den entgegenzuwirken lösen wir die dem Stoffstück zugewiesenen Constraints[3].

Ist die Distanz zwischen 2 Punktmassen, welche mit einem Glied verbunden sind, kleiner oder größer als die Soll-Länge, dann bewegen wir beide Massen aufeinander zu bzw. voneinander weg. Das tun wir nacheinander für alle Glieder. Durch das lösen eines Constraints kommt es vor, dass ein anderer Constraint den gelösten Zustand verlässt. Das führt zu einem unglaubwürdigen Verhalten des Stoffstücks, da sich einige Bereiche dehnen, während andere steiff wirken. Der Ablauf der Simulation ist zudem stark davon abhängig, welcher Constraint zuerst gelöst wird. Um diese Abweichungen zu minimieren, lösen wir die Constraints nicht nur einmal, sondern über mehrere Iterationen hinweg. Je mehr Iterationen durchgeführt werden, desto geringer ist der Fehler und desto steiffer wirkt das Stoffstück.

[2] engl. für Einschränkung, Als Constraint bezeichnen wir Bedingung z.B. den minimalen Abstand eines Objekts zu einem anderen.

[3] Ein Constraint wird als gelöst angesehen, wenn die ihm zugewiesene Bedingung erfüllt ist.

Anhang A

Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipisici elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.
Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.
Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.
Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis.
At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, At accusam aliquyam diam diam dolore dolores duo eirmod eos erat, et nonumy sed tempor et et invidunt justo labore Stet clita ea et gubergren, kasd magna no rebum. sanctus sea sed takimata ut vero voluptua. est

Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat.
Consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Glossar

LaTeX	Softwarepaket, das die Benutzung des Textsatzprogramms TeX mit Hilfe von Makros vereinfacht.
TeXstudio	LaTeX-Editor als freie Software verfügbar auf www.sourceforge.net/

Literaturverzeichnis

[Cou] https://gamedevelopment.tutsplus.com/tutorials/simulate-tearable-cloth-and-ragdolls-with-simple-verlet-integration--gamedev-519

[FÔ3] FÊDOR, Martin: Application of Inverse Kinematics for Skeleton Manipulation in Real-Time. In: *Proceedings of the 19th Spring Conference on Computer Graphics*. New York, NY, USA : Association for Computing Machinery, 2003 (SCCG '03). – ISBN 158113861X, 203–212

[Feb18] FEBUCCI: Easing Functions for Animations. (2018). https://www.febucci.com/2018/08/easing-functions/

[Map03] MAPLE, C.: Geometric design and space planning using the marching squares and marching cube algorithms. In: *2003 International Conference on Geometric Modeling and Graphics, 2003. Proceedings*, 2003, S. 90–95

[Wik22] WIKIPEDIA: Marching squares. (2022). https://en.wikipedia.org/wiki/Marching_squares

[Wiv] https://de.wikipedia.org/wiki/Verlet-Algorithmus#:~:text=Der%20Verlet%2DAlgorithmus%20ist%20eine,in%20der%20theoretischen%20Chemie%20verwendet.