Ephemerale Rigs – Ein Lösungsansatz für komplexe Bewegungen

Markus Hadinger

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Digital Arts

in Hagenberg

im September 2018

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 21. September 2018

Markus Hadinger

Inhaltsverzeichnis

Erklärung iii

Kurzfassung vi

Abstract vii

1 Definition von Rigs 1
- 1.1 Die drei Systeme eines Rigs . . . 1
  - 1.1.1 Control System . . . 2
  - 1.1.2 Motion System . . . 2
  - 1.1.3 Deformation System . . . 3
- 1.2 Anforderungen . . . 3
  - 1.2.1 Bilder pro Sekunde . . . 3
  - 1.2.2 Automation gegen Flexibilität . . . 4
  - 1.2.3 Controller Design . . . 4

2 Matrizen in der Computergrafik 6
- 2.1 Transformation von Matrizen . . . 7
  - 2.1.1 Einheitsmatrix . . . 7
  - 2.1.2 Inverse Matrix . . . 7
  - 2.1.3 Skalierungsmatrix . . . 7
  - 2.1.4 Rotationsmatrix . . . 8
  - 2.1.5 Translationsmatrix . . . 9
  - 2.1.6 Einschränkungen von Matrizen . . . 9
- 2.2 Matrizen und Hierarchien . . . 10

3 Die technischen Grundlagen von Rigs in Autodesk Maya 12
- 3.1 Nodes . . . 12
  - 3.1.1 Dependency Graph Nodes . . . 12
  - 3.1.2 Directed acyclic graph Nodes . . . 13
- 3.2 Graph . . . 14
- 3.3 Constraints . . . 14
- 3.4 Joints . . . 15

4 Die Problematik der komplexen Bewegung 17
- 4.1 Prozedurale Rigs . . . 20

4.1.1 Vorteile . . . 21
4.1.2 Probleme . . . 21
4.1.3 Erweiterung . . . 22
4.2 Wozu Hierarchien? . . . 22
4.3 Wozu Interpolation? . . . 23
4.3.1 Animation in Splined . . . 25
4.3.2 Animation in Stepped . . . 25
4.3.3 Interpolationslose Animation . . . 26
4.4 Zusammenfassung . . . 27

5 Ephemerale Rigs **29**
5.1 Hierarchielose Rigs . . . 29
5.2 Erkenntnisse und Verbesserungen des prozeduralen Ansatzes . . . 30
5.3 Technische Ansätze . . . 31
5.3.1 Informationsfluss . . . 31
5.3.2 Logische Hierarchien . . . 32
5.3.3 Traversieren im Rig . . . 33
5.4 Animationsverhalten . . . 35
5.4.1 Animation mit hierarchischen Rigs . . . 36
5.4.2 Vor- und Nachteile bei der Animation . . . 38
5.4.3 Nützliche Hilfsprogramme für ephemerale Rigs . . . 39
5.5 Zusammenfassung . . . 41

6 Fazit **44**

Quellenverzeichnis **46**
Literatur . . . 46
Audiovisuelle Medien . . . 46
Software . . . 47
Online-Quellen . . . 47

Kurzfassung

Diese Arbeit beschäftigt sich mit einem für Animatoren optimierten Lösungsansatz für Bewegungen, bei denen der Rotationspunkt eines Charakters stark wechselt, wie bei Handständen, Saltos oder parkour-ähnlichen Bewegungen. Zuerst müssen einige Begrifflichkeiten und die dahinter liegende Mathematik erklärt werden, die im Zuge der Problemanalyse notwendig sind. Dazu zählen zum einen die Einteilung von Rigs, deren Anforderungen zum anderen aber auch die technische Umsetzung und die grundlegenden Bausteine eines Rigs. Das Hauptproblem bei den vorher erwähnten Bewegung kommt von starren Hierarchien in einem Rig, die den Wechsel von Rotationspunkten unmöglich machen. Um Hierarchien im dreidimensionalen Raum behandeln zu können, werden Matrizen und ihre Rollen als Beschreibungen von Räumen erklärt, wie sie in Rigs verwendet und manipuliert werden können. Diese Arbeit behandelt zwei Lösungsansätze im Detail. Zum einen ein prozedurales Rig, das zum Zeitpunkt der Animation verändert werden kann, zum anderen ein ephemerales Rig, das diese Idee erweitert. Ephemerale Rigs verzichten komplett auf Hierarchien und sind deswegen für Bewegung mit wechselnden Rotationspunkten geeignet. Da ohne Hierarchien animiert wird tritt jedoch auch ein Problem mit Interpolation auf. Rotationen können nur interpoliert werden, wenn Hierarchien vorhanden sind. Ohne diese Hierarchien wird jede Bewegung gerade und nicht kreisförmig interpoliert, was die Frage nach der Notwendigkeit von Interpolation hervorruft.

Abstract

The following thesis focuses on the development of a solution for complex movements. The term *complex movement* describes motion, where the root of the character switches during the animation. An example would be parkour moves with flips an handstands. To clarify why this type of motion is a problem for a rig, certain technical requirements for a rig need to be described. The main problem animators face when creating complex motion is the fact, that the controls are in a fixed hierarchy. To understand hierarchies and how they work a basic knowledge of matrices is required and how they describe spaces. The usage of matrices in rigs is needed to build a flexible hierarchy system, which enables a more intuitive way for animators. This thesis focuses on two solutions for complex movements. One is a procedural rig, which is replaceable while animating, the other builds upon this idea which results in *ephemeral rig*. They work without hierarchies, which enables a flexible way to animate complex movements, but loses the advantage of interpolating rotations.

Kapitel 1

Definition von Rigs

Bevor über die Unterschiede von verschiedenen Rig-Systemen gesprochen werden kann, muss geklärt werden, was ein Rig ist und welche Ziele ein Rig verfolgt. Rigs dienen als Schnittstelle für den Animator zwischen dem unbeweglichen Model und der fertigen Animation. Anders formuliert, könnte man einen Rig auch als technische Lösung für Bewegungen bezeichnen. Rigs sind nicht artistisch, sondern technisch. Sie dienen nur dazu, um artistische Produkte zu erzeugen. All diese Definitionen zielen darauf ab, Rigs als Tools für Animatoren zu sehen, mit denen die gewünschten Bewegungen ohne Limitierung umgesetzt werden können. Dabei muss es sich nicht zwingend um einen Charakter handeln. Jedes System, das gesteuert von einem Animator Bewegung erzeugt, kann als Rig bezeichnet werden. Die Betonung hierbei liegt auf der Steuerung durch den Animator, da dies den Unterschied zu Simulationen beschreibt. Simulationen sind Regelwerke, die voreingestellt werden und von sich aus Bewegung erzeugen. Rigs verlangen auf der anderen Seite permanenten manuellen Input, um Output zu generieren.[1] Bei einem Rig steht die Effizienz von Animatoren im Vordergrund, nicht die erzeugte Animation.

1.1 Die drei Systeme eines Rigs

Rigs sind Anwendungsprogrammen nicht unähnlich. Beide beinhalten Möglichkeiten für Interaktion, die dann verarbeitet werden und neue Information generieren. Bei Programmen gibt es jedoch, im Unterschied zu Rigs, allgemein bekannte Designmuster [5, Kapitel 10.2], die gewisse Probleme genormt lösen oder zumindest Regeln zur Lösung vorgeben. Ein bekanntes Muster ist das *Model View Control* (MVC) [5, Kapitel 19.16]. Bei diesem Muster geht es vor allem um die Trennung verschiedener Komponenten in der Software. Ziel ist es, die Daten (Model), die Präsentation (View) und die Steuerung (Control) voneinander unabhängig zu halten, sodass sie im besten Fall einzeln ausgetauscht werden können, ohne die Funktionalität des Gesamten zu beeinträchtigen. Ein zusätzlicher, wichtiger Punkt ist die Hierarchie, in der sich die Komponenten befinden. Die Präsentation kommuniziert nie direkt mit den Daten, sondern diese werden immer

[1] Es existieren auch Hybride, wie Muskelsystem, die zum einen Eingaben von Animatoren benötigen, aber zum anderen auch eigenständig Bewegung erzeugen. Um diese Arbeit kompakt zu halten, wird auf Hybride und Simulationen nicht eingegangen und nur Rigs mit den vorher genannten Definitionen werden behandelt.

von der Steuerung weitergeleitet und manipuliert. Für Rigs gibt es derart genormte Muster nicht. Die Aufteilung des Rigs in drei Systeme, *Control, Motion* und *Deformation* (CMD) [3, S. 3], zeigen jedoch einige Parallelen zum MVC Pattern, wie in Abbildung 1.1 ersichtlich ist.

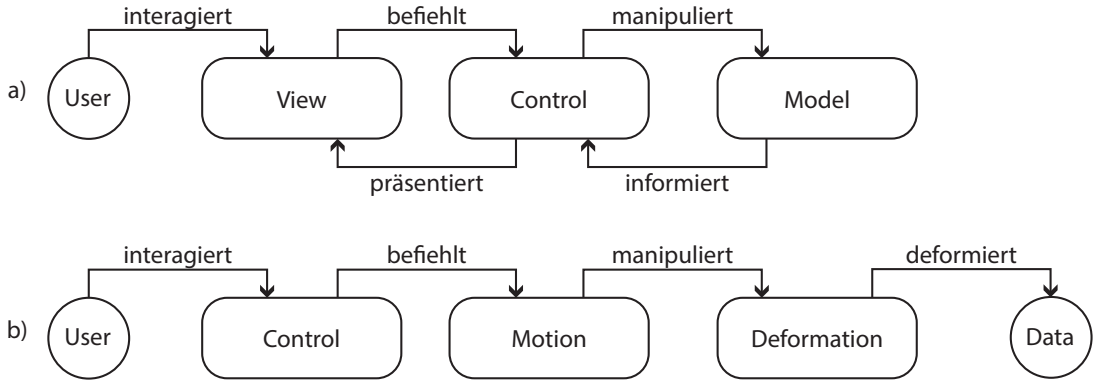


Abbildung 1.1: Zu sehen sind Parallelen und Unterschiede zwischen MVC (a) und CMD (b). Der deutlichste Unterschied ist das Fehlen der Rückverbindungen zum User und der zusätzliche Logikschritt zum Model. Rigs sind immer in einer Host-Applikation wie Maya eingebettet und brauchen daher keine Rückverbindung, um die Änderungen anzuzeigen, wie das MVC Muster. Die Anzeige der Änderungen übernimmt die Host Applikation.

1.1.1 Control System

Das Control System entspricht der Präsentationsschicht. Hier werden Eingaben für den Rig angenommen. Klassischerweise geschieht dies über Controller, bei denen einzelne Attribute manipuliert werden. Das Rotieren des Arms oder Heben des Fußes sind Beispiele für solche Eingaben. Die Eingaben müssen aber nicht zwingend von Animatoren erzeugt werden. Motion Capture Daten sind ebenfalls gültige Informationen. Durch das Wegfallen des Animators bei Motion Capture stellt sich jedoch die Frage, ob ein Rig dadurch zur Simulation werden würde. Da Animatoren die Aufgaben von Schauspielern übernehmen und Bewegungen erzeugen, und bei Motion Capture Schauspieler direkt Bewegungen erzeugen, läuft es auf das Gleiche hinaus: Eine artistische Person interagiert mit dem Control System des Rigs.

1.1.2 Motion System

Das Motion System entspricht der Steuerungsschicht. In erster Linie werden hier die Befehle der Anwender behandelt. Ein simples Beispiel wäre das vorher erwähnte Rotieren des Arms. Das Motion System liest den Befehl vom Control System ein und bewegt den korrespondierenden Joint. Dabei wird jedoch keine Veränderung an der Geometrie vorgenommen. Dieser Schritt fällt nicht in den Aufgabenbereich des Motion Systems. Das System kümmert sich ausschließlich um die Vorbereitung zur Deformation. Meistens beschreibt diese Schicht den größten und komplexesten Teil des Rigs.

1.1.3 Deformation System

Das Deformation System beschäftigt sich, wie der Name schon sagt, mit der aus den Eingaben resultierenden Deformation, meistens von Geometrie. Bekannte Vertreter sind Skin Deformer, Blendshapes oder Wraper. Diese Schicht beinhaltet meist keine selbst konstruierte Logik, sondern arbeitet mit den applikationseignen Techniken. Im Unterschied zu den beiden vorhergehenden Systemen kann das Deformer System nicht direkt mit der Datenschicht aus dem MVC verglichen werden, da die Datenschicht nur eine Ansammlung von Daten ist, in der keinerlei Berechnungen kalkuliert werden. Das Deformer System verändert die ihm unterliegenden Daten, wie Geometrie, jedoch selber.

Die Trennung in diese drei Schichten ist eher theoretischer Natur, da in der Praxis sehr viele Überschneidungen und Graubereiche auftreten. Die deformierte Geometrie kann zum Beispiel wieder als Quelle für weitere Berechnungen dienen.

1.2 Anforderungen

In der Definition des Rig, am Beginn des Kapitels 1, wird die Nähe zum Animator als wichtiger Faktor hervorgehoben. Rigs werden für Animatoren gebaut und sollten daher deren Bedürfnisse so gut wie möglich abdecken. Daraus ergibt sich eine Liste von Anforderungen, die ein Rig erfüllen muss, um als *animatorfreundlich* [34, S. 7] bezeichnet zu werden. Diese Anforderungen wechseln von Studio zu Studio und – viel wichtiger – von Animator zu Animator. Enge Zusammenarbeit und permanentes Feedback während der Entwicklung sind unerlässlich beim Bau eines Rigs [1, S. 1]. Grundsätzlich können einige einheitliche Anforderungen aufgelistet werden, die erreicht werden müssen.

1.2.1 Bilder pro Sekunde

Die Laufzeit ist eine der wichtigsten Komponenten in einem Rig. Jason Schleifer äußerte sich in seiner Serie *Animatorfriendly Rigging* [34, S. 8] dazu folgendermaßen:

> Remember this equation, because it's one of the most important ones you'll ever learn: FASTER = YES!! If you ask any animator if they'd like something faster, their answer will be an extremely loud and shocking "YES!!".

Interessanter als die tatsächliche Geschwindigkeit ist jedoch die untere Grenze, die das Rig nicht unterschreiten soll. Bewegungen können vom menschlichen Augen ab zwölf Bildern pro Sekunde gelesen werden, daher wird diese Grenze oft als absolutes Minimum gesehen [3]. Das ist aber durchaus kritisch zu sehen, vor allem in Bezug auf den technischen Fortschritt der letzten drei Jahre. Parallele Evaluierung [13], mehrere Kerne und das Auslagern von Berechnungen auf die GPU haben Rigs um ein Vielfaches schneller werden lassen. Je nach Medium wie Film oder TV sollte das Minimum bei 24 beziehungsweise bei 25 Hertz liegen.[2] Animationen müssen in Echtzeit gesehen werden. Timing und Rhythmus sind Grundsteine jeder Animation. Muss nach jeder Änderung die Animation erst als Vorschau exportiert werden, verlangsamt das den Workflow ungemein. Rigs können eine so hohe Komplexität aufweisen, dass die geforderte minimale

[2] Im Spielbereich sind 30 und 60 Herz die Norm.

Framerate durch die dennoch vorhandenen technischen Limitierungen nicht erreicht werden kann. Um dem entgegenzuwirken, sind Proxy Rigs gängige Alternativen. Bei Proxy Rigs wird auf alles verzichtet, was nicht absolut notwendig ist, wie in Abbildung 1.2 gezeigt.

Abbildung 1.2: Bei einem Proxy Rig, rechts, wird auf alles unwesentliche verzichtet. Detailgeometrie wird ausgeblendet, und nur komplexere Formen, wie die Hände oder das Gesicht, werden weiterhin vom Deformation System beschrieben. Die restlichen Geometrien folgen den korrespondierenden Joints, ohne wirklich deformiert zu werden.

1.2.2 Automation gegen Flexibilität

Wie im vorherigen Abschnitt 1.2.1 erwähnt, ist Geschwindigkeit ein entscheidender Faktor, nicht nur auf die Laufzeit bezogen, sondern auch auf die Effizienz. Animatoren sollten mit so wenig Aufwand wie möglich zum gewünschten Ziel kommen. Hier sind Automationen gefragt, die aufwendige Bewegungen vereinfachen sollen. Das Problem bei Automationen ist die zweischneidige Eigenheit, dass auf der einen Seite das Control System vereinfacht wird, aber auf der anderen Seite es dadurch auch unflexibler wird. Ein Animator wird ein Rig nicht immer so verwenden, wie es vorgesehen ist, sondern auf seine persönliche Art versuchen, zum Ziel zu kommen. Hier sind Automationen hinderlich, da sie gegen den Willen des Animators arbeiten können. Der Wunsch nach so viel Automation wie möglich bei gleichzeitig so viel Einflussnahme wie möglich, ist gegensätzlich, und kann somit nie zur Gänze erfüllt werden. Die Absprache mit den Animatoren ist der Schlüssel für einen guten Kompromiss.

1.2.3 Controller Design

Zum Thema Controller Design gibt es unzählige Ansätze, bei denen keiner wirklich richtig oder falsch ist. Letztendlich hängt das Controller Design von den Vorlieben des Animators oder den Richtlinien des Studios ab. Einige Grundeigenschaften sollten jedoch

bei jeder Designwahl gegeben sein: Übersichtlichkeit und Vorhersehbarkeit. Jeder Controller soll alleine durch sein Design einer gewissen Kategorie zugeteilt werden können. Die Kategorie muss nicht auf den ersten Blick erkennbar sein, Controller mit ähnlichen Eigenschaften sollten aber ein ähnliches Auftreten haben, wie in Abbildung 1.3 gezeigt.

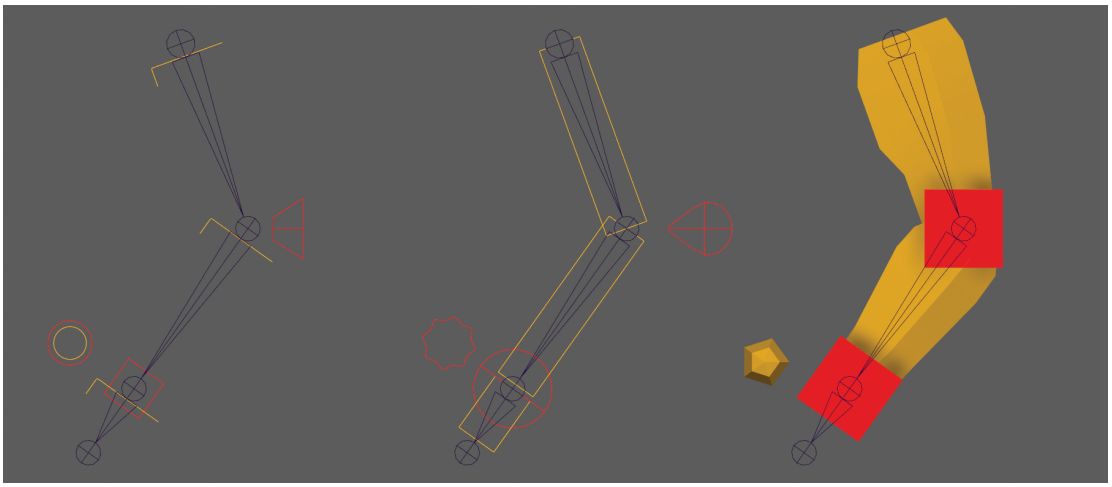


Abbildung 1.3: Drei verschiedene Varianten zum Controller Design. Jede dieser drei Varianten hat unterschiedliche Vor- und Nachteile. Die Controller links sind zurückhaltend, bieten einen guten Blick auf die deformierte Geometrie und liefern ein generell sauberes Animationsumfeld, sind aber schwieriger zu selektieren. Die Controller rechts haben die genau gegenteiligen Eigenschaften: dominant, aber selektierfreudig. Alle Varianten vereinen die klar erkennbaren Unterschiede zwischen IK und FK Controller sowohl farblich als auch geometrisch.

Die vorher erwähnte Vorhersehbarkeit bezieht sich auf das Verhalten des Rigs bei der Manipulation der Controller. Dabei gilt es, einige Fragen zu klären: können Gelenke transliert werden? Wenn ja, verhält sich dann die Translation wie eine Skalierung? Wo liegt der Unterschied zwischen Skalierung und Translation? Sollen verschiedene Manipulationen im Vorfeld verboten werden? Sind die Verbote im Rig einheitlich oder gibt es Ausnahmen? Die Liste der Fragen kann nur gemeinsam mit einem Animator beschlossen werden, und die entschiedenen Richtlinien sollten so einheitlich wie möglich realisiert werden. Das gesamte Controller Design zielt auf die Benutzerfreundlichkeit und die damit einhergehende Arbeitsgeschwindigkeit der Animatoren ab, und sollte daher mit großer Vorsicht behandelt werden.

Kapitel 2

Matrizen in der Computergrafik

In der nächsten Sektion beschäftigt sich diese Arbeit mit dem Raum, wie er definiert, und vor allem manipuliert wird. Die erste Frage, die geklärt werden muss, ist: "Was ist ein Raum?". In dieser Arbeit wird ein Raum mit einem Koordinatensystem gleichgesetzt. Das gebräuchlichste ist das kartesische Koordinatensystem, in dem die Position eines Punktes durch den Abstand zum Mittelpunkt des Systems beschrieben wird. Je nach Dimension des Raums mit zwei oder mehr Werten; X, Y, und im dreidimensionalen Raum zusätzlich noch Z. Ein Raum kann auch weitere Räume in sich beherbergen. Diese Kinderräume sind weitere Koordinatensysteme im bereits bestehenden System. Der Ursprung des Kinderraums ist der Punkt, der im Elternraum definiert wird. Diese Verschachtelung kann beliebig oft durchgeführt werden. Um die Position eines Punktes von seinem lokalem Raum in den globalen Raum, also in das äußerste Koordinatensystem, zu überführen, reicht eine Vektoraddition aller Elternräume aus, das heißt

$$P_{world} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} + ... + \begin{pmatrix} x_n \\ y_n \end{pmatrix}.$$

Ein Vektor kann in einem kartesischen Koordinatensystem auf alle Arten transformiert werden. Der Einfachheit halber wird hier nur der zweidimensionale Raum behandelt. Die Berechnungen sind sehr einfach auf den dreidimensionalen Raum überführbar. Die Gleichungen für die Translation [6, S. 30],

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x + d_x \\ y + d_y \end{pmatrix}, \tag{2.1}$$

für die Skalierung [6, S. 29],

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \cdot s_x \\ y \cdot s_y \end{pmatrix}, \tag{2.2}$$

und für die Rotation,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \cdot \cos\phi - y \cdot \sin\phi \\ x \cdot \sin\phi + y \cdot \cos\phi \end{pmatrix}, \tag{2.3}$$

weisen keine große Komplexität auf. Jede Berechnung an sich ist simpel und vor allem schnell zu berechnen, das Problem bei der Arbeit mit Vektoren liegt aber in der Unterschiedlichkeit der Gleichungen. Stattdessen wäre eine einheitliche und damit leichter optimierbare Form erstrebenswert [31, S. 17]. Hier kommen Matrizen ins Spiel.

2.1 Transformation von Matrizen

Bevor die einzelnen Möglichkeiten zur Transformation gezeigt werden, muss auf eine Eigenheit von Matrizentransformationen eingegangen werden. Skalierung und Rotation funktionieren im zweidimensionalen Raum mit einer 2×2 Matrix. Das Problem liegt in der Translation. Um Translationen berechnen zu können, muss eine zusätzliche Achse eingeführt werden, die normal zu den anderen Achsen steht. Bei zweidimensionalen Berechnungen ist die dritte die Z-Achse, im dreidimensionalen Raum die W-Achse, die schwer vorstellbar ist, aber mathematisch funktioniert. Aus diesem Grund sind die folgenden Matrizen dreidimensional, obwohl die Beispiele zweidimensional sind.

2.1.1 Einheitsmatrix

Auch wenn sich diese Arbeit nur mit den für Rigs wichtigen Teilen der Matrixberechnung befasst, nämlich der Beschreibung von Räumen, so ist eine Matrix dennoch besonders hervorzuheben. Die Einheitsmatrix [6, S. 37] ist ein Sonderfall der Matrizen, da eine Multiplikation mit ihr das Ergebnis nicht verändert – ähnlich einer Multiplikation mit eins. Die Einheitsmatrix lautet

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.4}$$

Der *World Space*, der äußerste Raum, hat als Matrix die Einheitsmatrix. Er manipuliert nichts, sondern dient nur als Basis für die Manipulationen der Kinder in ihm.

2.1.2 Inverse Matrix

Eine weitere besondere Matrix ist die inverse Matrix. Eine inverse Matrix beschreibt den Raum als sein Negativ. Eine inverse Matrix kann mit dem Kehrwert eines Skalars verglichen werden. Wie die Multiplikation eines Skalars mit seinem Kehrwert eins ergibt, so ergibt die Multiplikation einer Matrix mit ihrer inversen Matrix die Einheitsmatrix

$$I = M \cdot M^{-1}. \tag{2.5}$$

2.1.3 Skalierungsmatrix

Die simpelste Matrix ist die Skalierungsmatrix [6, S. 35]. Beim Skalieren werden alle Punkte um einen Skalierungsfaktor verschoben. Jede Achse eines Vektors wird mit dem dazugehörigen Faktor multipliziert, wie in Gleichung 2.2 gezeigt wurde. Diese Multiplikation kann auch als Matrix

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \cdot \begin{pmatrix} d_x & 0 \\ 0 & d_y \end{pmatrix} \tag{2.6}$$

geschrieben werden. Die Skalierungsmatrix entspricht in ihrer Form der Einheitsmatrix, wie in Abschnitt 2.1.1 beschrieben. Durch den Skalierungsfaktor werden Vektoren, wie in Abbildung 2.1 gezeigt, linear verschoben.

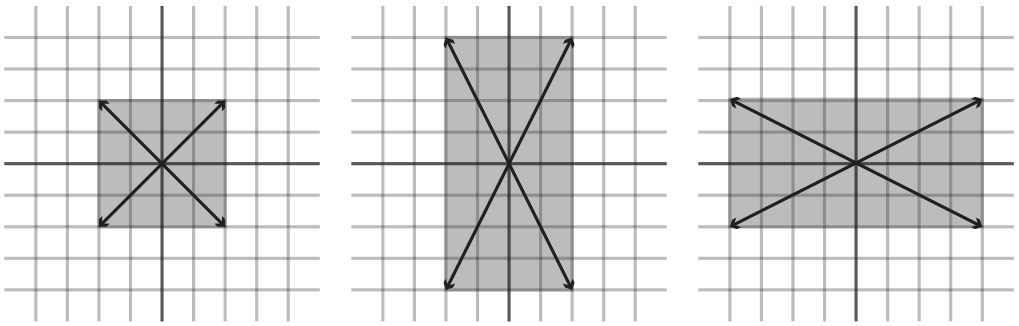


Abbildung 2.1: Ein Quadrat, beschrieben durch vier Vertices oder Vektoren, wird mit einer Skalierungsmatrix multipliziert. Die ursprünglichen Dimensionen von zwei mal zwei Einheiten ändern sich je nach Skalierungsmatrix auf zwei mal vier, wenn dy zwei entspricht, oder vier mal zwei, wenn dx zwei entspricht.

2.1.4 Rotationsmatrix

Auch die Rotation [6, S. 35], die in Gleichung 2.3 gezeigt wurde, kann als Matrix

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \cdot \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \tag{2.7}$$

angeschrieben werden. Eine Rotation setzt sich letztendlich aus zwei Komponenten zusammen: zum einen einer Skalierung um den Kosinus von α, zum anderen einer Skalierung um den um 90° versetzten positiven, beziehungsweise negativen Sinus von α. Die Kombination aus den beiden Funktionen ergibt die Rotation.

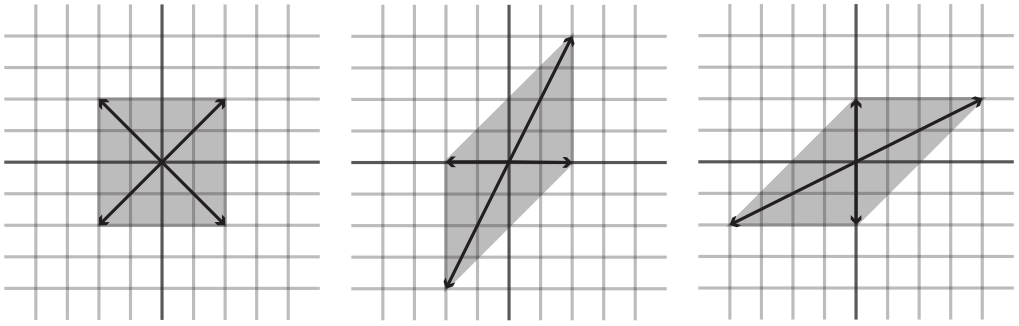


Abbildung 2.2: Die negative Sinus-Funktion ist für eine Verzerrung entlang der x Achse verantwortlich, während der positive Sinus entlang der y Achse verzerrt. Wichtig ist, dass es sich um eine zum Koordinatensystem parallele Verschiebung handelt. Die Kombination der parallelen Verschiebung mit der Skalierung um den Kosinus ergibt die Rotation.

2.1.5 Translationsmatrix

Translationen funktionieren in der Matrixschreibweise nicht als Multiplikation, solange die Matrizen dieselbe Dimension aufweisen wie das Koordinatensystem. Hier würden Translationen als Addition funktionieren, was die einheitliche Berechnung der Transformationen verwerfen würde. Wie in Abbildung 2.3 gezeigt, ist die Verschiebung entlang einer Achse allerdings mit Multiplikationen möglich. Die Idee in der Translation liegt in der Erweiterung der Matrix und des Koordinatensystems um eine Dimension. Diese Dimension steht normal zu den beiden bereits vorhanden Achsen, wie die z Achse normal auf x und y steht. Zusätzlich dürfen Vertices in der neuen, zusätzlichen Dimension nicht null sein, da sonst kein Effekt eintritt. Der Vertex wird in seiner neuen Achse um plus eins verschoben, um bei positiver Änderung positive Ergebnisse zu liefern. In Abbildung 2.3 ist deutlich zu sehen, dass eine Seite eine positive, während die andere eine negative Verschiebung beinhaltet. Deswegen müssen alle Vertices plus eins in der zusätzlichen Dimension als Wert besitzen, da sonst die Translation nicht gleichmäßig wäre.

Wird ein Punkt an der Position (x, y) um (d_x, d_y) verschoben, so lässt sich das in der Matrix

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.8}$$

niederschreiben [6, S. 35]. Ein angenehmer Nebeneffekt bei dieser Schreibweise ist, dass eine Verschiebung um $(0, 0)$ in der Einheitsmatrix resultiert, und keine Änderung hervorruft.

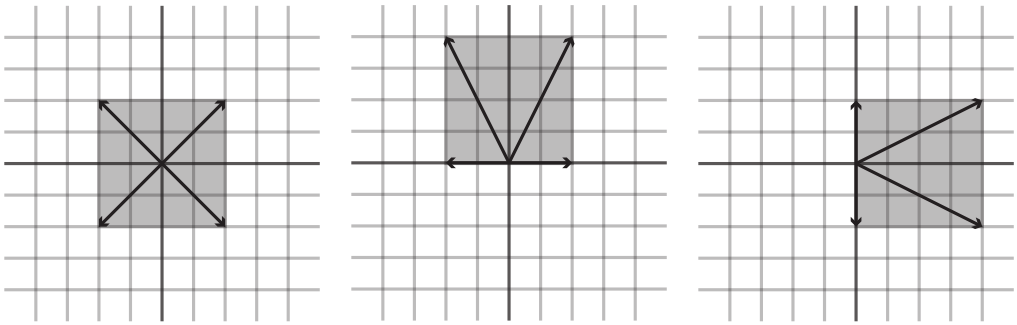


Abbildung 2.3: Bei der Translation werden alle Vertices gleichmäßig linear verschoben.

2.1.6 Einschränkungen von Matrizen

Auch wenn Matrizen eine große Liste von Vorteilen bringen, so haben sie dennoch eine Schwachstelle, welcher man sich beim Bauen eines Rigs bewusst sein sollte. Durch die Verwendung von Sinus und Kosinus entsteht ein Alias-Effekt, da die Berechnung für eine Rotation um 180° die gleichen Ergebnisse liefert wie −180°. Auch kann kein Unterschied zwischen 360° und 0° erkannt werden. Die Unterarmdrehung ist ein gutes Beispiel für

diese Problematik. Um eine gleichmäßige Drehung zu erhalten, besteht der Unterarm aus mehreren Joints, die den Twist des Handgelenks gewichtet nachahmen. Die Rotation kommt von der World-Matrix eines Controllers. Der Controller wird auf $-181°$ rotiert. Das Handgelenk hat optisch die richtige Rotation, da $179°$ gleich aussieht wie $-181°$. Der Unterarm flippt jedoch, da auf einmal die Rotationsrichtung des Unterarms wechselt.

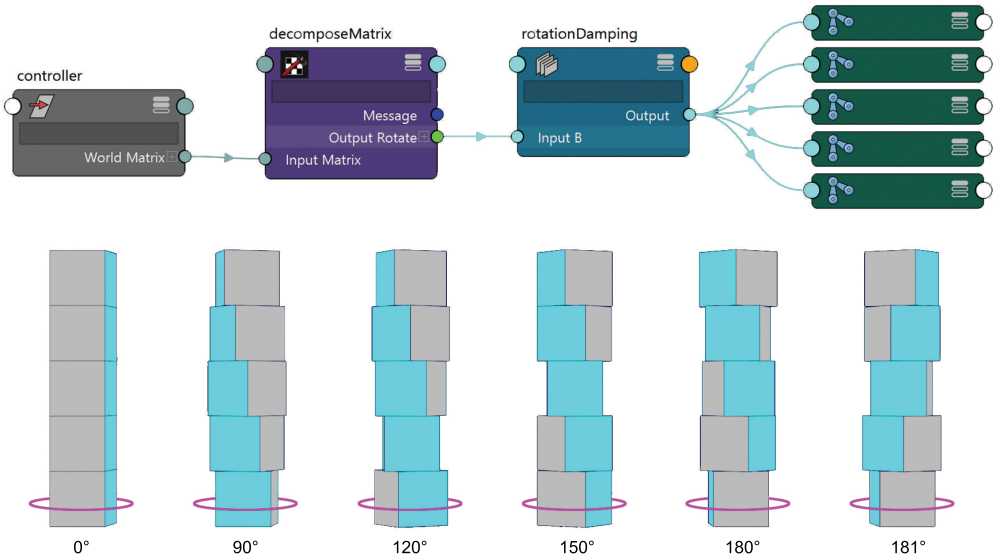


Abbildung 2.4: Der Node Graph der Software Autodesk Maya (oben) zeigt den Informationsfluss. Die Matrix des Controllers wird auf Translation, Rotation und Skalierung aufgesplittet. Die Rotation wird gedämpft an die Joints weitergegeben. Die Joints befinden sich in einer Hierarchie, die Rotationen addieren sich, und das letzte Glied hat immer die gleiche Rotation wie der Controller. Die unten gezeigten Ergebnisse zeigen die Rotationen der Joints bei unterschiedlichen Rotationen des Controllers. Von links nach rechts sind dies $0°$, $90°$, $120°$, $150°$, $180°$ und $181°$. Deutlich ist hier der Unterschied zwischen $180°$ und $181°$ zu sehen.

2.2 Matrizen und Hierarchien

In einer Matrix kann ein Raum[1] in seiner Gesamtheit – Translation, Rotation und Skalierung – beschrieben werden. Die genaue Mathematik von Matrizen würde den Rahmen dieser Arbeit sprengen, in dem Buch *Mathematische Grundlagen der Computergrafik* [2] wird genauer darauf eingegangen. Um von einem beliebigen lokalen Punkt in einem mehrfach verschachtelten Raumkonstrukt die globale Matrix, auch *World Matrix* genannt, zu erhalten, müssen alle lokalen Matrizen der Elternräume des Punktes miteinander multipliziert werden.

[1] Eine Matrix beschreibt eine Transformation. Liegen Objekte hierarchisch ineinander, so spannen diese Transformationen ein neues Koordinatensystem für die Kindelemente auf. Eine Transformation kann also auch als Definition eines Raums angesehen werden.

$$M_{world} = M_1 \cdot M_2 \cdot .... \cdot M_n$$

Diese Formel gibt die Matrix eines Punktes mit all seinen vererbten Manipulationen an [2, S.69]. Translation, Rotation und Skalierung ergeben die Matrix, die die lokale Transformation beschreibt. Wird die Matrix mit der Parent Matrix multipliziert, ist das Ergebnis die World Matrix

$$M_{world1} = M_{child1} \cdot M_{parent1}.$$

Die World Matrix wird dann als Parent Matrix für die darunterliegenden Kinder verwendet

$$M_{world2} = M_{child2} \cdot M_{world1}.$$

Die inverse Matrix kann verwendet werden, das Eltern-Koordinatensystem eines Kindes zu berechnen. Dazu wird die *World Matrix* des Kindes mit der inversen Matrix des Kindes multipliziert. Diese Gleichung

$$M_{parent} = M_{world} \cdot M^{-1}$$

ergibt die *Parent Matrix*.

Kapitel 3

Die technischen Grundlagen von Rigs in Autodesk Maya

In diesem Kapitel werden einige Begriffe und Techniken erklärt, die im Laufe der restlichen Arbeit erwähnt oder für das Verständnis benötigt werden. Auch wenn die Theorien, die in dieser Arbeit erwähnt werden, programmunabhängig sind, so ist Rigging an sich von Programm zu Programm anders und baut teilweise auch auf anderen Ideen auf. Dieses Kapitel beschäftigt sich mit den Techniken, wie sie von *Autodesk Maya* [20] verwendet werden. Es dient nicht als Anleitung zum Bau eines Rigs, sondern eher der Erklärung gewisser Begrifflichkeiten und Anwendungsfällen.

3.1 Nodes

Maya basiert auf Nodes. Jedes einzelne Element in einer Szene wird durch Nodes beschrieben. Ein Node ist die kleinste berechnende Einheit und besitzt Eingänge und Ausgänge. Ändern sich die Eingänge, wird der Node evaluiert und die Ausgänge beschrieben. Als Beispiel dient der Node *addDoubleLinear*, der eine einfache Addition zwischen zwei Gleitkommazahlen ermöglicht, wie in Abbildung 3.1 gezeigt. Nodes besitzen Attribute, die entweder als Ausgang oder als Eingang verwendet werden können, je nachdem ob sie von einem anderen Node gesteuert werden, oder einen anderen Node steuern. Dabei gibt es unterschiedliche Arten von Datentypen [30]: *Double linear* für Distanzen, angegeben in Millimeter, Fuß und dergleichen, *double angular* für Winkel, angegeben in Grad oder Radianten, *time* für Zeitangaben in Sekunden und dimensionslose Attribute wie *integer, float* und *bool* für Berechnungen. Die Werte ändern sich, je nach eingestellten Einheiten, mit Ausnahme der dimensionslosen Werte. Innerhalb eines Projektes sollten alle Dateien mit den selben Einheiten arbeiten, da es sonst zu Anomalien kommen kann.

3.1.1 Dependency Graph Nodes

Dependency Graph Nodes (DG Nodes) sind Nodes, die Berechnungen tätigen und mathematische Funktionen repräsentieren, wie der vorher erwähnte *addDoubleLinear*-Node. Der Name kommt von den Abhängigkeiten, in denen Nodes zueinander stehen

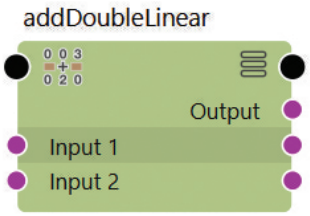


Abbildung 3.1: Input 1 und Input 2 sind die Eingabewerte für den *addDoubleLinear*-Node. Ändert sich einer der beiden Werte, so wird der Node neu evaluiert und der Ausgang (Output) neu beschrieben. Die violetten Kreise zeigen auf der linken Seite die Eingänge, und auf der rechten Seite die Ausgänge. Erwähnenswert ist, dass die beiden Inputs sowohl Eingänge als auch Ausgänge haben, während der Output nur einen Ausgang besitzt. Der Unterschied ist, dass der Output vom Node selber beschrieben wird, und daher keinen Eingang akzeptieren kann. Input 1 und 2 werden jedoch vom Node nicht verändert, können also als Eingang und Ausgang verwendet werden, auch wenn im Ausgang immer der selbe Wert steht wie im Eingang. Solche Ausgänge, wo Werte unverändert weitergeleitet werden, machen aus logischer Sicht keinen Sinn, können aber praktisch beim Sortieren des Graphen sein, siehe Abschnitt 3.2.

können. Node C ist von den Ergebnissen von Node A und B abhängig. Diese Verbindungen bauen einen Graph auf, auf den in Abschnitt 3.2 genauer eingegangen wird. Dabei kann jeder Knoten mit jedem anderen Knoten verbunden werden. Kommen unterschiedliche Einheiten wie angular und linear zusammen, wird eine Unitconversion eingebaut, die die Datentypen wieder kompatibel zueinander macht. Logischerweise können Vektoren nicht mit Einzelwertattributen verbunden werden, da ein Vektor aus drei Werten besteht. Es können auch Schleifen gebaut werden, sprich A ist von B, B ist von C und C ist wieder von A abhängig.

3.1.2 Directed acyclic graph Nodes

Im Gegensatz zu den DG Nodes können Directed acyclic graph Nodes (DAG Nodes) nicht in einer Schleife aufgebaut werden. DAG Nodes werden zum Aufbau von Hierarchien verwendet, welche schleifenlose Graphen sind. Die beiden Vertreter der DAG Nodes sind Transform- und Shape-Nodes und werden im Outliner von *Maya* angezeigt. Shape Nodes speichern die Geometrie oder Kurveninformation eines Objektes, während Transform Nodes die Translation, Rotation und Skalierung eines Objektes speichern. Zusätzlich werden in den Transform Nodes auch die Matrizen gespeichert, die sich aus den Hierarchien ergeben. Ein Shape Node benötigt immer einen Transform Node als Elter, da der Shape Node ansonsten keine Information besitzt, wo er sich im Raum befindet. Ein Transform Node kann beliebig viele Shape Nodes und Transform Nodes als Kinder haben. In *Maya* wird eine Hierarchie über die Matrizen, die jeder Transform-Knoten besitzt, definiert, wie in Abbildung 3.2 gezeigt wird, und die in Kapitel 2 behandelt werden.

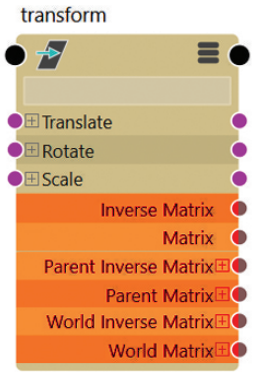


Abbildung 3.2: Jeder Transform Node in *Maya* besitzt die Attribute *Matrix*, *Parent Matrix* und *World Matrix*. Diese Attribute haben nur Ausgänge, können also nur ausgelesen und nicht beschrieben werden. *Maya* berechnet die Matrizen automatisch selbst.

3.2 Graph

Nodes, deren Eingangs- und Ausgangsattribute miteinander verbunden sind, ergeben einen Graphen. Der Graph zeigt den Informationsfluss des Rigs. Das beschreibt nicht nur mathematische Funktionen, bei denen Ergebnisse weitergeleitet werden, sondern auch die erzeugte Geometrie. Da alles in *Maya* aus Nodes besteht, wird auch Geometrie mit Nodes erzeugt. Die Geometrie kann zu anderen Nodes weitergeleitet werden, die weiter Manipulationen tätigen. Daraus entsteht ein Graph, der als *Construction History* [30] bekannt ist. Auch wenn dieser Graph anders genannt wird, so unterscheidet er sich nicht von einem Graph, der numerische Ergebnisse berechnet, wie in Abbildung 3.2 gezeigt. Ein Node evaluiert nur, wenn einer seiner Eingänge neu beschrieben wird. Dazu zählt das Erstellen einer Verbindung, oder die Änderung des Wertes am Eingang. Wird eine Verbindung gelöscht, so wird der Node nicht neu evaluiert, und der aktuelle Wert wird beibehalten. Deswegen funktioniert beim Modellieren das Löschen der *Construction History*. Nicht jede Änderung an einem Eingang führt zur Neuberechnung aller Ausgänge. Beim Schreiben eines Nodes muss angegeben werden, welcher Eingang Einfluss auf welchen Ausgang hat. Ändert sich die Translation an einem Transformation Node, wird die lokale Matrix neu berechnet, die Parent Matrix aber nicht. Mit dieser Information kann die Geschwindigkeit eines Rigs gesteigert werden. Rigs können darauf optimiert werden, so selten wie möglich zu evaluieren.

3.3 Constraints

Mit Constraints können Transform Nodes hierarchisch unabhängig transformiert werden. Ein Objekt in einem anderen Hierarchiezweig kann so manipuliert werden, als ob es sich in der selben Hierarchie befinden würde wie das steuernde Objekt. Die drei wichtigsten Constraints sind Parent Constraints[1], Orient Constraints[2] und Point Constraints[3].

[1] Das gesteuerte Element verhält sich wie ein Kind vom steuernden Element, ohne eines zu sein.
[2] Die Rotation des steuernden Elements wird übernommen.
[3] Die Position des steuernden Elements wird übernommen.

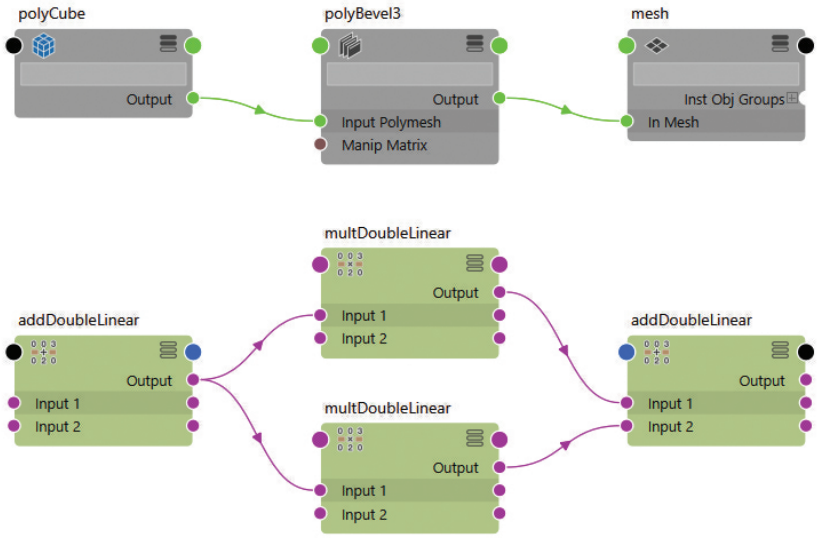


Abbildung 3.3: Geometrie (oben) erzeugt die gleiche Art von Graph wie eine mathematische Berechnung (unten). Ein Ausgang kann Information an mehrere Eingänge weiterleiten, aber ein Eingang kann nur Information von einem Ausgang annehmen.

Constraints arbeiten mit der Parent Matrix der Objekte, um mit unterschiedlichen Hierarchien umgehen zu können. Die Parent Matrix gibt die World Matrix des Elternobjektes an, dadurch befinden sich die beiden Objekte vom Constraint aus gesehen im selben Raum, dem Gloabl Space. Constraints haben einen großen Nachteil. Da sie mit der Parent Matrix arbeiten, evaluieren sie immer, wenn sich in der Hierarchie, in der sie liegen, etwas ändert, da diese aktualisiert wird. Das ist zum einen der Sinn von Constraints, da sie sonst kein richtiges Ergebnis liefern, zum anderen aber oft nicht nötig. Wird zum Beispiel die Rotation vom Hand Controller auf den Handjoint mit einem Parent Constraint übertragen, evaluiert dieser, wenn der Torso bewegt wird. Dabei hat sich an der lokalen Rotation der Hand nichts geändert. Es ist wesentlich effizienter, statt Constraints direkte Verbindungen zu bauen, und die Hierarchie von Joint und Controller gleich zu halten, als jeden Joint mit einem Constraint zu steuern. Abbildung 3.4 zeigt den Aufbau eines Constraints im Node Editor. Joints sind vor allem für Einsteiger im Thema Rigging interessant, da schnell und einfach komplexes Verhalten erzeugt werden kann, und Matrizen am Anfang überfordernd sind. Aus diesem Grund kommen sie auch in den meisten Einsteigerbüchern wie *Rig it Right* [4, S. 45] vor.

3.4 Joints

Als letzter Punkt werden hier nun Joints erklärt. Joints sind eine Erweiterung von Transform Nodes und besitzen daher die meisten Attribute und Eigenschaften, die ein Transform Node besitzt. Joints werden im Vergleich zu Transform Nodes um einige zusätzlich Attribute, die besonders für Skinning oder inverse Kinematik (IK) benötigt

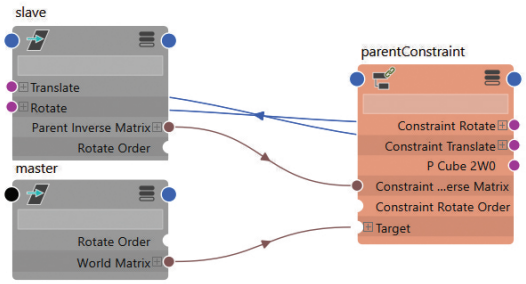


Abbildung 3.4: Der Constraint benötigt die *World Matrix* des Masters und die Parent Invers Matrix des Slaves. Multipliziert man die beiden Matrizen miteinander, so erhält man die lokale Matrix des Masters in dem Raum, indem sich der Slave befindet. Aus der lokalen Matrix lassen sich dann Translation, Rotation und Skalierung errechnen, die der Slave benötigt, um an der selben Position wie der Master zu sein.

werden, erweitert. Dazu zählen *Preferrd Angle*, das die Knickrichtung im IK angibt, oder *Bind Pose*, in dem die Transformationsmatrix des Joints zum Zeitpunkt des Skinnings gespeichert wird. Der wichtigste Unterschied für Rigger liegt jedoch im Einführen einer Joint Orientation und einer inversen Skalierung. Die Joint Orientation ist eine vorgelagerte Rotation, die die Rotation des Joints in seiner Ausgangsphase beschreibt. Der Joint besitzt also eine Rotation, ohne dass diese im Attribut *Rotation* auftaucht. Durch diese vorgelagerte Rotation läuft der Joint seltener in die Situation eines Gimbal Locks, welcher in Abbildung 5.6 erklärt wird. Die inverse Skalierung hat den Effekt, dass skalierte Eltern den Raum des Joints nicht verzerren. Die Skalierungsmatrix des Elternjoints wird invers in die lokale Matrix multipliziert. *Maya* baut diese Beziehung automatisch beim Erstellen von Jointketten auf. Joints haben von sich aus keinen Shape Node, können also beim Rendern nicht angezeigt werden. Es können ihnen aber Shape Nodes hinzugefügt werden, wie beispielsweise Kurven, was sie auch als Ausgangsnode für Controller interessant macht. Durch die Joint Orientation können Rotationen, die für den Animator unwichtig sind, ausgelagert werden, und die inverse Skalierung macht Controller Hierarchien bei dehnbaren Gliedmaßen stabiler und lesbarer.

Kapitel 4

Die Problematik der komplexen Bewegung

Die Hauptfunktion von Rigs liegt in der Ermöglichung von Animationen. Das bedeutet, je besser ein Rig zu animieren ist, desto besser ist das Rig als solches. In Kapitel 1 sind die verschiedenen Anforderungen an Rigs aufgelistet. Die beiden wichtigsten für dieses Kapitel sind Geschwindigkeit und Bedienbarkeit. Bei komplexen Bewegungen stehen beide Aspekte gegeneinander. Als Beispiel wird ein kurzes Storyboard, Abbildung 4.1, analysiert, indem einige Problematiken auftauchen. Die Problematik in dieser Ani-

Abbildung 4.1: Das Storyboard beschreibt eine Animation, in der einige, komplexe Bewegungen vorkommen. Es folgt eine kurze Erklärung der einzelnen Bilder, von links nach rechts. Der Bösewicht schleudert den Helden an den Füßen davon. Der Held fliegt durch die Luft. Der Held schafft es, sich an einem Rohr an der Decke festzuhalten. Mit Hilfe des Schwungs legt er einen eleganten Salto hin. Er landet wieder auf seinen Beinen.

mation liegt in den wechselnden Punkten, um die sich der Held dreht. Im ersten Bild schleudert der Bösewicht den Helden an den Füßen davon. Die Reihenfolge der Gliedmaßen in der Abhängigkeit lautet folgendermaßen. Die Füße des Bösewichts sind am Boden verankert. Die Füße tragen den Körper des Bösewichts. Seine Hände sind an seinem Körper befestigt und halten die Füße des Helden. Der Körper des Helden hängt an seinen Füßen. Die Arme des Helden sind das letzte Glied der Kette und hängen am Körper des Helden. Im starken Kontrast dazu steht das dritte Bild, wie in Abbildung 4.2 gezeigt wird. Die Hände des Helden sind am Rohr fixiert, der Held hängt an seinen Händen und die Füße sind das letzte Glied der Kette. Der Held hat zwei gegenläufige Hierarchien, in unterschiedlichen Shots. Die Animation wird noch komplizierter, wenn sich der Wechsel im selben Shot befindet. Als dritter Fall sind noch Bild zwei und vier

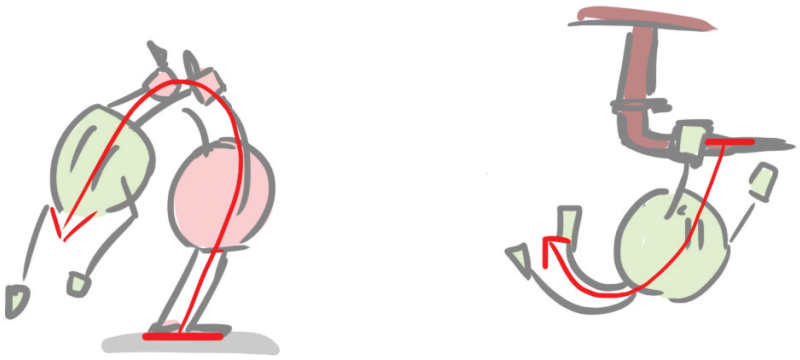


Abbildung 4.2: Der Held hat in unterschiedlichen Shots gegenläufige Hierarchien oder Abhängigkeiten. Laufen sie zuerst durch die Hände des Bösewichts über die Füße bis in die Hände des Helden, so wechselt im anderen Shot die Richtung. Nun hängt der Kopf an den Händen und bildet seinerseits wieder die Verankerung für die Füße.

in Abbildung 4.1 übrig. Hier hängen Hände und Füße am Körper des Helden. Bewegt er sich durch die Luft, ohne an irgendetwas gebunden zu sein, so ist der wichtigste Punkt der Schwerpunkt, von dem alle anderen Körperteile abhängig sind.

Klassische Rigs sind von der Hierarchie dem ersten Bild am ähnlichsten, mit einer Änderung: Die Reihenfolge bei Rigs lautet Hüfte, Oberschenkel, Unterschenkel, Knöchel, Ballen und zum Schluss Zeh, also genau entgegen der Abhängigkeiten beim Stehen. Etabliert hat sich diese Reihenfolge dank der inversen Kinematik. Es ist leichter, den gesamten Torso zu animieren, da hier die meiste Masse sitzt, und die Winkel der Beine selbstständig berechnen zu lassen. Schließlich stehen Menschen auch auf, indem sie ihr Becken heben, und nicht, indem sie die Beine ausstrecken. Das trifft natürlich nur aus der Sicht von Animatoren zu, anatomisch stimmt diese Aussage nicht. Für Animatoren ist die Position des Beckens wichtiger als der Winkel der Beine, als logische Übersetzung wird das in der Animation ebenfalls so gehandhabt. Inverse Kinematik benötigt in ihrer reduziertesten Form nur zwei Punkte: den Startpunkt, bei Beinrigs ist das die Hüfte, und einen Endpunkt, den Knöchel. Zusätzlich kann noch die Richtung, in die das Knie zeigt, angegeben werden, dies ist aber für die reine Funktionalität nicht ausschlaggebend. Als Ergänzung können Gliedmaßen meistens in zwei Modi animiert werden: der bereits erwähnten inversen Kinematik, und der Vorwärtskinematik. Die Vorwärtskinematik beschreibt eine hierarchische Bewegung, bei der das Elternelement das Kindelement beeinflusst. Im ersten Bild des Storyboards hängt der Held an den Händen des Bösewichts. Bewegt dieser seinen Arm, so bewegt sich der Held. Das ist ein klassisches Beispiel für Vorwärtskinematik. Wenn sich Objekte mit nur einem Fixpunkt bewegen, so wird meist diese Form der Kinematik angewandt. Ausnahmen bilden unter anderem die technischen Vorlieben der Animatoren.

Was definiert nun eine komplexe Bewegung?[1] Diese Arbeit beschäftigt sich mit den Rigging-Aspekten, daher ist es naheliegend, Bewegungen auch aus der Sicht eines Rigs zu

[1] Der Begriff komplexe Bewegung ist kein wissenschaftlicher Begriff. Er wird hier in dem vom Autor festgelegten Kontext verwendet.

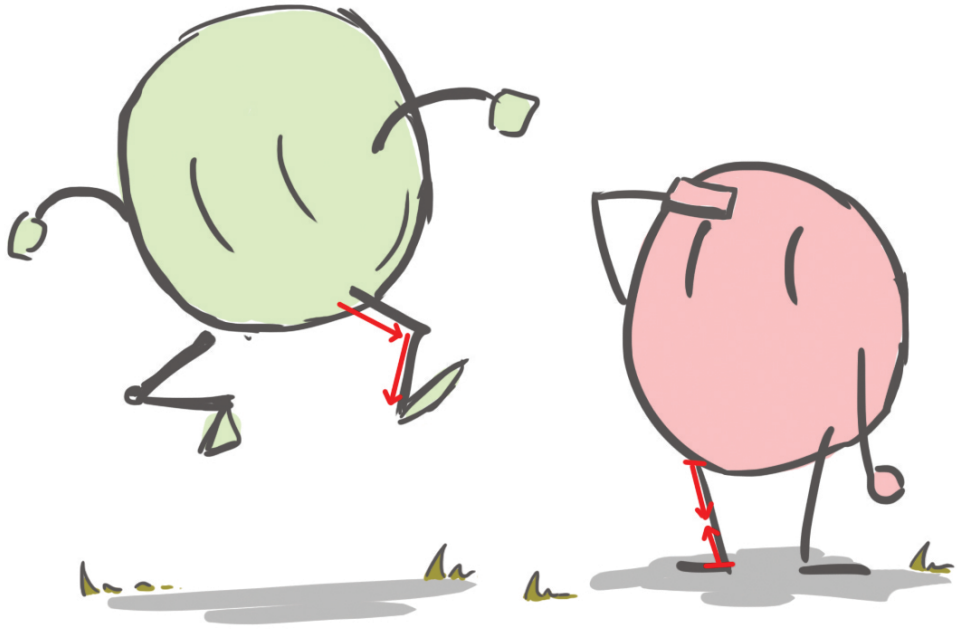


Abbildung 4.3: Abhängig von der Anzahl der Punkten an denen die Gliedmaßen stabile Berührungen aufweisen, werden unterschiedliche kinematische Methoden verwendet. Der Held auf der linken Seite befindet sich in der Luft. Seine Beine berühren nur den Torso, daher können sie frei schwingen. Hier wird die Vorwärtskinematik eingesetzt, bei der jedes hierarchische Elternelement sein Kindelement beeinflusst. Der Bösewicht auf der rechten Seite steht am Boden. Seine Beine haben zwei fixe Berührpunkte: den Torso und zusätzlich den Boden. Hier findet die inverse Kinematik ihre Anwendung, bei der die Winkel zwischen den einzelnen Gelenken automatisch berechnet wird, um den Endpunkt, den Boden, zu erreichen.

sehen. Komplexität tritt dann auf, sobald eine Bewegung entgegen der Joint-Hierarchie vollführt wird. Das einfache Gehen ist somit keine komplexe Bewegung. Auch das zweite und vierte Bild des Storyboards, bei denen der Held durch die Luft fliegt, zeigen keine komplexen Bewegungen. Der Schwerpunkt ist meistens der dominanteste Controller im Rig. Bei einem Salto dreht sich der gesamte Körper um diesen Schwerpunkt. Die Bewegung läuft also nicht gegen die Hierarchie. Sobald der Held im dritten Bild das Rohr festhält, arbeitet die Hierarchie der Bewegung gegen die Hierarchie der Joints. Ab jetzt kann die Bewegung nicht mehr mit den aktuellen Controllern animiert werden, da die Handcontroller Kinder des Schwerpunktcontrollers sind. Dreht sich der Schwerpunktcontroller, bewegen sich die Hände vom Rohr weg. Sind die Hände mit inverser Kinematik gesteuert, ist die Animation möglich. Einen gleichmäßigen Bogen, den der Held schwingt, zu animieren, ist jedoch Handarbeit und muss für jeden Frame einzeln angepasst werden. Um die Problematik komplexer Bewegungen noch einmal zusammen zu fassen: Ein Rig alleine kann nicht alle möglichen Fälle abdecken. Was, wenn der Charakter auf seinen Knien sitzt und sich nach vorne beugt? Was, wenn er am Brustkorb festgehalten wird und sich heftig gegen dies Umarmung wehrt? Was, wenn der Charakter sich an den Kopf greift? Die Punkte, die mit etwas Dominanterem verbunden

sein können, sind unzählig, und somit würden die Komplexität und damit einhergehend die Geschwindigkeit des Rigs einbrechen. Auf der anderen Seite benötigen Animatoren allerdings diese Freiheit, um das bestmögliche Ergebnis zu erzielen. Technische Limitierungen sollten nicht die Animation beeinflussen.

4.1 Prozedurale Rigs

In einer gängigen Produktion hat ein Charakter einen, für Sonderfälle vielleicht zwei oder drei Rigs. Diese Rigs sind so konzipiert, dass sie möglichst viele Fälle abdecken können. Wie zuvor erwähnt wurde, verlangsamt Komplexität Rigs. Die logische Schlussfolgerung ist demnach, Rigs so simpel wie möglich zu halten. Simple Rigs erfordern allerdings wieder mehrere Varianten des Rigs für unterschiedliche Situationen. Betrachtet man all diese gegenläufigen Aspekte, erkennt man, dass es keine perfekte Lösung gibt. Ein Kompromiss ist jedoch der Prozedurale Rig. Bei dieser Idee, vorgestellt von Richard Lico

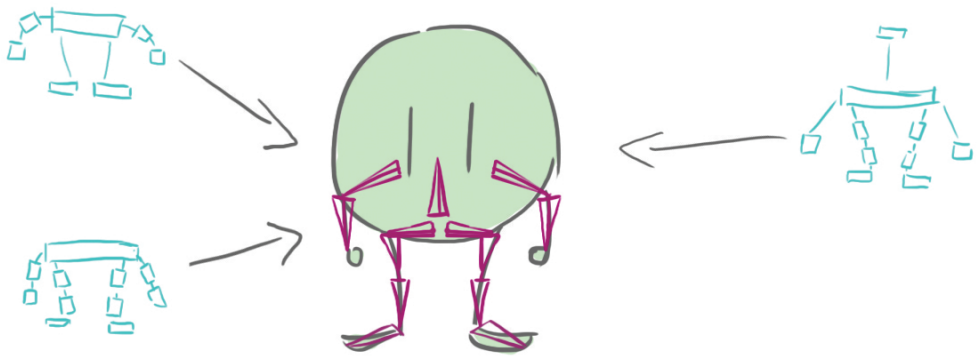


Abbildung 4.4: Das Deformation System ist in allen Shots und Animationen das selbe. Es kann allerdings von unterschiedlichen Control und Motion System angesteuert werden. Je nach Bedarf können so auch Teile der Animation mit einem anderen Rig überschrieben werden. Als Zwischenspeicher für die Animationsdaten dienen die Joints selber, was Sinn macht. So können Animationsdateien auch ohne Rig gespeichert werden, was die Größe der Daten im Zaum hält.

bei der GDC 2018 [33] für das Spiel *Moss*, wird nicht mit einem fixen Rig gearbeitet. Stattdessen wird das Rig mithilfe eines Skripts erstellt, und kann jederzeit neu generiert werden. Um diese Technik zu verstehen, wird die Auftrennung in die verschiedenen Schichten eines Rigs, wie in Kapitel 1 erklärt, wieder wichtig. Das Rig besteht nur aus dem Deformation System. Joints, Geometrie und Skinning sind vorhanden, da dies die einzigen Elemente sind, die bei jeder Animation einheitlich bleiben. Das System, das die Joints bewegt und damit die Geometrie deformiert, ist jedoch flexibel wie in Abbildung 4.4 gezeigt. Rigs können jederzeit ausgetauscht werden. Um die Animationsdaten nicht zu verlieren, wird vor dem Austauschen die aktuelle Animation auf jeden einzelnen Joint und jeden einzelnen Frame gespeichert. Ist der neue Rig geladen, wird die Animation von den Joints wieder auf den neuen Rig übertragen. Dieser neue Rig hat andere Eigen-

schaften wie der vorhergehende, wie zum Beispiel eine andere Hierarchie der Kontrollen, bewegt sich aber identisch. Mit diesem neuen Rig können Teile der Animation, die mit der ersten Version nicht möglich waren, überschrieben werden.

Dieser Lösungsansatz lässt jedoch außer Acht, dass die drei Systeme eines Rigs oft nicht klar getrennt werden können. Es gibt Teile eines Rigs, die nicht wirklich von den Animatoren, sondern von den Deformation Joints abhängig sind. Hilfs-Joints sind hier ein gutes Beispiel: Joints, die nicht selber die Animation tragen, sondern für eine saubere Deformation zuständig sind, wie zum Beispiel zusätzliche Joints im Ellbogengelenk, die vom Winkel des Ellbogens abhängig sind und den Knochen betonen, Joints, die Muskeln nachahmen, und dergleichen. Diese Joints werden nicht vom Control System angesteuert, sondern vom Motion System. Um eine saubere Trennung zwischen den wiederverwendbaren Teilen und den prozeduralen Teilen des Rigs zu gewährleisten, muss die saubere Trennung der Systeme reduziert werden, und Teile des Motion Systems werden vom Deformation System angesteuert.

4.1.1 Vorteile

Um die Anfangsproblematik nochmal aufzufrischen: Komplexe Rigs sind langsam, simple Rigs sind schnell, decken aber nur wenige Fälle ab. Vor allem bei mehreren verschiedenen Fällen in einem Shot steigt die Anzahl der Rigs rasant an. Prozedurale Rigs bilden hier eine gute Lösung, da pro Shot mehrere Rigs verwendet werden können, ohne den Overhead der nicht benötigten Rigs zu haben. Zusätzlich kann aus jeder Animation jedes Rig generiert werden. Es ist also nicht zwingend nötig, dass zwei Animatoren im selben Programm arbeiten. Wenn Animator 1 mit seiner Animation in Programm 1 fertig ist, exportiert er seine Arbeit in ein gängiges Format wie beispielsweise FBX. Animator 2 kann in Programm 2 diese Daten öffnen, das Rig generieren und weiterarbeiten. Als zweiter, bereits erwähnter Vorteil muss die Animationsdatei nicht mit einem Rig gespeichert werden. Solange die Animation auf den Joints liegt, macht es keinen Sinn, Rigs mit zu speichern, da nicht klar ist, welches Rig der richtige ist, oder genauer, es kein richtiges Rig gibt.

4.1.2 Probleme

Der folgende Abschnitt beschäftigt sich nicht direkt mit den Nachteilen dieser Technik, da es streng genommen keine Nachteile sind, sondern eher Probleme, die prozedurale Rigs nicht lösen. Auch wenn das Problem der Komplexität zur Laufzeit gelöst ist, so muss trotzdem für jeden auftretenden Spezialfall ein Rig gebaut, beziehungsweise geskriptet werden. Es können zwar durch die Möglichkeit der Kombination mehr Fälle abgedeckt werden als bei einem klassischen fixen Rig, für einen neu auftretenden Fall muss jedoch das Arsenal der Rigs erweitert werden. Je nach Komplexität der Animationen kann das eine Vielzahl an Rigs werden. Ein weiterer Nachteil von prozeduralen Rigs ist die Kompatibilität. Jeder geskriptete Rig muss mit dem Skelett des Deformation Systems zusammenpassen. Sollte im Laufe der Produktion ein Fehler auftauchen und die Hierarchie oder die Benennung im Deformation System geändert werden, so muss jedes einzelne Skript auf diese Änderung angepasst werden. Diese Änderungen bergen eine massive Fehlerquelle. Ein weiterer, diesmal wirklicher Nachteil ist die Abhängigkeit von Joints. Arbeitet ein Rig nicht mit Joints, sondern beispielsweise mit Kurven,

so funktioniert diese Technik nicht, da es keine Trägereinheit gibt, die direkt bespielt werden kann. Wird die Kurve mit Joints oder anderen Einheiten gesteuert, so ist das System wieder möglich. Werden die Punkte der Kurve allerdings direkt vom Motion System angesteuert, so gehen Informationen wie Skalierung und Rotation der Controller verloren.

4.1.3 Erweiterung

Spezielle Rigs für spezielle Bewegungen machen Sinn, da diese Rigs simpler und damit schneller sind. Dennoch sind sie nicht automatisch für jeden Fall geeignet. Grund dafür ist die nach wie vor fixe Hierarchie. Um eine universelle Lösung für jeden auftretenden Fall zu haben, muss also die Hierarchie beseitigt werden.

4.2 Wozu Hierarchien?

Nach der Analyse von komplexen Bewegungen und prozeduralen Rigs stellt sich die naheliegende Frage nach der Sinnhaftigkeit von Hierarchien. Die gesamte Problematik der komplexen Bewegung entsteht letztendlich nur aufgrund der Hierarchien im Rig. Bei all den negativen Effekten muss es eine Berechtigung geben, da sie sich ansonsten nicht etabliert hätten. Einer der direktesten Gründe für Hierarchien ist die Nähe zur Realität. Menschliche Knochen sind hierarchisch aufgebaut, und um deren Verhalten nachzuahmen, sind Hierarchien eine der leichtesten Möglichkeiten, wie in Abbildung 4.5 gezeigt wird. Hierarchien vereinfachen dank dem vererbenden Verhalten einen Rig

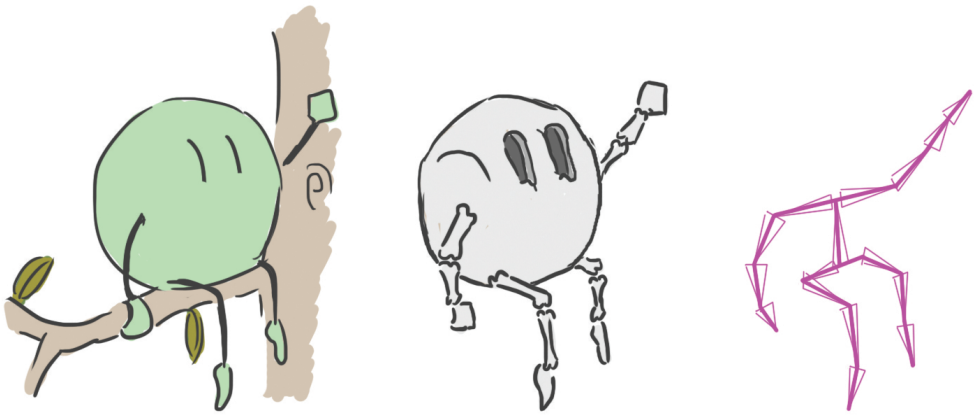


Abbildung 4.5: Je näher ein Rig an der aktuellen Anatomie eines Charakter ist, desto glaubwürdiger ist seine Bewegung. Ein Charakter kann nach belieben verzerrt werden, solange die Animation in der bereits etablierten Welt funktioniert. Deswegen ist Glaubwürdigkeit für Animation wichtiger als Realismus. Da Knochen in Hierarchien aufgebaut sind, ist es naheliegend, Joints ebenfalls in Hierarchien aufzubauen, um deren Verhalten glaubwürdig nachzuahmen.

ungemein. Wird der Körper des Charakters verschoben, ist garantiert, dass die Gliedmaßen danach korrekt sitzen, und das bei jeder möglichen Transformation. In Kapitel

2 wird die Mathematik hinter Hierarchien grob umrissen. Hierarchien bestehen in der Computergrafik aus einfachen Matrizenmultiplikationen und sind die schnellste Form der Transformation. Der große Vorteil von Hierarchien für Animatoren liegt jedoch in der Einfachheit der Rotation. Würde ein Rig ohne Hierarchien nur über Translationen gesteuert werden, wären Bewegungen wie das Schwingen eines Arms wesentlich schwieriger zu animieren, jedoch nicht schwieriger zu positionieren. Das Problem tritt erst bei der Animation, oder genauer, bei der Interpolation auf. Ein Rig ohne Hierarchien kann nicht korrekt interpolieren. Da Rotationen ohne Hierarchien nicht vererbt werden, ist es nicht möglich, Bögen automatisch zu animieren. Interpolation ist also der springende

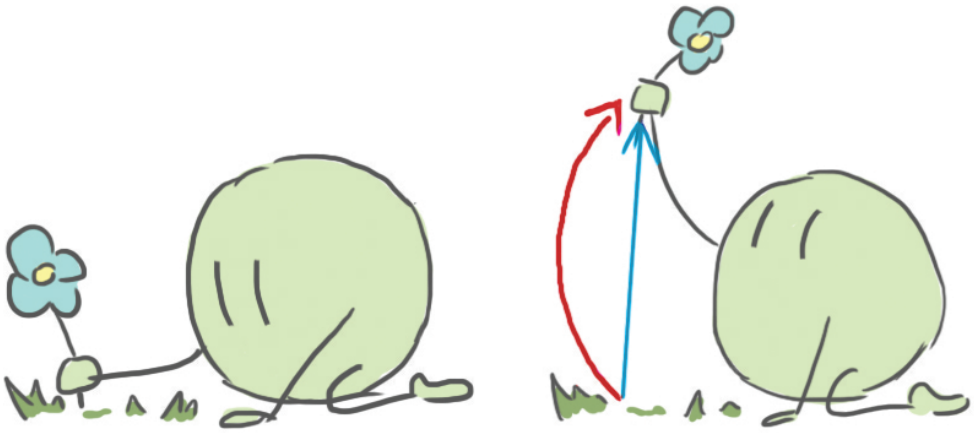


Abbildung 4.6: Wird die Hand mit hierarchischen Rotationen bewegt, beschreibt sie einen Bogen, wie er in Rot dargestellt ist. Sollten Hierarchien wegfallen, und Hand und Körper sind Geschwister, kann die Hand nur durch Translation bewegt werden. Eine Rotation würde die Hand drehen, aber nicht deren Position wechseln. Translationen sind von Natur aus linear, dadurch würde eine Interpolation ebenfalls linear sein, wie in Blau gezeichnet. Bögen müssten händisch nachgebaut werden, und würden nicht mehr automatisch erzeugt.

Punkt, warum Hierarchien in einem Rig so wichtig sind.[2] Das bedeutet, dass Hierarchien zum einen arbeitserleichternd sind, da dank Interpolation schneller animiert werden kann, zum anderen aber auch arbeitserschwerend sind, da sie komplexe Bewegungen aufwendig gestalten und daher langsamer animiert werden muss. Der nächste Schritt für eine Lösung für komplexe Bewegungen ist also nicht nur Hierarchien kritisch zu hinterfragen, sondern auch Interpolation als Ganzes.

4.3 Wozu Interpolation?

Der Begriff Interpolation kommt aus dem Lateinischen und bedeutet grob übersetzt *dazwischen glätten*. Genau das ist die Aufgabe von Interpolation in der Animation: Die Frames zwischen zwei Keys auffüllen. Dabei wird zwischen verschiedenen Varianten der

[2] Aus der Sicht von Animatoren sind Hierarchien für Interpolation wichtig. Aus der Sicht eines Riggers kommen Wartbarkeit, Geschwindigkeit und Vereinfachung hinzu.

Interpolation unterschieden. Drei davon sind in Abbildung 4.7 zu sehen. Jede Animati-

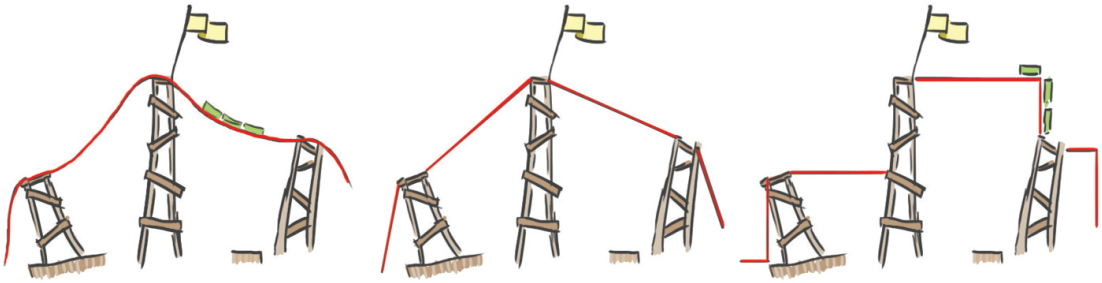


Abbildung 4.7: Interpolation beschreibt in der Animation, wie die Frames zwischen Keyframes berechnet werden. Dabei gibt es verschiedene Varianten, wie hier dargestellt. Die Kurve links ist im *Bezier Modus*. Dabei werden Kurven unter Berücksichtigung von Tangenten und den vorhergehenden Keys erzeugt. Diese Kurven werden oft für organische Animationen verwendet, da sie Beschleunigung und Verzögerung darstellen können. In der Mitte ist die lineare Interpolation abgebildet. Wie der Name richtig beschreibt, ist diese Berechnung eine simple, gewichtete Durchschnittsberechnung und erzeugt eine Gerade. Als letzte Variante ist rechts eine Kurve im *stepped Modus*. Dieser ist letztendlich keine richtige Interpolation. Der Wert wird gehalten, bis ein neues Keyframe einen neuen Wert setzt.

onssoftware besitzt die eine oder andere Variante, um Interpolation zu ermöglichen und zu manipulieren.[3] In Bereichen, in denen mit mehr Bildern pro Sekunde gearbeitet wird, wie zum Beispiel Computerspielen, nimmt Interpolation einen großen Teil der Arbeit ab. Generell kann als Faustregel gesagt werde, je höher die Framerate, desto wichtiger wird Interpolation. Um diese Aussage zu untermalen, folgt ein Extrembeispiel: So wie im Film 24 Hertz das Maß für flüssige Bewegung ist, liegt diese Zahl bei 60 Hertz in der Echtzeitbranche. Eine Erklärung für diesen großen Unterschied könnte im Fehlen der Bewegungsunschärfe liegen. Ist die Bewegung zu schnell, sind die Posen in zwei Bildern zu weit voneinander entfernt, um als eine Bewegung wahrgenommen werden zu können. Bewegungsunschärfe verbindet diese beiden Posen und macht die Bewegung wieder lesbar. Im Echtzeitbereich fehlt diese Bewegungsunschärfe meist, da sie rechenaufwendig ist. Um schnelle Bewegungen trotzdem lesbar zu halten, muss eine höhere Framerate erreicht werden. 24 Bilder pro Sekunde sind nicht ausreichend. 60 Bilder pro Sekunde sind einem Animator nicht zuzumuten, sollte er ohne Interpolation arbeiten. 60 Posen händisch zu erstellen wäre für kein Studio leistbar. Was jedoch möglich ist, sind einige wenige Posen zu erstellen. Diese werden mit Interpolation verbunden und, wenn die Animation schnell ist, Frame für Frame animiert. Interpolation ist also praktisch, wenn sich wenig Information in der Animation ändert. Diese Aussage wird in Abschnitt 4.3.3 genauer behandeln.

Als nächstes wird auf zwei unterschiedliche Arbeitsweisen eingegangen, wie Animationen erstellt werden können. Der große Unterschied zwischen diesen beiden Arbeitsweisen liegt im Umgang mit Interpolation oder genauer, wann von *stepped* auf *splined*

[3] Ausnahmen bilden hierbei zweidimensionale Pixel Programme, da Pixel keine Rotation oder Translation haben. Es ist reine Farbinformation ohne Bewegung.

umgestellt wird. Festzuhalten ist, dass es beim Erzeugen von Animationen wie bei so vielen Methoden im künstlerischen Bereich, kein richtig oder falsch gibt. Techniken etablieren sich, mehr oder weniger, nach persönlichen Vorlieben, Gewohnheiten oder Situationen. Trotzdem sollten einige Vorteile und vor allem Nachteile hervorgehoben werden, die gewisse Herangehensweisen an Animation mit sich bringen. In den folgenden beiden Abschnitten werden daher vermehrt eigene Meinungen vertreten, die von anderen Animatoren eventuell bestritten werden.

4.3.1 Animation in Splined

Bei dieser Methode wird von Anfang an mit glatter Interpolation, also Kurven, gearbeitet. Der Fokus liegt vielmehr auf Timing und Bewegung, da diese von Anfang an leichter lesbar sind. Keys werden dort gesetzt, wo Änderungen an den Bewegungen nötig sind. Durch diese Dynamik kann auch die Kreativität angeregt werden, da sich Posen zufällig ergeben können, an die nicht selber gedacht wurde. Diese können adaptiert und vertieft werden. Das Phänomen des *Happy Accident* tritt bei dieser Methode häufig auf. Zusätzlich ist die Animation von der Framerate unabhängiger. Durch die frühe Lesbarkeit des Timings kann viel schneller auf nötige Änderungen reagiert werden. Die Dynamik und Impulsivität gehen jedoch zu Lasten der Präzision. Animationen, die ausschließlich im Splined-Modus animiert werden, können schnell konfus und unleserlich wirken. Klare Impulse und Höhepunkte in der Animation sind schwerer herauszuarbeiten. *Schwammig* und *maschinell* sind Begriffe, die oft als Beschreibung dienen. Kurven sind sehr schnell lesbar im Splined modus, da die Animation von Anfang an aus Kurven besteht. Das verleitet zum baldigen Glätten der Kurven, was das vorher erwähnte maschinenhafte Verhalten hervorrufen kann.

4.3.2 Animation in Stepped

Auf der anderen Seite steht die Animation im Stepped Modus. Hier wird bis zum letzten Moment gewartet, um die Animation zu glätten. Stepped Animationen sind wesentlich posenorientierter. Bewegung kann lange nicht gelesen werden, da zuerst die für die Handlung wichtigsten, wenigen Posen gebaut werden. Diese, auch als Keyposen bezeichneten Posen, beschreiben die wichtigsten Akzente. Mithilfe eines Beispiels ist diese Technik leichter verständlich. Die Handlung des Shots ist sehr reduziert. Ein Charakter stemmt einen Stein vom Boden über seinen Kopf. Die wichtigsten Posen sind dabei das Stehen vor dem Stein, das Zupacken und zuletzt das Halten des Steins über dem Kopf. Diese drei Posen beschreiben die Handlung, weniger die Bewegung. Das Timing kann auch noch nicht festgelegt werden, sondern wird nur grob angepasst. Als nächstes kommen verschiedene Zwischenposen. Das Ausholen vor dem Zupacken, der erste Stemmversuch und der Kraftaufwand um den Stein von Hüfthöhe auf Brusthöhe zu bekommen. Zu sehen ist diese Animation in Abbildung 4.8. Jede Pose hat eine Bedeutung und fügt zusätzliche Information hinzu. Die Handlung wird genauer beschrieben, gleichzeitig wird die Bewegung leserlicher. Es werden immer mehr Posen hinzugefügt, bis jedes zweite, dritte, bei schnellen Bewegungen auch jedes erste Frame mit einer Pose versehen ist. Und erst dann wird auf Spline umgestellt. Da die Bewegung zu diesem Zeitpunkt schon gut lesbar sein sollte, sollte nicht mehr viel Zeit in die Polierarbeit fließen. Meist bleibt das Betonen der Akzente und des Gewichts über. Diese Methode hat natürlich auch

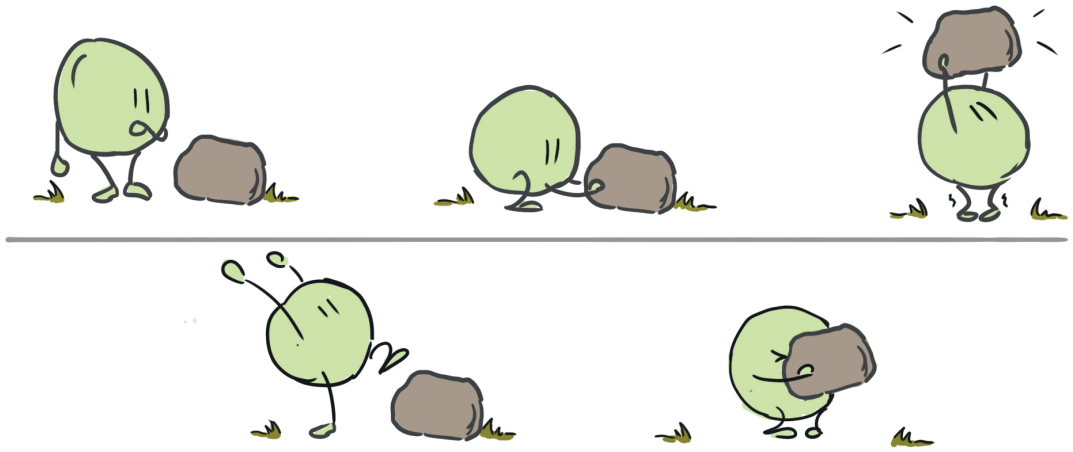


Abbildung 4.8: Die erste Zeile beschreibt die Keyposen. Es wird nur grob umrissen, worum es in der Animation geht. Der erste Durchlauf der Animation beschreibt nicht das Timing, oder zumindest nur sehr vage, sondern nur das Was, nicht das Wann. Im zweiten Durchlauf, der Zeile darunter, werden Zwischenposen hinzugefügt, sogenannte Inbetweens. Nun kann das Timing bereits besser angepasst werden, und es ist etwas mehr Information zur Handlung hinzu gekommen. Jeder weitere Durchlauf beschreibt immer weniger Handlung und immer weniger Bewegung, bis zusätzliche Posen keine zusätzliche Information mehr hinzufügen. Im Buch *Animator's Survival Kit* [7, S. 47] wird verstärkt auf das Thema Inbetweens eingegangen.

Nachteile. Dazu zählen die späte Lesbarkeit des Timings, aber vor allem auftretende Interpolationsfehler. Im Stepped Modus fallen große Rotationssprünge, wie zum Beispiel von 0 auf 360 Grad, nicht auf, nach der Interpolation jedoch schon. Sitzt die Hand in den einzelnen Posen am Stein, bedeutet das nicht, dass sie auch im Splined Modus fest am Felsen sitzt. Solche Fehler treten erst bei der Interpolation auf.

4.3.3 Interpolationslose Animation

Der Grund, warum auf diese beiden Techniken eingegangen wird, ist die Frage, ob überhaupt interpoliert werden muss. Traditionelle Animation mit Stift und Papier hatte auch nicht die Möglichkeit dazu, und es wurden trotzdem erfolgreiche Filme animiert. Interpolation ist also kein Muss, sondern eher eine Frage des Stils. Etablierte Studios wie Disney, Pixar, Dreamworks und Sony haben lange den Stil für kommerzielle Filme vorgegeben und dabei einen zueinander ähnlichen Stil entwickelt. Der Trend ging Richtung Realismus, mit physikalisch basierten Shadern und Renderern. Hinzu kam auch ein immer höherer Grad an Realismus in der Animation. Muskeln, Fett Haut, Stoff, Haare und viele weitere Elemente werden mit Hilfe von realistischen Simulationen berechnet. Durch die Annäherung an den Realfilm wurde natürlich auch die Framerate vom Realfilm übernommen. Was bedeutet, dass 24 Hertz auch im Computeranimationsfilm das Maß der Dinge geworden ist. Dabei hat Computeranimation die gleiche stilistische Flexibilität, die traditionelle Animation auch hat. Vor allem in den letzten Jahren wurden abstrahierte, kommerzielle Filme gewagt, die auf Interpolation teilweise verzichten. *Pea-*

nuts - Der Film [16] von Blue Sky Studios, die *Lego* [14] Filme von Animal Logic und der noch nicht veröffentlichte *Into the Spider Verse* [11] von Sony Picture Animation verzichten alle auf Interpolation und arbeiten mit variablen Framerates. Wie vorher erwähnt, ist Interpolation dann am effektivsten, wenn wenig Informationsänderung in der Bewegung vorhanden ist. Ab wann ist so wenig neue Information vorhanden, dass die Bewegung komplett weggelassen werden kann? Hat der Film einen gewissen Stil, der einen höheren Informationsverlust und damit eine reduziertere Animation zulässt? Fällt Interpolation weg, ist die Animation naturgemäß näher an der traditionellen Animation, wo nur animiert wird, wenn die Information benötigt wird. Bei zweidimensionaler Animation wirkt ein Charakter nicht leblos, wenn er mehrere Frames bewegungslos gezeigt wird, bei dreidimensionaler Animation jedoch schon. Wenn leichte Bewegungen, wie Moving Holds oder minimale Gewichtsverlagerungen gezeigt werden, reicht das meistens aus, um dem Charakter wieder Leben einzuhauchen, kompletter Stillstand funktioniert nicht. Vermutlich liegt das an der Nähe zum Realfilm, die die Animation über die Jahre erreicht hat. Ein Mensch, der sich nicht bewegt, wirkt tot. Entfernt man sich mit Reduktion, Stilisierung und Abstraktion jedoch von diesem, ist Stillstand wieder möglich. Oder hart ausgedrückt: Je reduzierter der Film ist, desto mehr verzeiht der Zuseher.

Raf Anzovin beschreibt in einem seiner Blogeinträge [29] zum Thema Ephemerale Rigs, siehe Kapitel 5, ebenfalls die Problematik von Interpolation. Er nähert sich jedoch von einer anderen Richtung, und sieht die Schwierigkeit eher in den Interpolationsfehlern. Wenn Gliedmaßen anderen Körperteilen folgen, in seinem Beispiel eine Hand dem Kopf, sind mit normalen Rigs komplexe Abhängigkeiten und Raumwechseln nötig, die ohne Interpolation wegfallen würden.

4.4 Zusammenfassung

In diesem Kapitel wurde die Problematik der komplexen Bewegung analysiert und welche Komponenten dazu beitragen. Bevor das nächste Kapitel beginnt, werden die wichtigsten Erkenntnisse noch einmal zusammen gefasst. Bewegungen sind dann komplex, wenn die Bewegung gegen die rig-interne Hierarchie läuft. Um nicht einen langsamen, dafür universellen Rig mitschleppen zu müssen, sind leichtere, temporäre Rigs von Vorteil, die auf gewisse Bewegungen spezialisiert sind. Um frei von der Limitierung von Hierarchien zu sein, muss Interpolation wegfallen. Um Interpolation ignorieren zu können, muss der Stil der Animation das zulassen. Der Aufwand, mit dem komplexe Bewegungen animiert werden können, hängt demnach vom Stil der Animation ab.

Abbildung 4.9: Einige Filmstudios wagen mit reduzierter Animation den Weg weg von interpolierter Animation. Einige Bespiele sind hier gezeigt. Von oben nach unten sind das *Into the Spider Verse, The Lego Movie, Peanuts - Der Film.* Bei den zwei Letztgenannten wurde auch auf realistische Hintergründe und realistisches Licht verzichtet. Die Filme bewegen sich wesentlich näher an den Comic-Vorlagen. Bei den Lego-Filmen wird mit übertriebenem, aber trotzdem realistischem Licht gearbeitet. Die Abstraktion ist in der Animation wesentlich stärker vertreten. Bildquelle [12], [15], [17].

Kapitel 5

Ephemerale Rigs

Die Idee hinter Ephemeralen Rigs wurde von Raf Anzovin entwickelt. Er beschreibt seine derzeitigen Ergebnisse in seinem Blog *Just to do something Bad!* [27]. Die folgenden Abschnitte basieren auf seiner Arbeit, werden aber durch Erkenntnissen aus Kapitel 4 erweitert.

5.1 Hierarchielose Rigs

Rigs müssen nicht in strengen Hierarchien aufgebaut werden. Die Idee, flachere Hierarchien zu verwenden, ist nicht neu geschweige denn weit hergeholt. Diese Form von Rigs wird auch als *Broken Skeleton* oder *Broken-Joint Skeleton* [35] bezeichnet, wobei diese Bezeichnung nicht gut gewählt ist. Der Begriff suggeriert ein kaputtes Verhalten, was nicht der Fall ist. Er beschreibt lediglich eine andere Form des Rigaufbaus. *Broken Skeletons* sind nicht hierarchielos, sondern fächern diese auf mehrere Geschwister auf. Die Gliedmaßen und der Torso können auf der selben Ebene liegen. Damit diese Teile des Rigs trotzdem ein korrektes Verhalten aufweisen, werden sie mit physischen Berechnungen verbunden.[1] Solche Verbindungen können beispielsweise die in Abschnitt 3.3 beschriebenen Constraints sein. Modulbasierte Rigs haben den Vorteil, austauschbar zu sein. Es können zum Beispiel mehrere Versionen eines Arms gebaut werden, und im finalen Charakterrig werden die am besten passenden Subrigs zu einem Masterrig zusammen gebaut. Der Videoblog *Cult of Rig* [32] von Rafael Fragapane beschäftigt sich im Detail mit dem Aufbau von modularen Rigs. Raf Anzovins Arbeit basiert zum Teil auf Ideen, die Rafael Fragapane in seinem Blog präsentiert, wenn auch in einem ganz anderen Kontext.

Broken Skeletons sind noch nicht ausreichend, um das Ursprungsproblem, die Vereinfachung komplexer Bewegungen, zu lösen. Ziel ist es, Hierarchien ganz zu entfernen. Aus technischer Sicht ist das kein Problem. Jeder Joint, der einen Knochen repräsentiert, kann ohne weiteres in der gleichen Ebene liegen. Wesentlich wichtiger als der

[1] Unter physischen Verbindungen versteht man Abhängigkeiten, die mit dem richtigen Editor in der Szene angezeigt werden können. In Autodesk Maya wären das beispielsweise Informationsabhängigkeiten, die mit dem Node Editor angezeigt werden können, oder hierarchische Abhängigkeiten, die vom Outliner angezeigt werden können. Logische Verbindungen können nicht angezeigt werden, da sie aus Programmaufrufen bestehen. Die Begriffe werden in Abschnitt 5.3.2 genauer beschrieben.

technische Aspekt ist die Änderung des Verhaltens. Fallen Hierarchien weg, sind Transformationen wie Rotation und Skalierung ohne Effekt. Diese Transformationen zeigen ihre Auswirkungen anhand ihrer Kinder. Die einzige Transformation, die nach wie vor von Bedeutung ist, ist die Translation, siehe Abbildung 5.1. Die Frage, um welchen

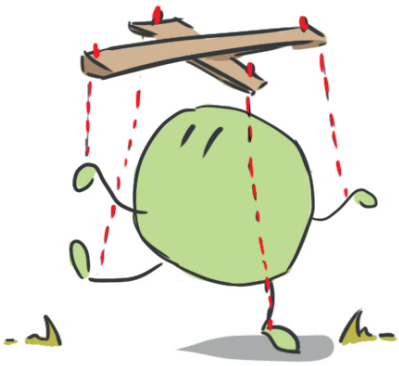


Abbildung 5.1: Die Manipulation des hierarchielosen Rigs ähnelt ein bisschen einem Puppenspiel. Bei beiden werden vom Animator beziehungsweise vom Puppenspieler nur Translationen vorgegeben. Diese Translationen geben dann die Rotationen vor, die sich daraus ergeben. Bis auf den Twist sind alle Rotationen von den Translationen abhängig.

Winkel der Ellbogen gedreht ist, steht nicht mehr im Vordergrund, viel wichtiger ist die Position, die die Hand zum Ellbogen hat. Bei einem hierarchielosen Rig geht es also um die Positionen der einzelnen Gelenke, und nicht um den Winkel dieser. Die Winkel können, falls sie benötigt werden, mit einfacher Trigonometrie berechnet werden. Drei Gelenke ergeben ein Dreieck, bei dem alle Längen bekannt sind, nachdem die Gelenke alle im selben Raum liegen. Sind drei Werte eines Dreiecks bekannt, können alle weiteren Werte berechnet werden. Die Idee von hierarchielosen Rigs ist es also, eine Pose durch die Position der Gelenke darzustellen. Für Animatoren ist diese Lösung zwar flexibel, da jeder Teil eines Rigs unabhängig voneinander bewegt werden kann, aber deswegen nicht effizienter. Dreht sich der Charakter in der Animation im Kreis, müssen trotzdem Rotationen entlang der Ausrichtung des Joints, also Twists, möglich sein. Eine Drehung nur mit Translationen in eine sauber Pose umzuwandeln, ist zusätzlich schwierig und zeitaufwändig. Der Vorteil von Hierarchien beim Posen ist also manchmal erwünscht, und manchmal störend. Um schnell und effizient zu arbeiten, wird ein Hilfsrig benötigt, der die Animatoren unterstützt, ohne sie einzuschränken.

5.2 Erkenntnisse und Verbesserungen des prozeduralen Ansatzes

Der vorhergehenden Abschnitt brachte die Erkenntnis, dass hierarchielose Rigs ohne ein unterstützendes Rig nur schwer animierbar sind. Die Idee von ephemeralen Rigs ist, dass Hierarchien dann aufgebaut werden, wenn sie Sinn machen, und gelöscht werden, wenn sie im Weg sind. Daher kommt auch der Name *Ephemeral*, was übersetzt *flüchtig, kurzlebig* bedeutet. Die Hilfsstrukturen sind temporär. Auf einer ähnlichen Idee baut auch das prozedurale Rig von Abschnitt 4.1 auf. Auch hier werden Rigs dann erstellt,

wenn sie benötigt werden, und ausgetauscht, wenn sie nicht optimal für die Animation sind. Der Grundstein für diese Technik bildet die Trennung zwischen Control, Motion und Deformation System, siehe Kapitel 1. Diese Trennung kann für das ephemerale Rig übernommen werden. Als nächster Schritt muss der Nachteil der prozeduralen Rigs, die Abhängigkeit von festen Hierarchien, umgangen werden. Da ephemerale Rigs in ihrer Natur bereits hierarchielos sind, tritt dieses Problem von Anfang an nicht auf. Der große Unterschied zwischen dem prozeduralen Rig von Richard Lico [33] und dem ephemeralen von Raf Anzovin besteht in der Lebensdauer der Rigs. Ersterer baut seine Rigs dann auf, wenn er sie benötigt, und tauscht sie aus, sobald sie nicht mehr effizient sind. Der springende Punkt und der Grund, warum ephemerale Rigs so viel flexibler sind, liegt darin, dass bei jeder einzelnen Manipulation ein neues Rig erstellt wird. Sobald ein Controller angegriffen wird, wird im Hintergrund eine neue, logische[2] Hierarchie aufgebaut. Ein weiterer Unterschied ist, dass die Hierarchien beim ephemeralen Ansatz logisch sind, im prozeduralen physisch. Dank der logischen Natur ist dieser Ansatz viel flexibler. Jeder Controller kann als Elterncontroller für eine variable Anzahl an Kindercontrollern fungieren. Da jeder Controller ein Geschwister zu den anderen Controllern ist, sind logische Abhängigkeiten auch leicht zu implementieren.

Bei prozeduralen Rigs wird die Animationsinformation in den Joints gespeichert und nicht im Rig. Das macht die Rigs austauschbar. Nachdem ephemerale Rigs auf einem ähnlichen Prinzip aufbauen, macht es Sinn, die Information ebenfalls im Deformation System zu speichern.

5.3 Technische Ansätze

Der interessanteste Teil von ephemeralen Rigs ist die Manipulation des Rigs und wie Animatoren dabei unterstützt werden können, ohne ihnen Freiheiten wegzunehmen.

5.3.1 Informationsfluss

Zu allererst sollte jedoch der Informationsfluss beschrieben werden. Wie in dem vorhergehenden Abschnitt erwähnt, wird die Information nicht in den Controllern gespeichert, sondern in den Joints, um die Austauschbarkeit zu gewährleisten. Raf Anzovin beschreibt seine Technik in einem seiner Blogeinträge [28]. Dabei wird eine logische Verbindung zwischen dem Joint und dem Controller aufgebaut, sobald der Controller manipuliert wird. Die Verbindung ist so lose wie möglich. In Maya kann das mit einem Python- oder MEL-Befehl realisiert werden, der die Attribute am Joint setzt. Eine direkte Verbindung wäre zu stabil und nicht flexibel genug. Das Rig muss so temporär wie möglich gehalten werden. Wird ein Joint, der Keyframes besitzt, via Skript manipuliert, so wird automatisch ein weiteres Keyframe hinzugefügt. Animatoren müssen sich nicht darum kümmern, dass Keys auf den Joints platziert werden, die Manipulation des Controllers ist ausreichend. Das System funktioniert auch, wenn der Controller von jemand anderem als den Animatoren manipuliert wird, wie zum Beispiel einem anderen Skript.

Das Problem bei diesem System ist, dass es beim Abspielen der Animation nicht funktioniert. Die Information wird in den Joints gespeichert und nicht in den Controllern selber, wie in herkömmlichen Rigs. Die Controller bewegen sich also nicht mit den

[2] Der Begriff *logisch* wird in Abschnitt 5.3.2 erklärt.

Joints mit, wenn die Animation abgespielt wird. Sind die Joints und die Controller nicht mehr synchron, wird das Animieren erschwert. Eine der Anforderungen an Rigs, die in Kapitel 1 beschrieben werden, ist Vorhersehbarkeit. Ein Animator muss immer wissen, was ein Controller verändert und wie er es verändert, wenn er manipuliert wird. Das ist eine Mindestanforderung und muss gewährleistet werden. Die Anforderung kann dennoch erfüllt werden. Es ist möglich, festzustellen, wann das Abspielen der Animation beendet wird. Sobald die Animation stoppt, können die Controller an die Stelle der Joints verschoben werden. Wie vorhin erwähnt, werden Joints von den Controllern beschrieben, sobald diese manipuliert werden. Beschreiben jetzt die Joints die Controller, so werden eben jene Joints wieder von den Controllern beschrieben. Prinzipiell ist es egal, da die Controller die Joints mit denselben Werten beschreiben, die die Joints ohnehin bereits besitzen. Unnötige Berechnungszeit ist es dennoch. Um diese zusätzliche Berechnung zu unterbinden, kann ein Rig sich in zwei verschiedenen Modi befinden. Raf Anzovin nennt diesen beiden Modi *Interact* und *Playback* [25]. Befindet sich das Rig im Playback Modus und Playback wird gestoppt, werden alle Controller an den Joints angeglichen, und erst danach der Modus von *Playback* auf *Interact* gestellt. Controller können ihrerseits nur Joints beschreiben, wenn sich das Rig im *Interact* Modus befindet. Die Wichtigkeit dieser beiden Modi wird im nächsten Abschnitt genauer behandelt.

5.3.2 Logische Hierarchien

Bisher wurde der Begriff logische und physische Hierarchie des Öfteren verwendet. In diesem Abschnitt geht es nun direkt um den Aufbau solcher, deswegen ist die Trennung und das Verständnis dieser beiden Begriffe nun um so wichtiger. Es gibt keine offizielle Bezeichnung für diese beiden Kategorien, weswegen diese beiden Begriffe hier für diese Arbeit definiert werden.

Physische Entitäten

Was meist unter einer Hierarchie verstanden wird, ist eine physische Hierarchie. Objekte, die in einer Szene in einer Hierarchie liegen, sind physisch. Sie existieren real.[3] Ein Beispiel dafür ist in Maya der Outliner. Physikalische Verbindungen sind Verbindungen, die zum Beispiel mit dem Connection Editor gemacht werden; direkte Verbindungen zwischen Objekten, die Information weiterleiten. Physikalische Entitäten sind stabil, aber unflexibel. Sie können vom Animator nicht einfach gelöscht, umgruppiert oder neu verbunden werden, da dies das Verhalten des Rigs ändern könnte. Auch können sie während des Abspielen der Animation nicht sauber erstellt oder gelöscht werden. Physikalische Entitäten werden von der Arbeitsszene beschrieben.

Logische Entitäten

Logische Entitäten manipulieren physische Entitäten über flüchtige Skripte oder Programme. Es sind keine direkten, hart verkabelten Informationsflüsse. Stattdessen verhalten sich logische Verbindungen eher wie ein Animator, der automatisiert Manipulationen tätigt. Logische Verbindungen sind nie in einer Szene zu finden, da sie sich nicht

[3] Eine Szene wird als real gesehen. Objekte, die sich in dieser Szene befinden, sind reale Objekte.

in irgendwelchen Verbindungen oder Objekten manifestieren, sondern rein Werte manipulieren. Wird ein Objekt via Skript an die Position eines anderen Objekts bewegt, so ist das eine logische Verbindung. Es existiert nur für die Dauer des Befehls, manipuliert die Position des ersten Objektes und ist anschließend nicht mehr existent. Animatoren könnten diesen Schritt auch händisch machen, das Skript automatisiert diese Tätigkeit. Logische Entitäten werden über Skripte beschrieben.

Aufbau einer logischen Hierarchie

Der Vorteil von logischen Hierarchien liegt in der Flexibilität. Die Verbindungen sind flüchtig und können jederzeit anders aufgebaut werden, was letztendlich der Grundgedanke ist, um komplexe Bewegungen einfach animierbar zu machen. In Kapitel 2 wird der Aufbau von Hierarchien mit Matrizen beschrieben. Dieselbe Technik wird verwendet, um logische Hierarchien aufzubauen, nur in einem Skript und nicht in der Szene selber. Um eine Hierarchie aufzubauen, müssen logische Kinder in den Raum der logischen Eltern überführt werden. Dazu wird die Weltmatrix des logischen Kindes mit der inversen Weltmatrix des logischen Elter multipliziert. Die daraus resultierende Matrix beschreibt die Position, Rotation und Skalierung des logischen Kindes, wäre es im logischen Elternraum. Diese Matrix kann auch als logische lokale Matrix bezeichnet werden. Wird diese Matrix mit der Änderung der Weltmatrix des Elternobjektes multipliziert, so verhält sich das logische Kind, als wäre es ein physisches Kind, ohne dabei in einer physischen Hierarchie zu sein. In Abbildung 5.2 wird die Berechnung einer physischen Variante gezeigt. Durch die physischen Verbindungen ist es in diesem Beispiel nicht mehr möglich, das Kindelement unabhängig zu bewegen. Es wird fix vom Netzwerk gesteuert. Deswegen ist der physische Ansatz nicht der richtige. Eine logische Variante würde von den Berechnungen komplett identisch ablaufen, nur als Skript und nicht als Netzwerk. Skripte haben zudem den Vorteil, in ihrer Komplexität variabel zu sein. Es können beliebig viele Kindelemente manipuliert werden, ohne dass das Skript geändert werden muss. Die Kindelemente müssen allerdings gefunden werden, bevor sie manipuliert werden können.

5.3.3 Traversieren im Rig

Auch wenn die Idee eines ephemeralen Rigs auf der kompletten Unabhängigkeit der einzelnen Gelenke zueinander basiert, so ist die komplette Unabhängigkeit für automatisiertes Verhalten nicht förderlich. Diese Verbindungen dürfen natürlich nicht das Verhalten der Controller direkt beeinflussen, da das die Aufgabe der Animatoren oder der von Animatoren ausgelösten Skripte ist. Die Verbindungen sollten also eher eine Metaebene bilden, die nur Zusammengehörigkeiten beschreibt, aber keine Informationen austauscht. Der Ellbogen muss nur wissen, dass er mit der Schulter und mit der Hand verbunden ist. Als Ausgangsbasis reicht diese Information, wie in Abbildung 5.3 gezeigt. Abbildung 5.4 zeigt dieses System in Maya mit Message Attributen realisiert. Mit diesen Verbindungen können alle abhängigen Controller gefunden werden. Manipuliert man einen Controller im Vorwärtskinematik Modus, so werden die dazugehörenden Controller über das Forward-Attribut gefunden und können ebenfalls manipuliert werden. Abbildung 5.5 zeigt diesen Vorgang. Anzovin beschreibt in seinem Blogeintrag [26] zu diesem Thema zudem die Schwierigkeiten, die auftreten, wenn ein Controller mit

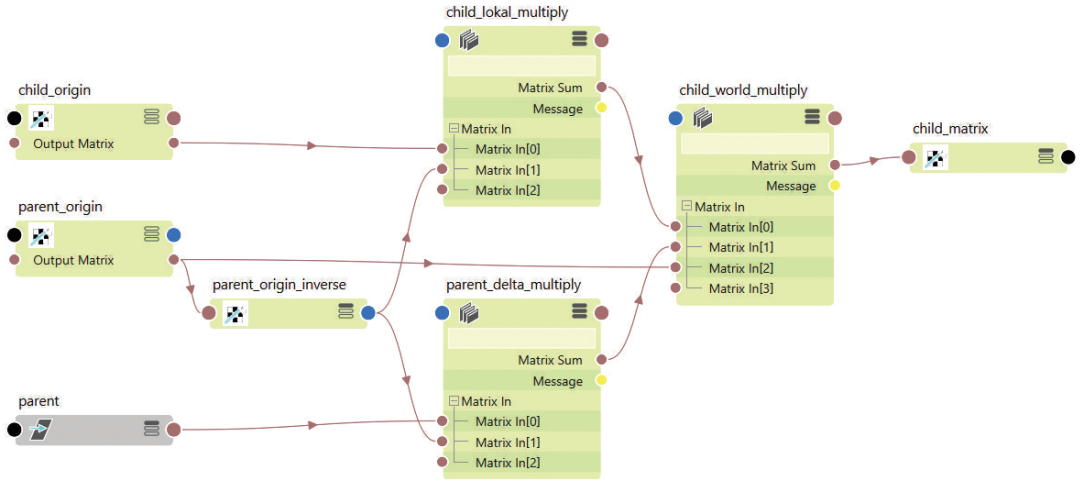


Abbildung 5.2: Die Nodes *child_origin* und *parent_origin* speichern die Matrizen der Objekte zum Startzeitpunkt der Manipulation. *child_lokal_multiply* berechnet die lokale Matrix von *child* gesehen von *parent* vor der Manipulation. *parent_delta_multiply* berechnet die Änderung der Matrix, die *parent* während der Manipulation erzeugt. Werden diese beiden Matrizen miteinander multipliziert, so erhält man die Manipulation, die auf *child* ausgeführt wird, allerdings vom Nullpunkt der Szene ausgehend. Wird die ursprüngliche Matrix von *parent*, *parent_origin*, ebenfalls hinzu multipliziert, so erhält man ein korrektes hierarchisches Verhalten von *child*. Hervorzuheben ist zudem, dass nur *child_world_multiply* und *parent_delta_multiply* während der Manipulation berechnet werden müssen, da sie mit sich ändernden Nodes verbunden sind. Alle anderen Nodes müssen nur einmal berechnet werden, und werden für die Dauer der Manipulation ignoriert.

einem anderen Controller verbunden ist. Das Rig sollte die Möglichkeit besitzen, zwei Controller miteinander zu verbinden. Das Beispiel, wenn sich der Charakter an den Kopf greift, ist schon einige Male erwähnt worden, und beschreibt jene Situation. Die Hand und der Kopf sind nicht direkt miteinander über das Beschreibungssystem verbunden, trotzdem soll die Hand mit dem Kopf mit bewegt werden. Es ist nicht schwierig, die Hand mit dem Kopf mit zu bewegen. Das kann über eine logische Hierarchie gelöst werden. Schwieriger ist vielmehr zu erkennen, dass die Hand nicht von Ellbogen oder Schulter bewegt werden darf. Es darf also nur solange traversiert werden, bis das Ende der Kette erreicht wurde, oder ein Controller von einem anderen gesperrt ist. In Anzovins Blog sind Codebeispiele angeführt, wie dieses Problem direkt mit Python gelöst wird [26].

Zusammenfassung der technischen Funktionalität

Die Grundtechniken, die einen ephemeralen Rig ermöglichen, sind in den Abschnitten oberhalb beschrieben. Die wichtigsten Eigenschaften des Rigs liegen in der Trennung zwischen Control System und Deformation System, dem Speichern der Animation im Deformation System, dem hierarchielosen Aufbau, dem Beschreibungssystem, den daraus erstellbaren, logischen Hierarchien und dem hierarchisch vererbbaren Verhalten der Controller. Ephemerale Rigs sind schneller als herkömmliche Rigs. Das Deformation

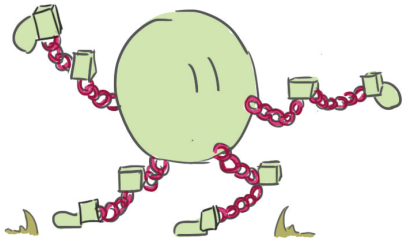


Abbildung 5.3: Die einzelnen Verbindungen zwischen den Controllern geben eine Beschreibung des Rigs. Es ist nur wichtig zu wissen welcher Controller mit welchem verbunden ist. Zum Beispiel ist der Ellbogen mit dem Arm, aber auch mit der Hand verbunden. Es wird nicht beschrieben, in welcher Abhängigkeit die Controller zu einander stehen, nur dass sie in einer stehen. Alle Verbindungen zusammen geben eine Beschreibung des Aufbaus des Rigs ab. Diese Beschreibung kann verwendet werden, um alle Controller zu finden, die bei einem vorwärtskinematischen Verhalten manipuliert werden.

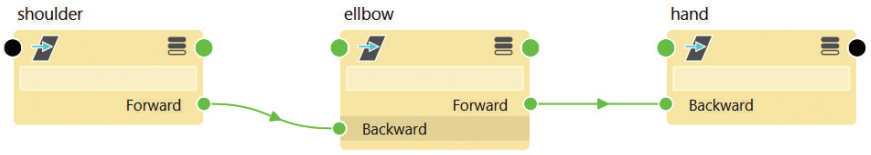


Abbildung 5.4: In Maya kann ein Beschreibungssystem mit Messages realisiert werden. Messages leiten keine Information weiter und dienen nur zum Abbilden von Zusammenhängen. Diese Verbindungen sind physisch, da sie nicht verändert werden sollen und rigspezifisch sind.

System weist die gleiche Komplexität auf, da die gleichen Deformationen erzielt werden müssen. Hier ist kein nennenswerter Geschwindigkeitsverlust oder -gewinn möglich. Das Controll System wird während dem Playback nicht aktualisiert. Es existiert zu diesem Zeitpunkt nicht, zumindest nicht als System. Die Controller sind zwar vorhanden, tragen aber nicht zur Animation bei. Es gibt keine Keys, die die Controller beeinflussen, deswegen müssen diese nicht evaluiert werden. Damit einhergehend existiert kein Motion System, das berechnet werden muss. Die Animation ist bereits im Deformation System fertig gespeichert und zusätzliche Berechnungen sind nicht nötig. Dank dieser Vereinfachung sind ephemerale Rigs immer schneller als herkömmliche Rigs. Diese Erkenntnis wird zwar in keinem der Blogs erwähnt, ist aber eine logische Schlussfolgerung, die sich aus der Vereinfachung des Rigs ergibt.

5.4 Animationsverhalten

Die Unterschiede in der Technik sind nun geklärt, was zu der nächsten Frage führt: Wie unterscheidet sich ein ephemeraler von einem hierarchischen Rig während der Animationsarbeit? Durch die Unterschiede im Aufbau ergeben sich auch Unterschiede in der Herangehensweise.

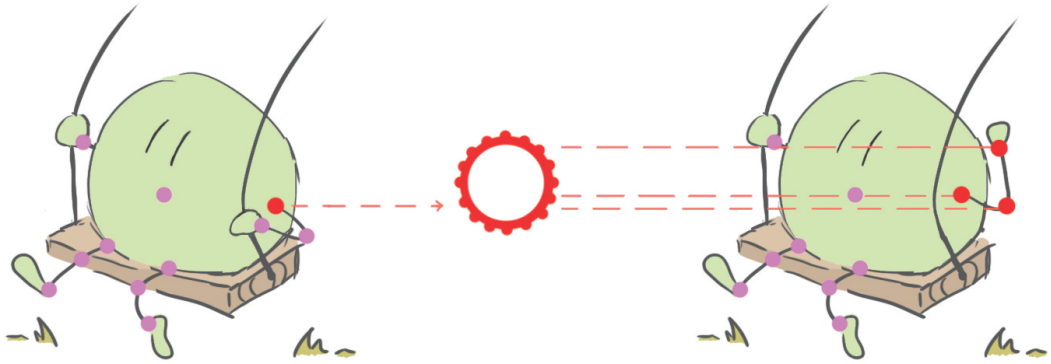


Abbildung 5.5: Der Arm des Charakters wird vom Animator rotiert. Die Rotation löst das Aufbauen einer logischen Hierarchie aus. Dadurch kann erkannt werden, welche Controller ebenfalls bewegt werden müssen, um ein vorwärtskinematisches Verhalten zu erzeugen. Diese Bewegung ist nicht interpolierbar, da die Hierarchie logischer Natur ist. Controller, die nicht von den Manipulationen durch die Animatoren betroffen sind, werden ignoriert.

5.4.1 Animation mit hierarchischen Rigs

Hierarchische Rigs sind interpolierbar. Die Interpolation muss daher auch beim Erstellen der Posen berücksichtigt werden. Rotationen bauen in den meisten Animationsprogrammen auf Euler-Rotationen auf. Diese Rotationen werden durch eine x-, y- und eine z-Achse beschrieben. Zusätzlich kommt noch eine Rotationshierarchie hinzu, wie diese Rotationen aufeinander einwirken [4, S. 223]. Abbildung 5.6 zeigt eine solche Rotationshierarchie, bei der die y-Achse dominanter ist als die x-Achse. Diese Hierarchien können zu schwer interpretierbaren Rotationen führen, die erst erkannt werden, wenn in den Splined Modus gewechselt wird. Für Animatoren sind Gimbal Locks eine Behinderung. Werden sie zu spät erkannt, müssen die Rotationen händisch ausgebessert werden. Funktionieren Animationen im Stepped Modus, bedeutet das nicht, dass das auch im Splined Modus der Fall ist, siehe Abbildung 5.7. Auch wenn Interpolation für Schwierigkeiten sorgen kann, bei mehr Bildern pro Sekunde ist sie extrem zeitsparend. Dadurch, dass nicht jedes Frame von Hand bearbeitet werden muss, können sich Animatoren auf die wichtigeren Aspekte konzentrieren, während Nebensächlichkeiten automatisiert sind. Durch Interpolation werden Bewegungen abgebildet, und nicht Posen aneinander gereiht, die wie eine Bewegung aussehen. Kurven müssen funktionieren, um eine funktionierende Animation abzubilden. Der Fokus liegt viel mehr auf Bewegung. Die Animation kann zeitlich gestaucht und gezerrt werden, ohne Informationsverlust[4] oder die Lesbarkeit zu riskieren. Auch Simulationen können mit diesen Animationen betrieben werden, da sie verlangsamt oder beschleunigt werden können. Realistischere Animationen basieren auf Hierarchien, da sie näher an der Realität sind und von sich aus natürlicher wirken. Die vererbbare Rotation wurde in Kapitel 4 bereits erwähnt

[4]Hier ist der Informationsverlust innerhalb der Animation gemeint. wird die Animation exportiert, und es befinden sich Keys auf Zwischenframes, kommt es sehr wohl zu Informationsverlust.

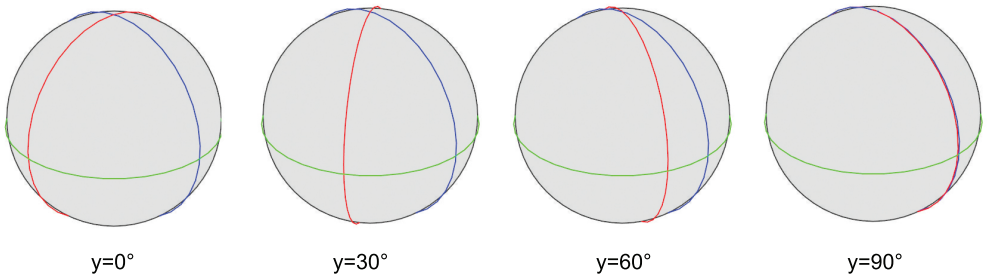


Abbildung 5.6: Die Rotationsordnung bei diesem Beispiel lautet xyz, was bedeutet, dass die dominanteste Achse die blaue z-Achse ist. Wird diese bewegt, rotieren alle Achsen mit. Rotiert man die grüne y-Achse, so bleibt die z-Achse unbeeinflusst. Die rote x-Achse beeinflusst nur sich selbst. Wird die y-Aachse an 90° angenähert, so beschreiben die x- und die z-Achse immer ähnlichere Rotationen, bis sie identisch sind. Diese Situation wird Gimbal Lock genannt und bezeichnet den Verlust einer Rotationsachse. Wenn die Kugel trotz Gimbal Lock entlang der ursprünglichen x-Achse gedreht wird, führt das zu sprunghaft an- und absteigenden Rotationswerten, die für den Animator nicht vorhersehbar sind. Durch diese stark wechselnden Rotationswerte ist eine saubere Interpolation nicht möglich.

Abbildung 5.7: Auch wenn die Posen (links und rechts) richtig aussehen, ist das kein Garant für sauber Interpolation. Die Pose sagt nichts über die Rotationsrichtung aus. In diesem Beispiel wird der Fuß von etwa -10 Grad auf 30 Grad gedreht. Es könnten aber auch -350 Grad auf 30 oder -330 sein. Die Posen sehen deswegen nicht anders aus, die Interpolation aber schon.

und beschreibt die Vorwärtskinematik, die nur in hierarchischen Rigs ohne Mehraufwand möglich ist. Radiale Bewegungen, die einen Großteil von Charakterbewegungen ausmachen, sind so leicht zu posen.

Die Vererbbarkeit ist sowohl ein Vorteil, kann aber auch ein Nachteil sein. Der Nachteil kann am besten mit einem Beispiel gezeigt werden. Ein Charakter greift nach einem Ball, der vorbei fliegt. Es gibt nur eine kurze Berührung, deswegen wird der Arm in Vorwärtskinematik animiert. Nachdem die Pose fertig ist, und die Finger an den Ball angepasst wurden, wird eine zweite Meinung zur Pose eingeholt. Diese lautet,

dass die Silhouette nicht so lesbar ist, wie sie sein könnte. Der Ellbogen sollte etwas weiter vom Körper weg sein. Um den Ellbogen zu bewegen, muss der Arm oder auch die Schulter angepasst werden. Wird der Ellbogen an eine andere Position verschoben, sitzen die Finger nicht mehr am Ball. Das bedeutet dass die gesamte Pose des Arms neu gemacht werden muss. Die hierarchische Manipulation beeinflusst alle Kindercontroller, und wird eine Änderung innerhalb der Hierarchie vorgenommen, müssen alle abhängigen Controller korrigiert werden. Je weiter oben der Controller sitzt, desto mehr Aufwand bedeutet das Überarbeiten.

5.4.2 Vor- und Nachteile bei der Animation

Der wichtigste Unterschied gleich vorne weg: Nicht jeder Animationsstil kann mit einem ephemeralen Rig effizient animiert werden. In Kapitel 4 wird auf die Überlegungen eingegangen, die genau diese Limitierungen hervorrufen. Ephemerale Rigs sind für stilisierte oder cartoonhafte Animationen optimiert, die keine perfekt flüssigen Bewegungen brauchen. Der Begriff *flüssig* bezieht sich hier auf die Bildfrequenz. Das Animieren mit diesen Rigs ist der traditionellen Animation nicht unähnlich. Der Fokus liegt ganz klar auf den Posen, da keine Bewegungen erzeugt werden können. Wird die Animation zeitlich gestreckt, können Bewegungen nicht mehr gelesen werden, da zu wenig Information vorhanden ist. Die Posen bleiben intakt, die Bewegung geht jedoch verloren. Zwischenposen sind nötig, um wieder lesbare Bewegungen zu erzeugen. Da keine Interpolation vorhanden ist, gibt es auch keine Probleme mit unvorhersehbaren Rotationen durch den Gimbal Lock. Die Controller müssen nicht dieselbe Rotation wie die Joints aufweisen, sondern können beim Selektieren immer eine Rotation von 0 haben, da die Änderung des Controllers wichtig ist und nicht die Rotation selbst, womit Gimbal Locks wesentlich seltener auftreten. Wenn sie auftreten, sind sie zu ignorieren. Die Werte der Rotation sind irrelevant, nur die Rotation selbst und die damit erzeugte Pose sind wichtig. Ohne Rücksicht auf Rotationen arbeiten zu können beschleunigt das Animieren ungemein, da ein ganzer Gedankenprozess wegfällt.

Einer der Nachteile der hierarchischen Rigs ist der Aufwand, der beim Überarbeiten der Posen auftritt. Das Selbe Beispiel beim ephemeralen Rig zeigt, dass sie wesentlich flexibler sind. Da alle Controller unabhängig voneinander sind, kann der Ellbogen überall positioniert werden, ohne dass die anderen Controller beeinflusst werden. Was bei hierarchischen Rigs auf eine Überarbeitung der Pose hinausläuft, ist beim ephemeralen Ansatz die Manipulation eines einzelnen Controllers. Die Controller können immer überall im Raum platziert werden, was den Animatoren Flexibilität und vor allem Freiheit beim Bau einer Pose gewährt, wie in Abbildung 5.8 gezeigt.

Soll ein hierarchisches Verhalten auftreten, kann es aktiviert werden. Sollen die Controller unabhängig sein, aktiviert man die Hierarchien nicht. Die Hierarchien können vorwärts, also Schulter, Ellbogen, Hand, oder rückwärts, Hand, Ellbogen, Schulter, aufgebaut werden. Das macht komplexe Bewegungen leichter. Der gesamte Rig kann über die Hand rotiert werden, in der nächsten Pose über den Knöchel. Die Skripte funktionieren, egal von welchem Controller aus gestartet wird.

Auch wenn diese Rigs wenige Limitierungen haben, so kann dennoch nicht jede Pose willkürlich gebaut werden. Jeder Teil des Rigs kann gestreckt und gestaucht werden, was dazu führt, dass der Charakter im Laufe einer Animation seine Proportionen verliert,

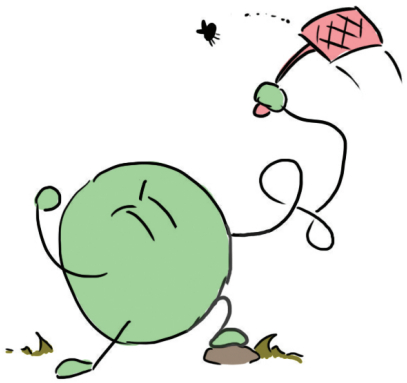


Abbildung 5.8: Da sich die Controller im Worldspace befinden und zueinander unabhängig sind, kann das Rig nach Belieben verzerrt werden, solange genug Controller vorhanden sind, um die Pose zu erstellen. Gelenke können jederzeit verschoben werden, ohne die anderen Controller zu beeinflussen. Sollen andere Controller mit beeinflusst werden, kann die Funktion aktiviert werden.

wenn nicht darauf geachtet wird. Einige Zusatzprogramme sind nötig, um die Form beizubehalten und generell Arbeiten mit dieser Form von Rigs zu erleichtern.

5.4.3 Nützliche Hilfsprogramme für ephemerale Rigs

Dieser Abschnitt beschäftigt sich nicht mehr direkt mit ephemeralen Rigs. Es werden einige nützliche Programme vorgestellt, die das Arbeiten mit ephemeralen Rigs erleichtern.

Onion Skin Renderer

Die Idee von Onion Skin Renderer kommt, wie so viele Ideen, aus der traditionellen Animation. Hier wird mit Lichttischen gearbeitet, um das Frame davor und danach abzulichten, während gezeichnet wird. Der Name kommt von dem Papier, das verwendet wurde, das, ähnlich wie die Schale einer Zwiebel, lichtdurchlässig ist. Wenn die umliegenden Posen sichtbar sind, ist es leichter, die aktuelle Pose zu zeichnen. Ohne Lichttische wäre Animation zeitaufwendiger, da die umliegenden Posen im Gedächtnis gehalten werden müssen. Die meisten zweidimensionalen Animationsprogramme[5] besitzen ebenfalls Onion Skin Renderer. Für Programme, die im dreidimensionale Raum arbeiten, wie *Autodesk Maya*, gibt es oft keine hauseigenen Onion Skin Renderer. Es gibt aber Plugins, die von der Community geschrieben werden, wie in Abbildung 5.9 gezeigt. Ein Vertreter der Onion Skin Renderer ist zum Beispiel von Christoph Lendenfeld [21].

Warum sind Onion Skin Renderer gerade für ephemerale Rigs so wichtig? Bei herkömmlichen Rigs können Keys entfernt werden, um eine glatte Bewegung zu erzeugen. Die Interpolation übernimmt das zu einem gewissen Grad für die Animatoren. Diese müssen die Pose noch verfeinern, aber die ungefähre Ausgangsposition und das Timing

[5] Einige Beispiele für zweidimensionale Animationsprogramme mit integriertem Onion Skin Renderer sind Toon Boom Harmony [22], Adobe Animate [18] oder TVPaint [23].

Abbildung 5.9: Ein Onion Skin Renderer mit einem ephemeralen Rig von Raf Anzovin. Die grünen und roten Posen beschreiben das davor und danach liegende Frame. Mit diesen beiden Posen kann die Pose dazwischen wesentlich leichter gebaut werden. Die Bewegung des Schwanzes kann ohne diese Funktion nicht sauber angeglichen werden. Bildquelle [10].

können durch die Interpolation errechnet werden. Ephemerale Rigs haben diesen Vorteil nicht. Werden Keys von Controllern gelöscht, bewegt sich der Controller für dieses Frame nicht mehr. Deswegen sind Onion Skin Renderer hier westlich wichtiger, da für jedes Frame jeder Controller von Hand gesetzt werden muss und keine Bewegung automatisiert werden kann.

Tweens

Unter den Sammelbegriff *Tweens* werden hier alle Programme zusammengefasst, die die namensgebenden Inbetweens erzeugen. Die Idee, auf der alle Tweens aufbauen, ist letztendlich sehr simpel. Wird zwischen zwei bereits existierenden Keyframes ein weiterer Key mithilfe eines Tweens erzeugt, so sind die Werte des Keys der gewichtete Durchschnitt der beiden angrenzenden. Abbildung 5.10 zeigt die Erzeugung eines getweenten Keys. Ein verbreiteter Vertreter unter den Tweens ist das Plugin *TweenMachine* [24] von Justin Barrett. TweenMachine baut auf demselben Prinzip des gewichteten Durchschnitts auf. Mit TweenMachine ist es auch möglich, Werte jenseits der 100% Grenze einzugeben, was für Overshoots[6] nützlich sein kann. Tweens sind Interpolationen nicht unähnlich, schließlich ahmen sie bei halber Änderung genau das Verhalten einer linear interpolierten Kurve nach. Ephemerale Rigs sind jedoch nicht brauchbar interpolierbar. Die Frage, warum Tweens für ephemerale Rigs wichtig sind, ist nicht unberechtigt. Die Nützlichkeit von Tweens liegt nicht in der automatischen Erstellung von Posen. Tweens dienen der schnellen Erstellung von Ausgangsposen. Die Animatoren müssen trotzdem jede Pose händisch verfeinern, aber als erster Schritt für eine neue Pose sind Tweens optimal, nicht nur für ephemerale Rigs, sondern für alle Rigs. Ephemerale Rigs haben jedoch den Nachteil, dass externe Plugins nicht funktionieren. Die meisten Plugins arbeiten mit selektierten Controllern, die Keys besitzen. Da die Keys jedoch nicht auf den

[6] Overshoots sind ein Begriff aus der Animation. Die Bewegung schießt über das Ziel hinaus und federt wieder zurück in die angepeilte Position.

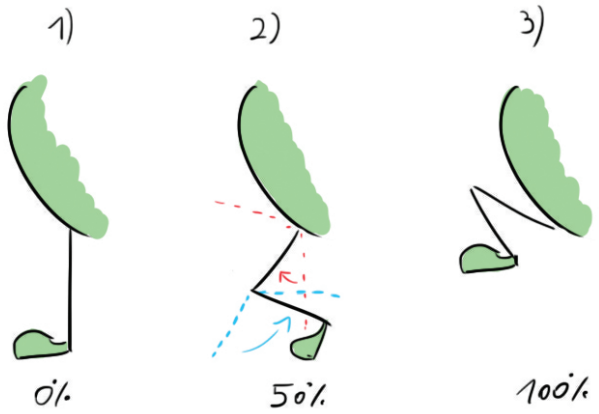


Abbildung 5.10: Die Keys auf Frame eins und drei existieren bereits. Mithilfe eines Tweens wird der Key auf Frame zwei erzeugt. Die Transformation der Gelenke entspricht in diesem Beispiel der halben Änderung der Rotation. Die Gewichtung kann natürlich frei gewählt werden.

Controllern, sondern auf den Joints liegen, muss ein eigenes Plugin dafür geschrieben werden.

Picker

Wie bei Tweens gibt es auch bei Pickern mehrere Plugins, die ähnliche Funktionen aufweisen. Die Grundfunktion eines Pickers besteht in der Erleichterung der Selektierung von Controllern. Die Controller werden abstrahiert in einem eigenen Fenster angezeigt und sind so leichter zugänglich. Abbildung 5.11 zeigt das Plugin *AnimSchoolPicker* [19]. Picker sind für ephemerale Rigs besonders interessant, da nicht nur Controller über die Maske des Pickers selektiert werden können, sondern auch Skripte aufgerufen werden können. Ephemerale Rigs bauen hauptsächlich auf logischen Verbindungen auf, weswegen ein Teil der Logik, wie zum Beispiel das Aufrufen von Funktionen, in den Picker ausgelagert werden kann. In Abbildung 5.12 ist der Picker von Anzovin zu sehen.

5.5 Zusammenfassung

Die physische Komponente von ephemeralen Rigs ist traditionellen Rigs sehr ähnlich und nicht wirklich besonders. Sie baut auf denselben Techniken auf wie joint-gesteuertes Skinning oder andere Deformer, die die Geometrie manipulieren. Auch das Verwenden einer flachen Hierarchie ist an sich noch kein großer Unterschied, da modulare Rigs ebenfalls auf diesem Ansatz aufbauen, siehe Abschnitt 5.1. Was ephemerale Rigs hervorhebt, ist zum einen der Grad der Auflösung von Hierarchien. Im Gegensatz zu modularen Rigs liegen nicht nur die einzelnen Module hierarchisch nebeneinander, sondern jeder einzelne Joint im Rig. Zum anderen ist das Wegfallen der physischen Motion Layer einzigartig. Das Rig besteht nur aus Control Layer und Deformation Layer. Die beiden Schichten haben keine physische Verbindung, auf der Information übertragen wird. Das macht

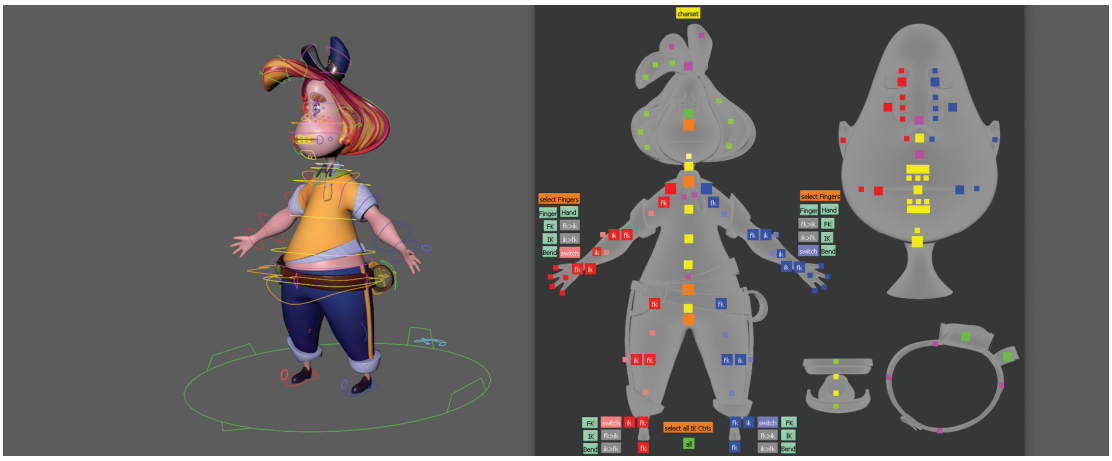


Abbildung 5.11: Der Charakter *Rosal* aus dem Kurzfilm *Ein Paket, das keinen Namen trägt* [8] kann komplett über den Picker auf der rechten Seite selektiert werden. Controller sind oft nicht erreichbar in der Animationsszene, da sie in anderer Geometrie stecken und somit nicht sichtbar sind. Über den Picker sind sie jedoch immer erreichbar. Zusätzlich können Skripte über den Picker aufgerufen werden, was die Bedienbarkeit weiter vereinfacht und ephemerale Rigs steuerbar macht, auch wenn das hier abgebildete Rig nicht ephemeral ist.

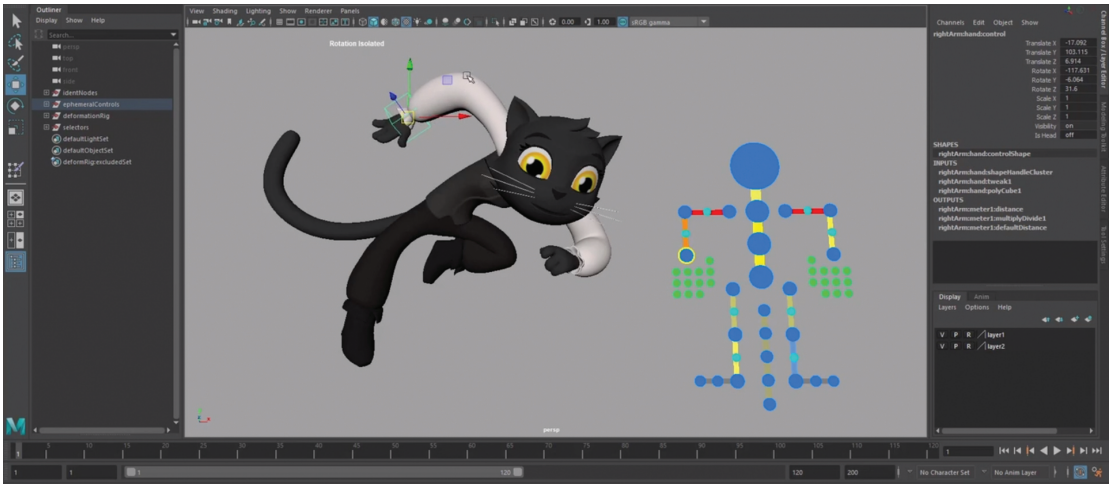


Abbildung 5.12: Der Picker von Anzovins ephemeralen Rig ist selbst geschrieben und funktioniert nur mit seinem Rig. Der Picker zeigt nicht nur die verschiedenen Controller an, sondern auch, wie sehr Glieder verzerrt sind. Die Arme des Pickers sind rot, was bedeutet, dass sie länger sind, als vom Model vorgegeben. Diese Information ist wichtig, um die Proportionen des Charakters zu erhalten. Bildquelle [9].

den Rig überaus schnell, da beim Abspielen keine Berechnungen getätigt werden. Die gesamte Information der Animation ist bereits auf den Joints gespeichert. Der größte Unterschied liegt jedoch in der Bedienung des Rigs und im Transport der Information zu den Joints. Durch die flache Hierarchie der Controller kann jedes Gelenk unabhängig

von den anderen bewegt werden. Dieser änderungsfreudige Ansatz macht das Erstellen und vor allem das Verfeinern von Posen wesentlich intuitiver. Zusätzliche Hilfen wie das Nachahmen eines vorwärtskinematischen Verhaltens kompensieren die Nachteile, die beim Entfernen von Hierarchien auftauchen. Der Motion Layer besteht zum größten Teil aus Skripten, die die logischen Hierarchien aufbauen und die Joints dementsprechend bewegen. Diese Flexibilität ermöglicht die einfache Animation für komplexe Bewegungen. Bei allen Vorteilen besitzen ephemerale Rigs jedoch auch einige Nachteile, die in Kapitel 4 hergeleitet wurden. Durch die unsaubere Interpolation bei Rotationen muss jedes einzelne Frame von Animatoren überprüft und korrigiert werden. Je höher die Framerate des Filmes, desto geringer wird der Geschwindigkeitsvorteil, der bei der Erstellung der Pose erzielt wurde. Für realistische Animationen mit 24 Bildern pro Sekunde oder Spielen mit 30 Bildern pro Sekunde, sind diese Rigs nicht geeignet. Der Stil des Filmes spielt eine große Rolle in der Frage, ob ephemerale Rigs von Vorteil sind oder nicht.

Kapitel 6

Fazit

Komplexe Bewegungen rufen mit traditionellen, hierarchischen Rigs immer einen größeren Mehraufwand in der Animationsarbeit hervor. Die Analyse in den vorhergehenden Kapiteln erklärt die Herkunft des Problems und die Gedankengänge, die zu den Lösungsansätzen führen. Eine wichtige Erkenntnis ist, dass es keinen perfekten Rig gibt. Jeder Stil benötigt eine andere Herangehensweise, und ein Rig, der gut in Szenario A funktioniert, kann katastrophal in Szenario B sein. Hierarchielose Rigs sind für realistische Animationen nicht geeignet, während cartoonhafte Animationen von hierarchischen Rigs eingeschränkt werden. Realistische Animationen benötigen Rigs, die die Realität besser nachahmen können, und Bewegungen zulassen, wie sie anatomisch möglich sind. Darüber hinaus sollen nachvollziehbare Bewegungen einfach erstellt werden können, um von einer Pose zur nächsten zu kommen, ohne Gelenke oder Knochen zu brechen. Die physische Beschaffenheit des Körpers ist bei cartoonhaften Animationen nebensächlich. Die Form des Körpers als Ganzes und die Übertreibung stehen im Vordergrund. Dabei kann Anatomie hinderlich sein. Wenn gebogene Arme die Animation besser machen, so sollte dies möglich sein. Auch müssen die Übergänge zwischen den Posen nicht logisch sein, sondern richtig aussehen. Hier sind ephemerale Rigs praktisch, da jeder Controller im Rig unabhängig manipulierbar ist, aber Abhängigkeiten je nach Situation aktiviert werden können. Da die Grenzen der Stile fließend sind, sind auch die Anforderungen an die Rigs nicht so schwarz-weiß, wie hier in diesen beiden Extrembeispielen dargestellt. Ein Hybrid aus den beiden Extremen ist der vorgestellte prozedurale Rig, der seine Hierarchien jederzeit an die Situation anpassen kann. Er erlaubt nicht dieselbe Freiheit wie ein hierarchiefreies Rig, das ist aber Teil des Kompromisses. Ein weiterer Faktor für die Wahl des Rigs ist das Zielmedium. Film und Fernsehen benötigen maximal 25 Bilder pro Sekunde. Spiele verwenden höhere Frequenzen, welche entweder 30 oder 60 Herz betragen. Je niedriger die Frequenz, desto weniger wichtig wird Interpolation. Je unwichtiger Interpolation wird, desto weniger wichtig werden Hierarchien. Die Wahl der Frequenz spielt also ebenfalls eine Rolle beim Bau eines Rigs.

Da die Idee von ephemeralen und damit einhergehend hierarchielosen Rigs relativ neu ist, ist die Entwicklung spannend zu verfolgen. Vor allem, ob realistische Rigs Teile des ephemeralen Ansatzes übernehmen und für ihre Zwecke adaptieren. Spannend bleibt auch, ob der neue Ansatz sich gegen etablierte Techniken durchsetzen und eventuell eine Art Standardform des Rigs werden kann. Die Entscheidung dazu liegt nicht bei den Riggern, sondern bei den Animatoren. Denn, wie im ersten Kapitel erwähnt, Rigs

werden für Animatoren gebaut. Wenn diese mit der neuen Technik besser arbeiten können, können hierarchielose Rigs eine Standardform werden.

Quellenverzeichnis

Literatur

[1] Maria Lee, Nancy Tsang und Trevor Tsung-Yin Hsieh. „Creating Cute Characters in Finding Dory: Baby Dory and Destiny“. In: *SIGGRAPH ASIA 2016 Computer Animation Festival*. SA '16. Macau: ACM, 2016, 52:1–52:2. URL: http://doi.acm.org/10.1145/2997500.3012318 (siehe S. 3).

[2] Wolfram Luther und Martin Ohsmann. *Mathematische Grundlagen der Computergraphik*. Braunschweig/Wiesbaden: Springer Verlag, 2013 (siehe S. 10, 11).

[3] Tim McLaughlin, Larry Cutler und David Coleman. „Character Rigging, Deformations, and Simulations in Film and Game Production“. In: *ACM SIGGRAPH 2011 Courses*. SIGGRAPH '11. Vancouver, British Columbia, Canada: ACM, 2011, 5:1–5:18. URL: http://doi.acm.org/10.1145/2037636.2037641 (siehe S. 2, 3).

[4] Tina O'Hailey. *Rig it right! Maya animation rigging concepts*. Burlington: Routledge, 2018 (siehe S. 15, 36).

[5] Christian Ullenboom. *Java ist auch eine Insel*. Bonn: Galileo Press, 2009 (siehe S. 1).

[6] John Vince. *Geometry for computer graphics*. London: Springer, 2005 (siehe S. 6–9).

[7] Richard Williams. *The animator's survival kit. A manual of methods, principles and formulas for classical, computer, games, stop motion and internet animators*. New York: Macmillan, 2012 (siehe S. 26).

Audiovisuelle Medien

[8] *Ein Paket, das keinen Namen trägt*. Animationsfilm, Masterprojekt von Markus Hadinger, Christoph Lendenfeld, Kyra von Baeckmann, Doris Rastinger. 2018 (siehe S. 42).

[9] *Ephemeral Rig Interaction — suspend mode*. Juni 2018. URL: https://vimeo.com/274607054/baa3c99e8a (siehe S. 42).

[10] *Monkey animation demo*. Okt. 2017. URL: https://vimeo.com/238447747 (siehe S. 40).

[11] *Spider-Man: Into the Spider-Verse*. Animationsfilm. Drehbuch: Phil Lord Regie: Bob Persichetti, Peter Ramsey, Rodney Rothman. 2018 (siehe S. 27).

[12] *SPIDER-MAN: INTO THE SPIDER-VERSE - Official Teaser Trailer*. Dez. 2017. URL: https://www.youtube.com/watch?v=ii3n7hYQOl4 (siehe S. 28).

[13] *Supercharged Animation Performance in Maya 2016*. Juni 2015. URL: https://www.youtube.com/watch?v=KKC7A9bbUuk (siehe S. 3).

[14] *The Lego Movie*. Animationsfilm. Drehbuch/Regie: P. Lord, C. Miller Produktion: I. Khait, R. Lee, D. Lin, J. P. Middleton. 2014 (siehe S. 27).

[15] *The LEGO® Movie - Official Main Trailer [HD]*. Okt. 2013. URL: https://www.youtube.com/watch?v=fZ_JOBCLF-I (siehe S. 28).

[16] *The Peanuts Movie*. Animationsfilm. Regie/Drehbuch: S. Martino Produktion: P. Feig, B. Schulz, C. Schulz, M. J. Travers, C. Uliano. 2015 (siehe S. 27).

[17] *The Peanuts Movie Official Trailer 1 (2015) - Animated Movie HD*. Juni 2015. URL: https://www.youtube.com/watch?v=zQpUQPrAfQM (siehe S. 28).

Software

[18] *Adobe Animate*. URL: https://www.adobe.com/at/products/animate.html (siehe S. 39).

[19] *AnimSchoolPicker*. URL: https://www.animschool.com/pickerInfo.aspx (siehe S. 41).

[20] *Autodesk Maya*. URL: https://www.autodesk.com/products/maya/overview (siehe S. 12).

[21] *Onion Skin Renderer*. URL: https://github.com/Viele/onionSkinRenderer (siehe S. 39).

[22] *ToonBoom Harmony*. URL: https://www.toonboom.com/ (siehe S. 39).

[23] *TVPaint*. URL: https://www.tvpaint.com/ (siehe S. 39).

[24] *TweenMachine*. URL: https://justinsbarrett.com/tweenmachine/ (siehe S. 40).

Online-Quellen

[25] Raf Anzovin. *Hacking The Maya Animation System*. 2017. URL: https://www.justtodosomethingbad.com/blog/2017/11/4/hacking-the-maya-animation-system (besucht am 28. 07. 2018) (siehe S. 32).

[26] Raf Anzovin. *How to construct a node graf*. 2017. URL: https://www.justtodosomethingbad.com/blog/2018/5/23/2ryvzcmndukmxitm5qpoiangx5lu5s (besucht am 30. 07. 2018) (siehe S. 33, 34).

[27] Raf Anzovin. *Just do Something Bad!* 2017. URL: https://www.justtodosomethingbad.com (besucht am 28. 07. 2018) (siehe S. 29).

[28] Raf Anzovin. *Nuts and Bolts*. 2017. URL: https://www.justtodosomethingbad.com/blog/2017/10/22/nuts-and-bolts (besucht am 28. 07. 2018) (siehe S. 31).

[29] Raf Anzovin. *Trapped in Keyframe Interpolation!* 2017. URL: https://www.justtodosomethingbad.com/blog/2017/10/2/the-only-good-function-curve-is-a-dead-function-curve (besucht am 28. 07. 2018) (siehe S. 27).

[30] Autodesk. *Nodes and attributes*. 2018. URL: https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2015/ENU/Maya/files/Nodes-and-attributes-Nodes-and-attributes-overview-htm.html (besucht am 06. 09. 2018) (siehe S. 12, 14).

[31] Peter Bui. *Lecture 4: Transformations and Matrices*. 2010. URL: https://www3.nd.edu/~pbui/teaching/cse.40166.fa10/slides/Lecture_4_Transformations_and_Matrices.pdf (siehe S. 6).

[32] Rafael Fragapane. *Cult of Rig*. 2017. URL: http://www.cultofrig.com/ (besucht am 28. 07. 2018) (siehe S. 29).

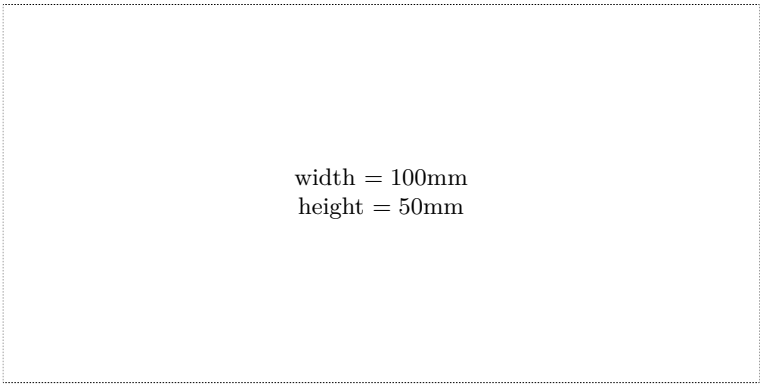
[33] Richard Lico. *Animating Quill: Creating an Emotional Experience*. Youtube. 2018. URL: https://www.youtube.com/watch?v=u3CzLVpuE4k (besucht am 28. 07. 2018) (siehe S. 20, 31).

[34] Jason Schleifer. *Animator Friendly Rigging*. 2008. URL: http://jasonschleifer.com/afr/ (besucht am 30. 07. 2018) (siehe S. 3).

[35] John Wiley. *About Broken-Joint Skeletons*. 2015. URL: http://help.autodesk.com/view/MAYAUL/2016/ENU/?guid=GUID-1A61DE91-4B0A-4864-9728-6014FD97FB0C (siehe S. 29).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —