Diplomarbeit

Fachgebiet der Diplomarbeit:
Graphische Datenverarbeitung

Thema der Diplomarbeit:

Untersuchungen zur effektiven Animation eines computergenerierten Charakters

Diplomand: Raschid Johannes Abdul-Nour

Referent: Prof. Dr.-Ing. Dipl.-Math. Monika Lutz

Korreferent: Prof. Dr. Manfred Merkel

Wintersemester 2003

Fachhochschule Gießen-Friedberg

Bereich Friedberg

Fachbereich IEM

Fachrichtung Medieninformatik

Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfasst habe und nur die in der Arbeit angegebenen Hilfsmittel und Literaturstellen verwendet habe.

Büdingen, den 1. November 2003 Raschid Abdul-Nour

Danksagung

Ich möchte mich an dieser Stelle bei Herrn Norbert Friedl für die Anregungen beim Charakterentwurf bedanken. Eckhard Kunz sei an dieser Stelle ebenfalls gedankt, der seine beiden äußerst geduldigen Teckel Atilla und Allané für Animationsstudien zur Verfügung stellte.

Ein besonderer Dank geht an meine Professorin Prof. Dr.-Ing. Dipl.-Math. Monika Lutz sowie an Laboringenieurin Dipl.-Math. (FH) Sonja Emmel, ohne deren Unterstützung diese Arbeit unmöglich gewesen wäre.

Nicht zuletzt möchte ich mich bei einigen Freunden und Verwandten bedanken, die mir während des Verfassens dieser Arbeit mit Rat und Tat zur Seite standen, sei es, indem sie etwa die Arbeit Korrektur lasen oder auch nur moralische Unterstützung leisteten. Es seien nur einige namentlich genannt: Beim Korrekturlesen dieser Arbeit waren mir Sascha Schäfer, Adel Abdul-Nour und besonders Christian Lohrey außerordentlich behilflich. Weitere Materialien zum Studieren der Anatomie von Hunden erhielt ich von Simon Duda und Julia Eschenauer, und Marcus Griebel war mir bei einigen Feinheiten der Typografie und des Druckes behilflich.

Außerdem haben sich einige Experten auf dem Gebiet der Charakteranimation bereit erklärt, mir einige Fragen zu diesem Thema zu beantworten. Ein Dank geht daher an Sebastian Faber (LIGA_01 Computerfilm), Thomas Helzle (ScreenDream) und Jörg Liebold (Freelancer) sowie an Wolfgang Borgfeld für die freundliche Vermittlung.

Diese Arbeit ist meiner Mutter gewidmet.

Inhalt

1 Einführung

1.1 Herausforderungen der Charakteranimation

Seit etwa Mitte der 70er Jahre kann eine Renaissance der Animation als eine Form des Massenentertaiments beobachtet werden. Sie führte dazu, dass alte Techniken, die schon kurz nach der Erfindung des Films bis in die 40er Jahre des 20. Jahrhunderts hinein entwickelt wurden, wieder aktuell wurden. Nur wenig später, durch das Aufkommen der Computergrafik und –animation, gelang es weiter, die Bedeutung dieses Bereichs in der Unterhaltungsindustrie zu stärken. Inzwischen ist, auch etwa durch Computerspiele, die Charakteranimation so wichtig wie nie zuvor (Williams2001).

Bei der Betrachtung dieses Themas darf auch nicht außer Acht gelassen werden, dass speziell deutsche beziehungsweise europäische Firmen der Entwicklung in diesem Bereich den großen Unternehmen in den USA immer noch etwas hinterher hinken. Während die Anwendung der Computeranimation zum Zwecke einfacherer Animationen inzwischen in vielen europäischen Unternehmen üblich ist, ist die Animation eines kompletten Charakters noch relativ selten in einem gewerblichen Umfeld zu finden.

Die Modellierung und das Animationssetup eines Charakters ist besonders herausfordernd, weil Techniken aus den unterschiedlichsten Bereichen der Computergrafik eingesetzt werden müssen.

Für eine glaubhafte Charakteranimation muss sowohl die Gestik als auch die Mimik des Charakters animierbar seien. Die Animation des Körpers wird dabei vor allem durch Envelope-Deformationen und Inverse Kinematik erreicht, die des Gesichts durch Shape-Animation mithilfe von Tools der Nonlinearen Animation.

Für die häufige Verwendung eines Charakters ist es wichtig, die Animationsteuerung möglichst einfach zu gestalten. Nur dann ist eine Verwendung in einem gewerblichen Umfeld rentabel.

1.2 Ziel dieser Arbeit

Die vorliegende Arbeit will den technischen Hintergrund der Charakteranimation untersuchen und den aktuellen Stand in diesem Fachgebiet der Grafischen Datenverarbeitung dokumentieren. Dabei ist das Ziel, die Steuerung eines Charakters zu entwickeln, die möglichst leicht erlernbar und effizient zu bedienen ist.

Im praktischen Teil der Arbeit sollen die Schwierigkeiten und Herausforderungen, die sich aus der Umsetzung der Ideen in die Praxis ergeben, untersucht werden. Beispielhaft wurde dazu ein kompletter Charakter entworfen und für diesen Charakter ein Control-Rig konstruiert, also eine Anordnung von Elementen, welche die Steuerung des Charakters ermöglichen. Außerdem wurde eine grafische Benutzeroberfläche entwickelt, welche die Steuerung des Rigs so weit wie möglich vereinfachen soll. Hierfür wurde Softimage|XSI in der Version 3.5 verwendet, welches eines der führenden Softwarepakete auf diesem Gebiet darstellt. Das hier Besprochene ließe sich auf ähnliche Weise auch auf anderen Animationssystemen durchführen.

Der Charakter, der nach dem Vorbild eines Teckels, einer Hunderasse ähnlich dem Dackel, entworfen wurde, ist dabei in Zusammenarbeit mit einem Komikzeichner entstanden. Der Charakter oder eine Weiterentwicklung soll möglicherweise einmal in einer längerfristigen kommerziellen Serienproduktion eingesetzt werden. Daraus folgt, dass sich dieser Charakter durch eine besonders leicht bedienbare Animationssteuerung auszeichnen soll, die damit gegebenenfalls auch einmal von anderen Animatoren leicht zu erfassen ist.

Aus dem Arbeitsablauf beim Erstellen dieses Charakters mit seinem Rig und seiner Animation ergibt sich unmittelbar die Gliederung dieser Arbeit, auf den Arbeitsablauf wird genauer in Kapitel 2.4, „Der Animationsprozess“, eingegangen.

In Kapitel 2, „Grundlagen der Charakteranimation“, soll zunächst ein theoretischer Überblick über das Thema gegeben sowie die wichtigsten Begriffe aus diesem Bereich erläutert werden um das weitere Verständnis zu erleichtern.

Kapitel 3, „Erstellen des Charakters“, befasst sich dann mit dem Entwurf des Charakters und dem Planen der späteren Animationen, also Überlegungen, die bereits zu Beginn des Projekts anstehen, aber für den Erfolg desselben entscheidend sind. Das Kapitel befasst sich weiterhin mit dem Prozess des Modelings und ähnlichen Tätigkeiten, da diese auch für die Animation weitreichende Folgen haben.

Anschließend werden dem Leser in Kapitel 4, „Technische Mittel zum Aufbau des Control-Rigs“, als Vorbereitung auf die nächsten Schritte zunächst die wichtigsten Hilfsmittel hierfür vorgestellt, die speziell Softimage|XSI zur Verfügung stellt.

Kapitel 5 befasst sich mit einem der entscheidenden Aufgaben in diesem Zusammenhang, dem „Aufbau des Control-Rigs“, während sich Kapitel 6 mit der „Entwicklung einer grafischen Benutzeroberfläche zur Animationssteuerung“ beschäftigt. Diese Benutzeroberfläche ist besonders wichtig, da sie dem Animator, der häufig aus einem eher künstlerischem Umfeld kommt, ein angenehmes und effizientes Arbeiten in den häufig sehr komplexen grafischen Benutzeroberflächen heutiger 3D-Animations-Softwarepakete ermöglichen soll.

Kapitel 7, „Die Anwendung in der Praxis“, soll den Umgang mit der entstandenen Charaktersteuerung noch einmal erläutern, sowie die Wiederverwendbarkeit der Technologie für spätere Projekte untersuchen.

Abschließend soll Kapitel 8, „Fazit“, das vorliegende Gesamtergebnis bewerten.

2 Grundlagen der Charakteranimation

2.1 Einleitung

Ziel dieses Kapitels ist es, einen theoretischen Überblick über das Thema Charakteranimation zu geben und die wichtigsten Begriffe aus diesem Bereich zu erläutern. Damit soll das weitere Verständnis erleichtert werden.

Die folgende Untersuchung des üblichen Animationsprozesses und seiner Besonderheiten sowie der Schwierigkeiten bei der Aufgabe, einen lebendig wirkenden Charakter zu animieren, dient dazu, die Herausforderungen zu verdeutlichen, die an eine benutzerfreundliche Charaktersteuerung gestellt werden.

2.2 Eine Definition der Charakteranimation

Lange vor dem Siegeszug der Computer war die Animation, und auch die Charakteranimation, obwohl dieser Begriff noch nicht geprägt war, schon eine anerkannte Kunstform. Maßgeblichen Anteil daran hatte vor allem Walt Disney mit seinen unnachahmlichem Gespür für Entertainment, die seine Zeichentrickfilme zu den Wegbereitern dieser Kunstform als Massenunterhaltung machten (Thomas1981).

Das Wort „Animation" leitet sich aus dem lateinischen Wort „animare" ab und bedeutet etwa „beleben" oder „beseelen". Obwohl aus dieser Tatsache beinahe direkt folgt, dass lebendige Charaktere Gegenstand der Animation sind, waren die Computeranimationen der ersten Jahre bevorzugt nicht lebende Objekte. Die Ursachen dafür sind die technischen Vorteile, welche die Animation starrer Körper, die sogenannte „Rigid Body Animation", gegenüber der „Soft Body Animation" mit sich bringt (Watt2002).

Eine eher technische Definition des Begriffes besagt, dass Animation das Erstellen von Sequenzen von zusammenhängenden Bildern ist, die in einer hinreichend schnellen Geschwindigkeit abgespielt werden, um die Trägheit des menschlichen Auges zu überlisten und den Eindruck von realer Bewegung entstehen zu lassen (Arima2001).

Technisch betrachtet ist der Unterschied zwischen Animation und Charakteranimation nicht allzu groß. Künstlerisch allerdings ist der Unterschied bedeutend. Das Ziel der Charakteranimation, einen lebendig wirkenden Charakter zu erzeugen, ist schon an der Wortbedeutung zu erkennen. Dieses Ziel, einen nicht existierenden, virtuellen Charakter lebendig wirken zu lassen, sollte sich allerdings als äußerst schwierig erweisen.

Einer der schönsten Versuche, die Charakteranimation am Computer zu beschreiben, findet sich im Buch von Richard Williams (Williams2001):

> „If drawn ‘classical’ animation is an extension of drawing, then computeranimation can be seen as an extension of puppetry – high tech marionettes. Both share the same problems on how to give a performance with movement, weight, timing and empathy.”

Dieser Vergleich von computergenerierten Charakteren mit herkömmlichen Marionetten ist nicht nur hilfreich, um einem Laien das Thema nahe zu bringen, es verdeutlicht auch, wie wichtig neben technischer Perfektion die künstlerische Herangehensweise an derartige Aufgaben ist (Abb. 2.1). Die Technik, die Gegenstand dieser Arbeit ist, darf und muss dabei den Künstler soweit wie möglich unterstützen und ihm die Arbeit möglichst leicht machen ohne ihn zu beschränken.

Abb. 2.1: Der Puppenbauer mit seinem Meisterwerk: Szene aus dem Disneyfilm „Pinocchio“ von 1940 (Thomas1981, S. 12)

2.3 Wichtige Begriffe der Charakteranimation

2.3.1 Low-Level-Animation

Das Animieren auf „Low-Level"-Niveau bedeutet, im Gegensatz zu der im nächsten Kapitel beschriebenem „High-Level-Animation", das Animieren einzelner Parameter.

Dabei ist die am Häufigsten eingesetzte Technik das sogenannte Keyframing, das heißt das Setzen von bestimmten Werten eines Parameters zu bestimmten Zeitpunkten, also das Setzen von Keys. Der Computer übernimmt dann die Interpolation der Werte zwischen den Keys, wobei der Benutzer bei Softimage|XSI, genau wie bei anderen professionellen Animationstools, noch weitere Einflussmöglichkeiten hat. Dies geschieht über die Beeinflussung der sogenannten „Funktionskurven", die beim Setzen von Keys automatisch erzeugt werden und die den Wert des Parameters über der Zeit in Frames auftragen (Abb. 2.2).

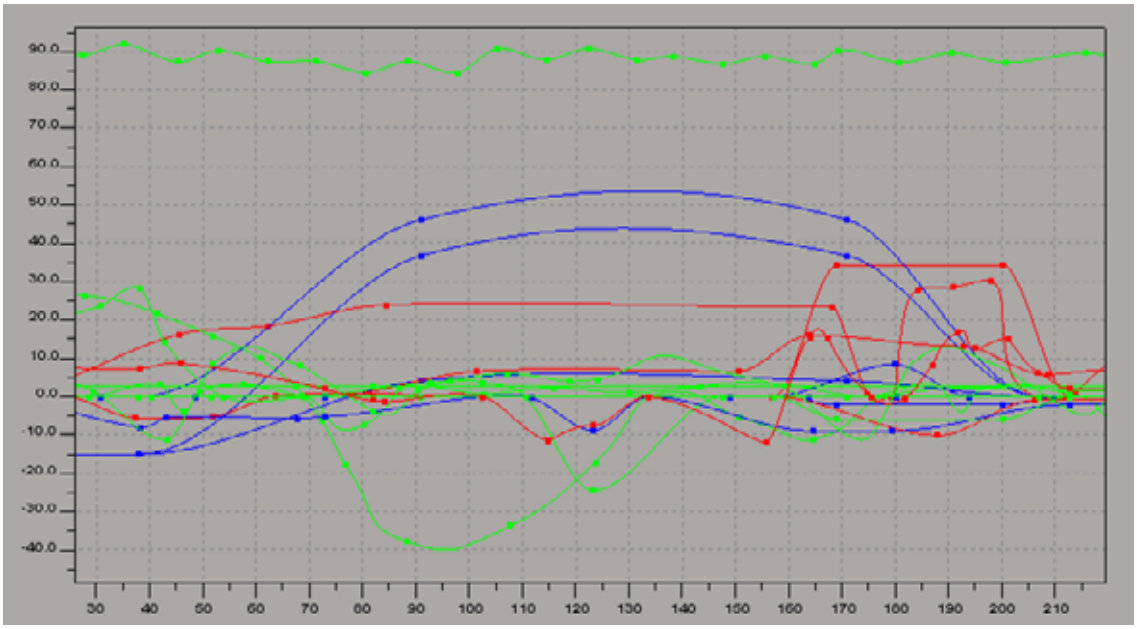

Abb. 2.2: Die Werte zwischen den Keys werden durch Funktionskurven definiert

Die Keyframing-Technik hat Ihren Ursprung lange vor dem Aufkommen der Computeranimation. Bis etwa 1920 zeichneten die meisten Zeichentrickfilm-Animatoren ihre Sequenzen normalerweise noch komplett selbst. Aus dieser Zeit ist die Geschichte überliefert, dass der Animator Dick Huemer, einer der damals führenden Animatoren, von seinem Arbeitgeber darauf angesprochen wurde, wie gut seine Zeichnungen wären und ob er nicht noch wesentlich mehr davon abliefern könnte. Seine Antwort „Give me someone to put the in-between drawings and I'll do two to three times as much work" darf wohl als Begründung eines Berufszweiges, der sogenannten „Inbetweener", angesehen werden (Williams2001).

Prinzipiell übernimmt in heutigen Animationsprogrammen der Computer die Rolle des Inbetweeners, weshalb in einigen Programmen, wie etwa „Macromedia Flash“, der Vorgang der Interpolation noch als „tweening“ bezeichnet wird (Williams2001).

Das Keyframing ist nicht die einzige Technik der Low-Level-Animation, in Softimage|XSI zählt auch das Animieren über Constraints (in deutsch etwa: „Beschränkungen“), Linked Parameters („verlinkte Parameter“), Expressions und Scripted Operator (kurze, in die Szene eingebaute Programmstücke) sowie über den direkten Input von Eingabegeräten mithilfe des sogenannten „Device Driver-Moduls“, zu den Low-Level-Methoden. Diese Techniken werden in Kapitel 4 näher erläutert (XsiDoc2002, Animation).

Die vorliegende Arbeit hat sich zum Ziel gesetzt, einen Charakter mitsamt einem, die Bewegung steuernden, Control-Rig, zu entwerfen, und damit eine besonders effiziente Low-Level-Animation zu ermöglichen.

2.3.2 High-Level-Animation

Im Gegensatz zu der Low-Level-Animation werden bei der High-Level-Animation nicht die einzelnen Keys direkt editiert. Stattdessen werden „Animationsblöcke“ aus Keys, Constraints oder Ähnlichem gebildet, die dann zum Beispiel mit Hilfe des Animation Mixers miteinander kombiniert werden können (siehe dazu auch Kapitel 2.3.3).

Ein Anwendungsbeispiel für die High-Level-Animation wäre es etwa, verschiedene statische Posen eines Charakters mit Low-Level-Tools zu erstellen und diese dann anschließend mithilfe von High-Level-Tools ineinander übergehen zu lassen, um sie so zu Animationen zu verarbeiten.

Die Erfahrung hat jedoch gezeigt, dass die Tools der High-Level-Animation von einem Großteil der professionellen Animatoren zur Zeit kaum eingesetzt werden. Gründe dafür dürften in dem Wunsch nach maximaler Kontrolle auch über Details liegen, in der teilweise noch etwas praxisfremden Umsetzung der Tools in der grafischen Benutzeroberfläche, oder auch nur in der fehlenden Gewöhnung an diese Tools (siehe dazu auch „Anhang B: Ausgewählte Expertenmeinungen“).

Dies ist einer der Gründe, weshalb sich die vorliegende Studie vorwiegend auf die Low-Level-Animation bezieht. Auch auf die, zu dieser Studie gehörende, praktische Arbeit haben die Besonderheiten der High-Level-Animation nur relativ wenig Einfluss genommen, da XSI für die High-Level-Animation eigene Tools zu Verfügung stellt.

2.3.3 Nonlineare Animation

Eine Ansammlung der zum Thema Charakteranimation gehörenden Begriffe, besonders im Umfeld von Softimage|XSI, kann nicht ohne den Begriff der „Nonlinearen Animation“ auskommen, auch wenn sich dieser Begriff mit der „High-Level-Animation“ zu großen Teilen überschneidet.

Das „lineare" in diesem Begriff bezieht sich hier auf den Arbeitsablauf, der non-destruktiv ist und damit ein nicht-lineares Arbeiten erlaubt. Konkret bedeutet dies, dass auch nach Abschluss des größten Teils einer Animation auch noch einzelne „Bauteile" geändert werden können, etwa ein Walk-Cycle, und diese Änderungen dann in der gesamten Animation aktualisiert werden. Umgekehrt kann, indem diese Bauteile anders miteinander arrangiert werden, eine völlig andere Animation entstehen (XsiDoc2002, Animation).

Diese Bauteile werden in XSI „Action Sources" oder auch nur „Actions" genannt, sie können jede beliebige Low-Level-Animation enthalten, etwa Keyframe-Animation, Constraints oder auch Expressions. Von einer Action Source können beliebig viele Instanzen angelegt werden, diese nennt man dann „Action Clips" oder nur „Clips". Clips wiederum können, hauptsächlich der Übersicht halber, zu anderen Clips zusammengefasst werden, den sogenannten „Compound Clips".

Das Tool zur Nonlinearen Animation in Softimage|XSI wird „Animation Mixer" genannt. Dieses Tool, das vom Aufbau her einer Videoschnittsoftware ähnelt, ermöglicht es Gruppen von Animationen, die oben genannten Clips, miteinander zu kombinieren ohne den Inhalt der Clips selbst zu verändern. Dazu können die Clips etwa in einer Schleife abgespielt werden, sie können durch Übergänge ineinander übergeblendet werden und sie können sogar gleichzeitig abgespielt werden, wobei eine Gewichtung der einzelnen Clips gegeneinander stattfinden kann.

In Softimage|XSI können Elemente einer Szene in einer, „Model" genannten, Datenstruktur zusammengefasst werden. Einer der Eigenschaften dieser Models ist, das für jedes Model ein eigener Mixer angelegt wird. Standardmäßig ist jede Szene selbst ein Model, es können jedoch in einer Szene zusätzliche Models angelegt werden.

2.3.4 Inverse Kinematik

Bevor der Begriff der Inversen Kinematik erklärt werden kann, sollte zunächst der grundlegende Begriff der Kinematischen Ketten und der Vorwärts-Kinematik („Forward Kinematic" oder kurz „FK") erklärt werden.

Kinematische Ketten sind Strukturen aus einer gegebenen Menge von Objekten, die durch Gelenke miteinander verbunden sind. Diese Gelenke erlauben den Elementen der Struktur, sich zueinander zu bewegen. Dadurch entsteht eine Hierarchie von Gelenken ,auch Knoten genannt, mit einer zugehörigen Transformation, die das dem Gelenk zugeordnete Element bewegt (Watt2002).

Objekte, die direkt in einer Hierarchie miteinander verbunden sind, bezeichnet man auch als „in einer Parent-Child-Beziehung verknüpft“, wobei das in der Hierarchie weiter oben stehende Element als „Parent“ und die ihm direkt untergeordneten Elemente als „Children“ bezeichnet werden. Softimage|XSI stellt mit dem „Draw-Chain-Tool“ ein Werkzeug zur Verfügung, welches automatisch Kinematische Ketten konstruiert.

Die Vorwärtskinematik basiert nun, vereinfacht dargestellt, darauf, sich vom Anfang der Hierarchie bis an deren Ende hinabzuarbeiten und für jedes Gelenk die zugehörige Transformation festzulegen. Es ist leicht einzusehen, dass es unter diesen Umständen äußerst schwierig ist, eine natürliche Bewegung nachzuahmen oder etwa das Ende der kinematischen Kette an einem bestimmten Ort zu platzieren.

Dennoch gibt es durchaus sinnvolle Anwendungsmöglichkeiten für die Vorwärtskinematik, nämlich immer dann wenn eine kinematische Kette entweder äußerst kontrolliert platziert werden soll oder wenn eine Bewegung explizit für bestimmte Teile einer Kette definiert werden soll. Häufig wird versucht, die Arbeit mit Vorwärtskinematik durch Expressions oder andere Arten des Scriptings zu vereinfachen. Ein Beispiel für eine derartige Anwendung der Vorwärtskinematik, wie es auch im weiteren Verlauf dieser Arbeit noch näher erläutert wird, ist etwa die Animation eines Schwanzes, bei dem sich eine Bewegung durch die komplette Kinematische Kette hindurch fortpflanzen soll.

Im Gegensatz dazu beruht die Inverse Kinematik (kurz: „IK“) darauf, nur den Anfang, das Ende und gegebenenfalls eine Rotationsebene direkt zu animieren, die Positionen und Rotationen der einzelnen Elemente der Kette werden automatisch berechnet. Der Anfang der kinematischen Kette wird dabei durch ihre „Root“ bestimmt, ihr Ende durch den sogenannten „Effektor“ (Abb. 2.3). Eine Rotationsebene, die für Ketten die flexibler zu beeinflussen sein sollen nötig ist, wird entweder durch Angabe eines „Up-Vectors“ oder einer „Resolution Plane“ definiert. Beide Methoden funktionieren vom Prinzip her ähnlich, es wird die Rotation der gesamten kinematischen Kette definiert. Der Up-Vector definiert einen Punkt, auf den die Y-Achse des ersten Knochens einer Kette ausgerichtet wird. Die Resolution Plane definiert die Achse, um welche die Kette sich dreht (XsiDoc2002, Animation).

Der Algorithmus, der die Berechnung der Transformationen der anderen Elemente übernimmt, wird „Solver“ genannt. Es benötigt neben den angegebenen animierbaren Parametern noch weitere, nicht animierbare Angaben, wie etwa die Länge der Bones und deren bevorzugter Winkel zueinander. Diese Werte werden bereits beim Zeichnen der Kette definiert, können jedoch auch später modifiziert werden.

Außerdem gibt es bei Softimage|XSI noch die Möglichkeit, Rotationsgrenzen für einzelne Bones anzugeben, die nie über- beziehungsweise unterschritten werden. Die Verwendung dieser Funktion wird jedoch von den meisten Quellen nicht empfohlen, da sie zu ungewollten Unstetigkeiten in der Animation führen kann. Stattdessen hat es sich als effektiver erwiesen, die Steuerung in diesem Bereich dem Animator zu überlassen beziehungsweise beispielsweise durch Constraints die Transformationen der Steuerungsobjekte einzuschränken (Rossano2002).

Im Übrigen besteht auch die Möglichkeit, Vorwärtskinematik und Inverse Kinematik parallel einzusetzen. Durch einen Parameter, der für jede kinematische Kette in XSI gesetzt werden kann, lässt sich dann zwischen den beiden möglichen Sets von Transformationen weich überblenden.

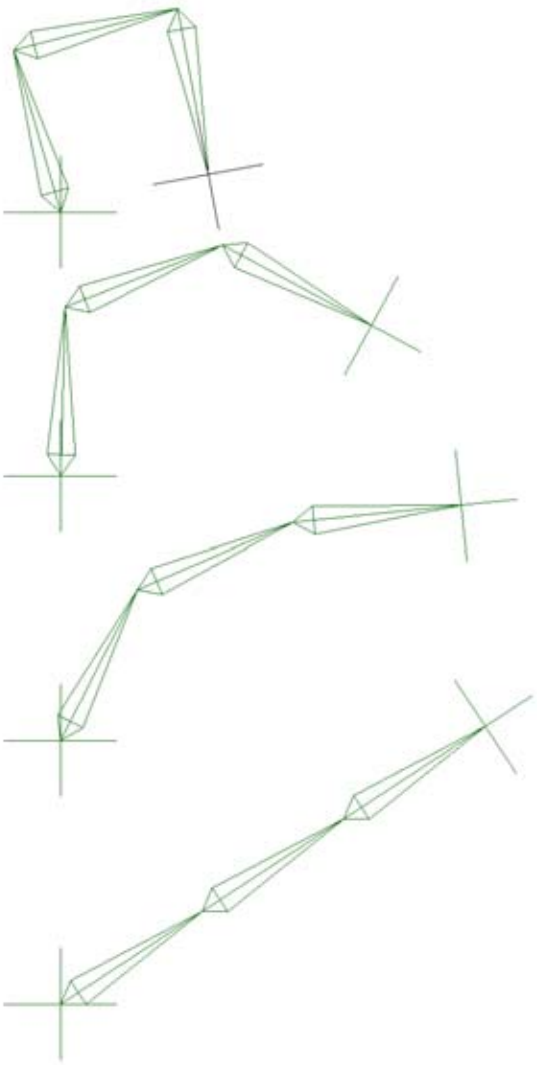

Abb. 2.3: Der Solver berechnet bei der Inversen Kinematik die Rotationen der Bones automatisch

Es sei noch erwähnt, dass kinematische Ketten, wenn man sie als Hierarchien von Objekten betrachtet, durch Constraints ergänzt werden können. Eine Struktur mit ähnlichen Eigenschaften wie eine Hierarchie von Objekten die durch Parent-Child-Beziehungen miteinander verbunden sind, kann ebenfalls durch Constraints konstruiert werden (vergleiche Kapitel 4.2, „Hierarchien und Constraints“).

In diesem Fall besteht der größte Unterschied zu Hierarchien darin, dass explizit nur ganz bestimmte Transformationen, etwa Skalierung, Rotation oder Translation, weitergegeben werden können. Ein weiterer Unterschied besteht darin, dass ein Element, welches durch ein Constraint mit einem anderen verbunden ist, sich nicht notwendigerweise im selben lokalen Koordinatensystem befinden muss.

2.3.5 Deformartionen und Enveloping

Generell meinen Deformationen in XSI die Veränderung der Form eines Objektes im Verlauf der Zeit. Speziell in XSI gilt, dass Deformationen nicht auf Objekte angewendet werden, sondern nur auf sogenannte Cluster, also Gruppen von Geometriekomponenten. Für den Fall, dass eine Deformation auf das ganze Objekt wirken soll, erstellt XSI unter der Oberfläche automatisch einen Cluster, der alle Geometriekomponenten eines Objektes enthält (Rossano2002).

Es gibt in Softimage|XSI unzählig viele verschiedene Deformationsoperatoren, seien es einfache Deformationen wie „Bend“, „Bulge“, „Twist“, Deformationen durch andere Objekte wie „Deform by Spine“ und „Deform by Surface“ oder Deformationen durch Simulationen wie etwa die Softbody-Simulation. Wenn man aber von Deformationen im Zusammenhang mit der Charakteranimation spricht, ist meistens das sogenannte „Enveloping“ gemeint (XsiDoc2002, Modeling and Deformations).

Mit dem Begriff des Envelopings bezeichnet man im Allgemeinen das Binden von Geometrie an IK- beziehungsweise FK-Elemente. Die einfachste Methode ist dabei das sogenannte „Rigid Enveloping“. Bei dieser Methode werden, entweder durch Constraints oder durch das Aufbauen einer Parent-Child-Hierarchie, ein oder mehrere Geometrie-Objekte an ein oder mehrere IK- beziehungsweise FK-Elemente gebunden. Für das Nachbilden einer mechanischen Bewegung, wie etwa die eines Roboterarmes, ist diese Technik gut geeignet, aber organische Objekte verhalten sich im Allgemeinen nicht so starr.

Im Gegensatz zu diesem „Rigid Enveloping“ steht das „Flexible Enveloping“. Dabei wird die Oberfläche eines oder mehrerer „Envelope-Objekte“, zu deutsch „Hüll-Objekte“, durch den Einfluss eines oder mehrerer sogenannter „Deformer-Objekte“ weich verformt. Die Menge aller Envelope-Objekte, auf die ein Envelope-Deform-Operator angewendet wurde bezeichnet man als Envelope. Im Allgemeinen werden als Deformer wiederum IK- beziehungsweise FK-Elemente verwendet, in XSI kann aber theoretisch jedes Objekt als Deformer verwendet werden. Beachtet werden muss dabei allerdings, dass bei anderen Objekten als Bones nur deren Koordinatencenter für das Weighting verwendet werden.

Mit dem Weighting bezeichnet man das Festlegen der Gewichtung, mit der Deformer einen bestimmten Bereich der Hüll-Objekte beeinflussen. Zwar wird zunächst ein automatisches Weighting durchgeführt, welches entweder den Abstand oder den Normalenvektor der Hüllbereiche zu den Deformer-Objekten berücksichtigt, bei komplexeren Objekten ist ein nachträgliches Korrigieren des Weightings jedoch meistens unumgänglich.

XSI stellt mit der „Weight Map“ ein äußerst praktisches Tool zur Verfügung, das es erlaubt die Gewichtung der einzelnen Deformer direkt auf die Hülle zu „malen” und das Ergebnis in Echtzeit zu veranschaulichen. Dennoch ist der Prozess des Weightings häufig sehr zeitaufwändig.

2.3.6 Shape Animation

Im Allgemeinen ist die „Shape Animation“ auch bekannt unter dem Begriff “Morphing”, dies trifft allerdings nur auf dessen dreidimensionalen Varianten zu. Das sogenannte Pixelblending, bei dem zwei rein zweidimensionale Bilder ineinander überführt werden, wird durch den Begriff der „Shape Animation“ nicht umfasst.

Mithilfe der Shape Animation wird ein Objekt beziehungsweise ein Cluster von einer bestimmten Form in eine andere überführt. Einer breiten Öffentlichkeit wurde diese Technik 1991 durch die Effekte der Firma „Industrial Light & Magic“ für den Film „Terminator 2: Judgment Day“ bekannt gemacht, in der zum ersten Mal ein im Computer generierter Hauptdarsteller eines Filmes mit komplexen Handlungsmustern auftrat (Rahman2003).

In Softimage|XSI wird für die Shape Animation durch den Benutzer der Animation Mixer verwendet, der seine hauptsächliche Anwendung im Gebiet der nonlinearen Animation hat (Abb. 2.4). XSI erlaubt also das Morphen in einer nonlinearen Arbeitsweise, mit den Quelldaten für das Morphen kann nondestruktiv gearbeitet werden. Dazu werden sogenannte „Shape Keys“ erzeugt, die einen Zustand des betreffenden Clusters speichern. Mithilfe des Animation Mixers werden diese Shape Keys als „Shape Clips“ instanziiert und können nun gemischt und verschieden gewichtet werden (XsiDoc2002, Animation).

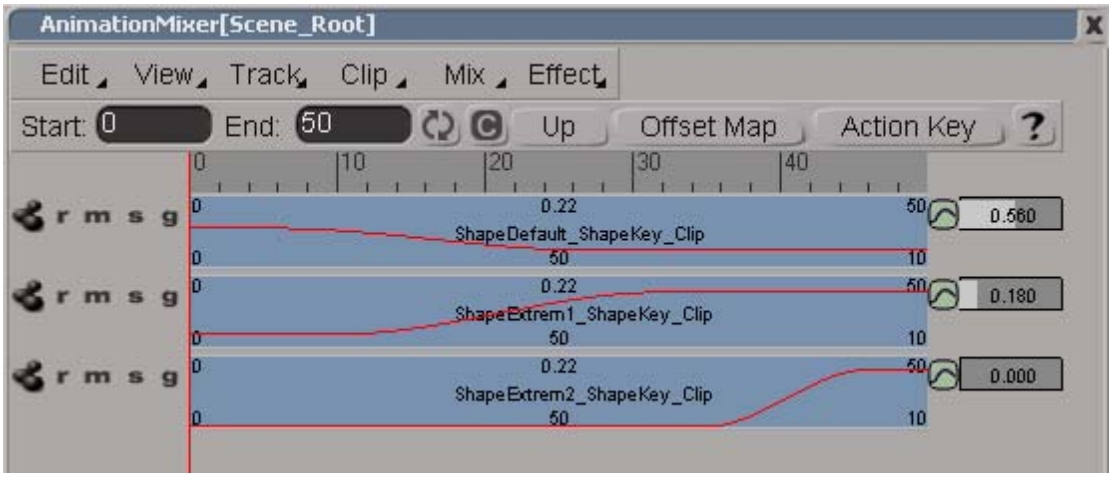

Abb. 2.4: Beispiel eines Animationsmixers, in Rot dargestellt sind die "Weight-Curves" zu erkennen

Wichtig für gute Ergebnisse in XSI ist das Wissen um den Unterschied zwischen dem Additiven und dem Normalisierten Mischen. Diese wichtige Einstellung, die für jeden Mixer separat getroffen werden muss, ist bei XSI leider etwas in den Optionen des Animationsmixer versteckt, nichtsdestotrotz überaus entscheidend. Im Normalisierten Modus wird die Summe der Gewichtungen für einen Shape-Cluster immer anteilsmäßig auf 100% beschränkt, das Ergebnis liegt, solange keine negativen Gewichtungen angegeben werden, immer „zwischen“ den ursprünglichen Shape Keys, während im Additiven Modus das Addieren oder Subtrahieren einzelner Shape Keys möglich ist.

Der Additive Modus erlaubt also eine wesentlich größere Vielfalt von Formen und größeren gestalterischen Einfluss, weshalb er für Shape-Animationen meistens besser geeignet ist (Maraffi2001 und XsiDoc2002, Animation).

2.3.7 Primär- und Sekundäranimation

Es ist eine häufig anzutreffende Praxis, die Animationen bezüglich ihrer Wichtigkeit für das Verständnis in verschiedene „Levels“ (Stufen) einzuordnen. Sie werden dann als Primäre, Sekundäre und gegebenenfalls auch Tertiäre Animation oder auch Aktionen bezeichnet. Wie viele andere Techniken auf diesem Gebiet, wurde auch diese zuerst von Animatoren des Disney-Konzerns entwickelt (Maraffi2001).

Die Primäranimation sollte danach die Animationen umfassen, welche für die Handlung unerlässlich sind und sie vorantreiben. Diese Animationen sollten vom Zuschauer leicht und ohne größere Anstrengungen erkannt werden können.

Sekundär- und eventuell auch Tertiäranimationen bezeichnen dagegen die etwas subtileren Animationen. Sie werden in den seltensten Fällen vom Zuschauer direkt bemerkt, bei ihrem Fehlen allerdings vermisst. Besonders bei lebenden Charakteren sind Sekundärbewegungen, wie etwa Muskelbewegungen oder die Atmung essentiell, um den Eindruck einer lebendigen Figur zu erhalten.

Es gibt natürlich unbegrenzte Anwendungsmöglichkeiten für Sekundäranimationen; Einige besonders sinnvolle und nötige Prinzipien sind jedoch in fast jeder Umgebung anzutreffen und können als eine Art Checkliste verwendet werden, wenn eine Animation zu leblos wirkt.

Hier ist beispielsweise das Prinzip der sogenannten „Counteraction“ zu nennen, welches die Gegenbewegung bestimmter Teile eines Objektes, wie etwa der Haare, zur Gesamtbewegung des Objektes umfasst, oder auch das Prinzip der Antizipation, welches die Vorwegnahme oder Vorbereitung einer wichtigeren Bewegung umfasst, etwa das Ausholen bei einem Wurf oder ein „in die Knie gehen“ vor einem Sprung. Weitere Beispiele für sekundäre Animationen werden in Kapitel 2.5, „Regeln für eine realistische Animation“, näher erläutert (Williams2001).

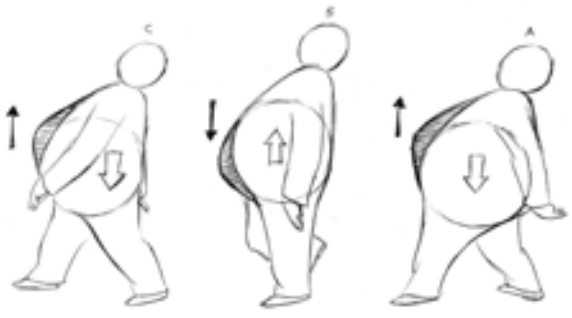

Abb. 2.5: Beispiel für eine Gegenbewegung als Sekundäranimation (Quelle: Williams2001, S. 156)

2.3.8 Charaktersteuerung

Der Begriff der Charaktersteuerung ist eng mit dem des „Character-Control-Rigs“ verknüpft, welcher im nächsten Kapitel angesprochen wird. Er bezeichnet einfach die Gesamtheit aller Methoden, die es ermöglichen, einen computergenerierten Charakter

möglichst effizient zu animieren. Die Steuerung über Steuerungsobjekte, wie sie später erläutert wird, ist dabei eine der am Häufigsten anzutreffenden Möglichkeiten, es gibt aber noch diverse andere.

Sehr häufig wird auch eine Steuerung über Slider eingesetzt. Eine Steuerung über Slider gilt für manche Anwendungen als die effizienter zu bedienende Variante, hat aber den Nachteil, das eine dreidimensionale Koordinatenangabe über Slider wesentlich weniger intuitiv ist als über Steuerungsobjekte (Abb. 2.6).

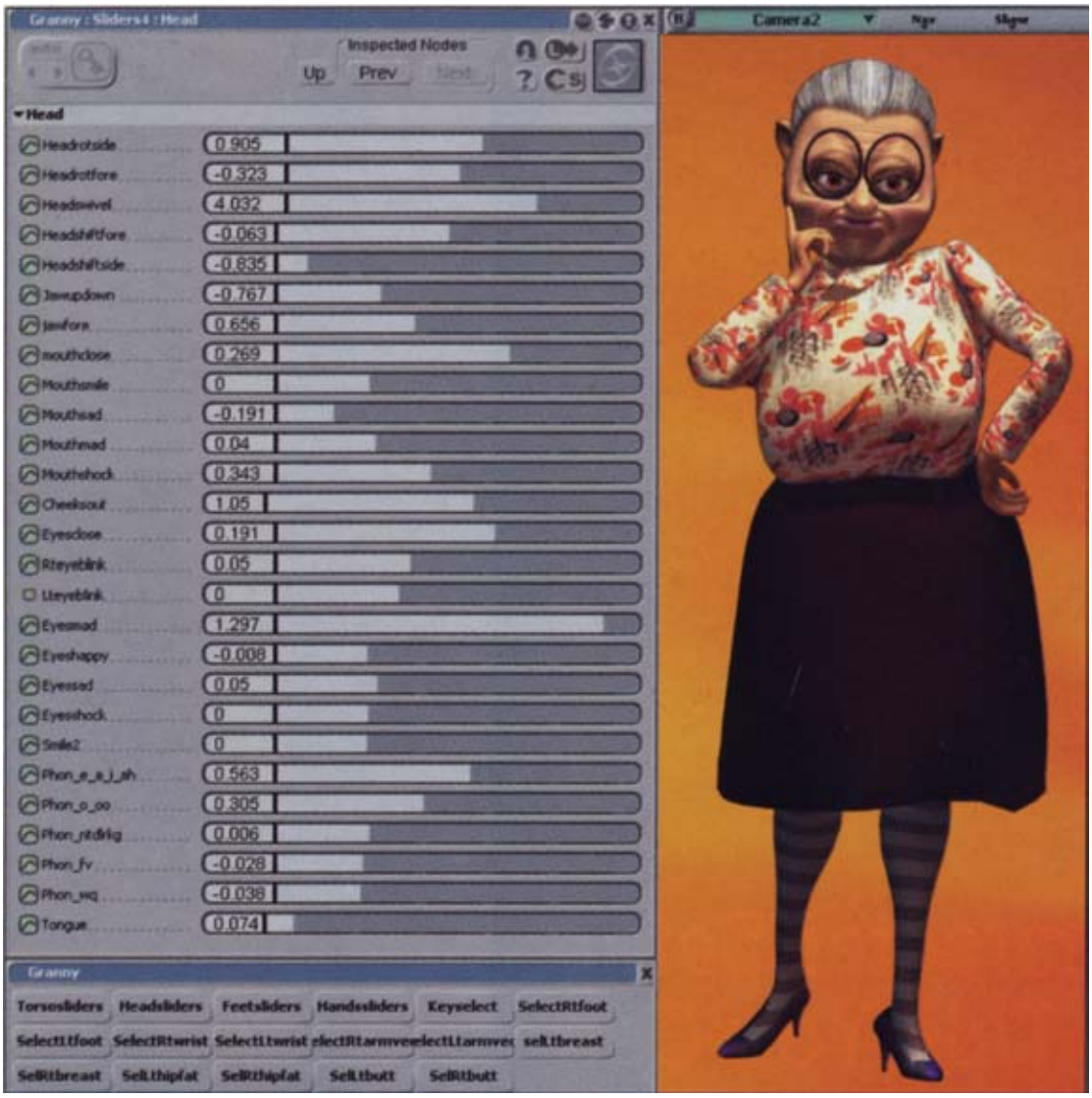

Abb. 2.6: Beispiel für eine Charaktersteuerung über Slider (Quelle: Maraffi2001)

Die Schwierigkeit bei dem Entwickeln einer guten Charaktersteuerung besteht darin, dem Animator, der möglicherweise einen weniger technischen Hintergrund hat, eine effiziente und einfache Bedienung zu ermöglichen, ohne ihn dabei in seinem Einfluss auf den Charakter einzugrenzen.

Zur Charaktersteuerung gehört auch das Entwickeln von Automatisierungen, die dafür zuständig sind, gewisse Teile des Skeletts in Abhängigkeit von Inputparametern wie etwa anderen Teilen des Skeletts, Steuerungsobjekten oder Slidern, automatisch zu bewegen. Ein Beispiel dafür ist etwa die automatische Platzierung der Hüfte und der Brust in Abhängigkeit von den Positionen der Füße. Besonders bei der Entwicklung dieser Automatisierungen besteht die große Gefahr, dem Animator Einfluss auf den Charakter zu entziehen. Da jedoch im vorhinein nur selten jede später benötigte Animation vorhergesehen werden kann, ist dieses unbedingt zu vermeiden. Eine sinnvolle Möglichkeit dazu besteht darin, jede Automatisierung durch einen zusätzlichen Parameter im Betrag ihrer Auswirkung variierbar zu programmieren. Zu diesem Thema und seiner praktischen Anwendung wird in Kapitel 5.3.3 weiter eingegangen.

Ein Großteil der Überlegungen zur Entwicklung der Charaktersteuerung beziehen sich auch auf eine effizient zu bedienende und möglichst selbsterklärende Benutzeroberfläche, auch Aspekte der Usability sind also von Bedeutung. Softimage|XSI stellt dafür mit der „Synoptic View“ ein sinnvolles Werkzeug zur Verfügung, um relativ einfach eine grafische Benutzeroberfläche zu programmieren, die, sinnvoll eingesetzt, die Steuerung des Charakters erleichtern kann (siehe dazu Kapitel 6.4, „Die Synoptic View von Softimage|XSI“).

2.3.9 Character-Control-Rig

Ein Character-Control-Rig, oder kurz Rig genannt, ist laut Definition eine Skelett-Steuerungs-Struktur. Seine Aufgabe ist, die Bedienung des Charakters zu ermöglichen und, soweit möglich, zu vereinfachen. Im Allgemeinen werden auch alle Elemente, welche die Bewegung eines Charakters überhaupt erst ermöglichen, dem Control-Rig zugeordnet. Es kann eine äußerst komplexe Struktur aus FK- und/oder IK-Elementen, zusätzlichen Deformer-Objekten, Hilfsobjekten und Steuerungsobjekten umfassen. Dem Benutzer, also meistens dem Animator, sollen dabei aber möglichst nur die für die Animation wichtigen Bestandteile präsentiert und zugänglich gemacht werden (Maraffi2001).

Für den Aufbau eines Control-Rigs können Parent-Child-Beziehungen, Constraints und Expressions verwendet werden, häufig wird eine Kombination aus allem verwendet.

In einem professionell gestalteten Rig wird der Animator so gut wie nie die Effektoren einer IK-Kette selbst animieren. Der Hauptgrund dafür liegt darin, dass Transformationen in XSI standardmäßig als lokale Parameter gespeichert werden, das heißt relativ zum Parent des animierten Objektes. Da der Effektor einer IK-Kette im Allgemeinen Child der Root der IK-Kette ist, wäre es beinahe unmöglich bei direkter Animation die Root unabhängig vom Effektor zu animieren. Ein Beispiel dafür wäre eine IK-Kette, die ein Bein einer Figur nachbildet: Will der Animator die Hüfte, die Root der Kette, unabhängig vom Bein, dem Effektor der Kette, bewegen, gelingt das nicht über direkte Animation der Bestandteile der IK-Kette (Rossano2002).

Stattdessen werden sogenannte Steuerungsobjekte verwendet, die essentieller Bestandteil des Character-Control-Rigs sind. Sie werden, meistens über Constraints, mit den Elementen der IK-Kette wie Root oder Effektor verbunden und können dann unabhängig voneinander animiert werden. Wenn man den in Kapitel 2.2 bereits zitierten Vergleich eines computergenerierten Charakters mit einer Marionette weiterführen würde, wären die Steuerungsobjekte mit dem Spielkreuz des Marionettenspielers identisch (Abb. 2.7).

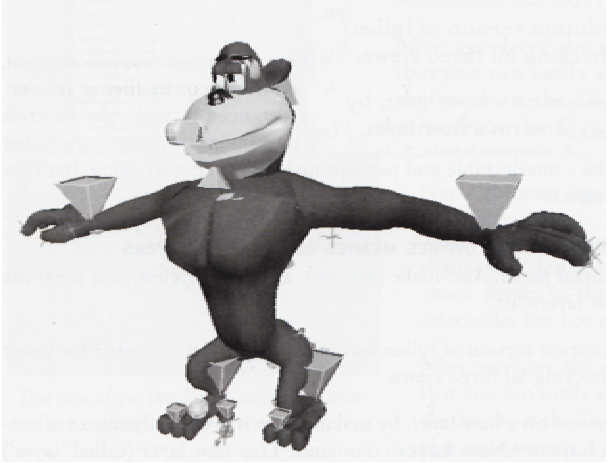

Abb. 2.7: Beispiel für eine Charaktersteuerung über Steuerungsobjekte (Quelle: Arima2001)

Ein Control-Rig wird in Softimage|XSI so gut wie immer in einem Model zusammengefasst. Ein Model ist in Softimage|XSI, wie es bereits in Kapitel 2.3.3, „Nonlineare Animation“, erläutert wurde, eine Datenstruktur bestehend aus verschiedenen Elementen. Für jedes Model wird ein eigener Mixer erstellt. Dadurch ist es möglich, Actions für dieses Rig im Mixer des Models zu speichern. Ein weiterer Vorteil eines Models, der besonders bei der Verwendung für ein Control-Rig zum Tragen kommt, ist, dass jedes Model seinen eigenen Namensraum hat. In einer Szene können also mehrere Objekte den selben Namen haben, wenn sie in unterschiedliche Models integriert sind. Das ist besonders dann wichtig, wenn etwa mehrere Instanzen des gleichen Rigs in einer Szene verwendet werden sollen.

Es ist nicht nötig, für jeden Charakter ein eigenes Rig zu erstellen. Für weniger komplexe Anwendungen stellt XSI standardmäßig zwei Rigs zur Verfügung, eines für Zweibeiner und eines für Vierbeiner (XsiDoc2002, Animation). Diese beiden Rigs sind allerdings nicht besonders stark automatisiert und müssten für eine professionelle Anwendung in den meisten Fällen wohl noch an die individuellen Bedürfnisse angepasst werden. Im Rahmen dieser Arbeit soll im Gegensatz dazu ein Rig für einen Vierbeiner von Grund auf neu entwickelt werden. Dabei soll unter Anderem darauf geachtet werden, dass dieses Rig leicht auf andere Charaktere und Anwendungen angepasst werden kann.

2.4 Der Animationsprozess

Die Abfolge der Tätigkeiten, die letztendlich zu einer Charakter-Animation führen, können als „Charakter-Animationsprozess“ bezeichnet werden. Verschiedene Quellen empfehlen hier teilweise etwas unterschiedliche Reihenfolgen, auch kann der Prozess durchaus nach dem persönlichen Geschmack variiert werden (Maraffi2001).

Für eine erfolgreiche Produktion sind jedoch die im Folgenden erläuterten Tätigkeiten meistens zumindest in einer ähnlichen Reihenfolge sinnvoll. Der hier empfohlene Ablauf ist im Übrigen auch weitgehend in der Struktur der vorliegenden Arbeit wiederzufinden.

Eine gute Planung sollte generell am Anfang jedes Projektes stehen. In diesem Fall heißt das, dass es enorme Ressourcen sparen kann, wenn möglichst früh ein möglichst umfassender Entwurf eines Charakters feststeht. Dazu gehört die Entwicklung der Story, des Aussehens des Charakters, erste Animationsstudien und dergleichen (Kapitel 3.2).

Als nächstes folgt meistens das tatsächliche Erstellen des Charakters, was die Modellierung (Kapitel 3.3) sowie das Vorbereiten des Renderns und die Texturierung umfasst (Kapitel 3.4). Häufig wird auch empfohlen, das Modeling und die Texturierung erst nach der eigentlichen Animation zu verfeinern, da besonders Anfänger oft den Fehler machen, zu viel Zeit in die Modellierung eines Charakters zu stecken. Diese Zeit fehlt dann gegen Ende des Projekt häufig (Maraffi2001).

Vor der eigentlichen Animation muss der Aufbau beziehungsweise bei gegebenem Rig die Anpassung des Control-Rigs erfolgen, was das Entwickeln der Skelettstruktur, das Erstellen der Steuerungsobjekte und gegebenenfalls das Vorbereiten der Shape-Animation umfasst (Kapitel 5).

Auch bei der Entwicklung einer Benutzeroberfläche kann entweder auf einer bereits entwickelten Grundlage aufgesetzt werden oder eine Oberfläche von Grund auf neu programmiert werden (Kapitel 6).

Die eigentliche Arbeit der Animation kann nun beginnen. In Kapitel 7, „Die Anwendung in der Praxis“, wird der Umgang mit speziell dem im Rahmen dieser Arbeit entworfenem Rig näher erläutert.

Im nun folgenden Kapitel „Regeln für eine realistische Animation” werden zunächst einige wichtige Merkregeln aus dem Bereich der Animation erläutert, die beim Verständnis der Anforderungen an eine Charaktersteuerung vonnöten sind.

2.5 Regeln für eine realistische Animation

2.5.1 Animation als Kunstform

Natürlich kann der Vorgang der eigentlichen Animation nicht in Gänze im Rahmen dieser Arbeit erläutert werden, wenn er überhaupt schriftlich erläutert werden kann. Für diese Art der künstlerischen Arbeit ist jahrelange praktische Erfahrung nötig. Einige wichtige generelle Regeln können jedoch zum weiteren Verständnis beitragen.

Es ist eine jahrelange Übung erforderlich ist, um in dem Gebiet der Animation ein überzeugendes Ergebnis zu erreichen. Neben einem Verständnis für die natürlichen Abläufe in der Natur und künstlerischem Verständnis ist auch eine große Beobachtungsgabe

erforderlich. So ist es sicher kein Zufall, dass einer der Mitbegründer der Animation als Form des Massenentertainments, Walt Disney, auf einer Farm aufwuchs, wo er die Natur ganz hautnah und unmittelbar beobachten konnte. Vielleicht ist das einer der Gründe für seine lebensnahen Animationen (Thomas1981).

Natürlich ist Animation mehr als nur eine Kopie der Natur. Komik zum Beispiel ist in Animationen in Kombination mit Karikatur und Übertreibung zu erreichen. Es ist allerdings wichtig zu verstehen, dass etwas nur karikiert werden kann, wenn es vorher komplett verstanden wurde.

Dennoch gibt es einige Grundregeln, die für die Animation lebendiger Objekte hilfreich sind. Im Folgenden sollen die wichtigsten davon kurz erläutert werden. Genau wie Zeichentrick-Animationen erst im Laufe vieler Jahre das hohe Niveau heutiger Produktionen erreicht haben, sind auch diese Regeln im Laufe vieler Jahre entwickelt und verfeinert worden Auch bei den Disney-Animationen wurde anfangs kein großer Wert auf Dinge wie Anatomie oder Konzepte wie Gewichtsverlagerung und Ähnliches gelegt (Thomas1981, Abb. 2.8).

Abb. 2.8: Animation aus dem Disney-Film "Steamboat Willie" von 1928. Zu diesen Zeiten wurde noch kaum auf etwa eine korrekte Anatomie Wert gelegt (Quelle: Thomas1981, S. 35)

2.5.2 Physikalisch begründete Regeln

Eine der physikalischen Gesetze, die beim Animieren nie aus den Augen verloren werden sollte, ist das Massenträgheitsgesetz. So wirkt sich etwa die Bewegung des Schwanzes eines Tieres nicht sofort über den kompletten Schwanz aus, sie nimmt stattdessen am Anfangspunkt des Schwanzes ihren Anfang und setzt sich dann nach und nach durch den ganzen Schwanz fort (Arima2001).

Eng mit der Massenträgheit ist auch das Gesetz des Stauchens und Quetschens verbunden. Ein auf einem festen Körper aufprallender Ball wirkt, wenn er sich dabei etwas verformt, gleich völlig anders als ein starrer Ball. Hierbei muss allerdings darauf geachtet werden, dass die Volumenveränderung des Körpers in ihren physikalisch gesetzten Grenzen bleibt.

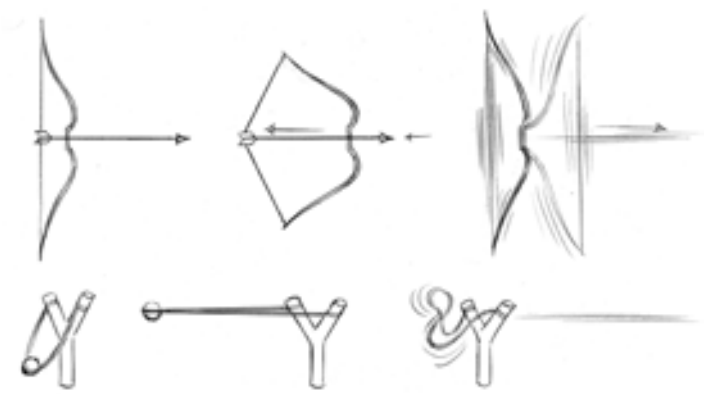

Abb. 2.9: Das Prinzip der Antizipation an anschaulichen Beispielen (Quelle: Williams2001, S. 274)

Auch das Prinzip der Antizipation, also sozusagen der „Vorwegnahme“ einer Bewegung, ist physikalisch begründet (Abb. 2.9). Es umfasst etwa das Spannen eines Bogens vor dem Abschuss oder das in die Knie gehen vor einem Sprung und ist einer der wichtigsten Bestandteile von überzeugend wirkenden natürlichen Bewegungen (Williams2001).

Quasi das Gegenstück zur Antizipation ist die “Counteraction“ oder „Gegenbewegung“. Eine Gegenbewegung tritt nach der Hauptanimation auf und ist sozusagen das Nachschwingen der ursprünglichen Bewegung. Beispiele sind etwa das Nachschwingen eines Schwanzes oder von Haaren (Williams2001).

Ein weiterer Punkt, der, wenn er beim Animieren beachtet wird, für Natürlichkeit sorgt ist die Gewichtsverlagerung. Jedes Objekt, das auf Beinen oder Ähnlichem balanciert, führt ständig eine Gewichtsverlagerung aus. Beim Animieren dieser Gewichtsverlagerung ist allerdings zu beachten, dass man häufig keine gute Gewichtung erhält, wenn man die Natur eins-zu-eins kopiert. Unter Umständen sollten die sogenannten Ups und Downs, also die Extrema der Gewichtsverlagerungen, sogar noch akzentuiert werden (Williams2001).

2.5.3 Richtiges Timing

Das richtige Timing ist für eine gute Animation entscheidend. Ein Fehler, der besonders bei der Computeranimation von Anfängern sehr häufig gemacht wird, sind nichtüberlappende Aktionen. Mit anderen Worten bedeutet das, dass mehrere Bewegungen die zeitlich exakt simultan laufen, unrealistisch wirken (Arima2001).

Dieser Fehler kommt deshalb in der Computeranimation so häufig vor, weil sie dazu verleitet zu einzelnen Frames sehr viele Keys an verschiedenen Parametern zu setzen, anstatt unterschiedliche Animationen unabhängig voneinander zu animieren. In der Praxis führen überlappende Aktionen also zu mehr Keyframes, was auch wieder die Bedeutung eines guten Workflows bei der Low-Level-Animation unterstreicht und eine effizient zu bedienende Charaktersteuerung fordert (Williams2001).

Bei dieser Gelegenheit sei auch darauf hingewiesen werden, dass gutes Timing auch die Pausen umfasst. Ein Charakter sollte unter keinen Umständen ununterbrochen in Bewegung sein. Sinnvoll eingesetzte, nicht regelmäßige Pausen gehören ebenso wie die Bewegung selbst zu einer natürlichen Animation (Arima2001).

Das richtige Timing ist auch eine der Eigenschaften, die Figuren voneinander unterscheidbar macht. Viele reale Schauspieler bringen ihr eigenes Timing mit, das ihrem Spiel eine Wiedererkennbarkeit gibt. Genau diese Wiedererkennbarkeit der Bewegungen sollte auch einem virtuellen Charakter mitgegeben werden.

2.5.4 Durch den Arbeitsablauf begründete Regeln

Es wird, wenn der Arbeitsablauf beim Animieren untersucht wird, zwischen zwei verschiedenen Methoden unterschieden, die natürlich auch je nach persönlichem Geschmack kombiniert beziehungsweise variiert werden können. Diese beiden Methoden werden im Folgenden als der „Pose-to-Pose“- und der „Straight-Ahead“-Ansatz bezeichnet (Williams2001, Rossano2002, Thomas1981).

Beim „Pose-to-Pose“-Ansatz werden die Frames in der Reihenfolge ihrer Bedeutung für die Animation animiert. Zunächst werden also die allerwichtigsten Posen eines Charakters definiert, meistens sind dies auch die Extrempunkte einer Bewegung. Anschließend werden die etwas weniger wichtigen Momente einer Animation angegangen, oft handelt es sich hier um die sogenannten „Breakdowns“ einer Bewegung, also die Momente, die genau zwischen zwei Extrema liegen. Erst zum Schluss werden dann die restlichen Frames gezeichnet oder vom Computer interpoliert.

Diese Methode hat den Vorteil, dass der zeitliche Ablauf einer Animation während der Arbeit sehr gut zu verfolgen und kontrollieren ist, die Pose-To-Pose-Methode ist in den meisten Fällen auch sehr wirtschaftlich. Die entstandenen Animationen können aber unter Umständen an einem fehlenden „Fluss“ leiden (Williams2001).

Ein guter Tipp beim Arbeiten mit der Pose-To-Pose-Technik ist das sogenannte „Silhouetting“. Dabei werden die Posen einzeln als Silhouetten betrachtet. Ziel ist, diese Posen derart zu definieren, dass sie auch in der Darstellung als Silhouetten klar zu erkennen sind. Hintergrund dieser Technik ist, dass leicht lesbare Silhouetten zu den gewünschten, leicht erkennbaren, Animationen führen (Arima2001).

Der „Straight Ahead“-Ansatz dagegen sieht ein sequentielles Bearbeiten der Animation vor. Die Planung des Ablaufes fällt dabei schwerer, die Animationen wirken aber häufig spontaner und flüssiger (Williams2001).

Bei der Computergrafik besteht, im Gegensatz zur gezeichneten Animation, auch die Möglichkeit, zunächst die wichtigsten Animationen, die Primäranimationen, zu animieren und anschließend in einem zweiten oder dritten Durchgang die sekundären beziehungsweise tertiären Animationen. Dieses Vorgehen erleichtert die Zuordnung der Sekundäranimationen zu den Primäraktionen und kann daher auch zu überzeugenderen Animationen führen.

Es gibt nicht den einzigen Königsweg, was den Arbeitsablauf beim Animieren angeht, wie so oft liegt die Wahrheit in der Mitte. Je nach den speziellen Erfordernissen sollte ein Mittelweg zwischen den beschriebenen Methoden gefunden werden. So könnte man zunächst die Schlüsselmomente der Animation zeichnen und anschließend die Primäranimation dazwischen sequentiell bearbeiten. In einem zweiten Durchgang würden dann die Sekundäranimationen hinzugefügt. Dieses Vorgehen dürfte auch das in der Praxis am Häufigsten angewendete sein (Williams2001).

2.6 Zusammenfassung

Das durchaus komplexe Gebiet der Grafischen Datenverarbeitung hat, wie eigentlich alle wissenschaftlichen Fachgebiete, seine eigene Terminologie, die dem Leser im ersten Teil dieses Kapitels nahegebracht wurde. Kenntnis über die Fachbegriffe ist für das weitere Verständnis unbedingt vonnöten, wenn es im Folgenden mehr an die Arbeit in der Praxis geht.

Im zweiten Teil des Kapitels wurde die Charakteranimation aus der Sicht des Animators betrachtet. Der Vorbereitung des Animierens, dem diese Untersuchung hauptsächlich gewidmet ist, muss eine gute Kenntnis des Animationsprozesses selbst vorangehen, will sie in der Praxis effektive und qualitativ gute Ergebnisse liefern.

3 Erstellen des Charakters

3.1 Einleitung

Eine der wichtigsten Lehren des Projektmanagements besagt, dass Fehler, die zu Beginn eines Projektes auftreten und zu spät korrigiert werden, im Allgemeinen wesentlich schwerer wiegen als Fehler, die gegen Ende des Projektes auftreten. Fehler, die in der Konzeptphase gemacht werden, können später häufig nur schwer korrigiert werden. Ein effektives Arbeiten erfordert also gute Planung.

In der Charakteranimation bedeutet das natürlich, dass das Konzept für den Charakter sowie das Planen der späteren Animation besonders entscheidend für den Projektverlauf sind. In Kapitel 3.2 werden deshalb zu den Themen Storyentwicklung, Charakterdesign und Animationsstudien verschiedene Techniken erläutert und Hinweise gegeben.

Im darauffolgenden Kapitel 3.3 wird am Beispiel des Teckels, der Bestandteil des Praxis-Projektes zu dieser Ingenieursarbeit ist, die Modellierungsphase erläutert und dabei besonderer Wert auf Punkte gelegt, die für ein späteres Enveloping des zu modellierenden Objektes wichtig sind.

Nach dem gleichen Schema wird in Kapitel 3.4 das Arbeiten mit Shadern und Texturen im Hinblick auf eine spätere Animation beleuchtet. Besonders, wenn ein relativ realistischer Look erreicht werden soll, ist für den Gesamteindruck der fertigen Animation der Bereich des Renderings und der Beleuchtung genauso wichtig wie eine gute und realistische Animation. Im Rahmen der Vorbereitung zu „Jurassic Parc“, dem ersten großen Kinofilm, in welchem zum ersten Mal in großem Stil realistisch wirkende Kreaturen computergeneriert wurden, wurde beispielsweise aufgrund unbefriedigend wirkender Animationen zunächst längere Zeit an diesen Animationen gearbeitet, bevor die Entwickler den Grund für den mangelnden Realismus im Bereich des „Rendering & Lighting“ erkannten (Quelle: Vortrag „Terminator 3“ von Pablo Helman, VFX-Supervisor, Industrial Light & Magic im Rahmen der eDIT|VES 2003, 28.09.2003).

3.2 Entwurf des Charakters

3.2.1 Storyentwicklung

Frank Thomas und Ollie Johnston, zwei der wichtigsten Animatoren in der Mitte des 20. Jahrhunderts, sahen in den schlechten Storys und stereotypen Charakteren der Zeichentrickanimationen der 20er und 30er Jahre, die darin begründet lag, dass Zeichentrickanimationen damals häufig nur für kurze Werbefilme und slapstickartige Kurzfilme verwendet wurden, den Hauptgrund für die häufig unnatürlich und staksig wirkenden Animationen dieser Zeit. Eine gute Story und ein überzeugender Charakter können

demnach also auch für die Qualität der Animationen selbst von Bedeutung sein (Thomas1981).

Von Projekt zu Projekt unterschiedlich wird im Allgemeinen die Reihenfolge von Storyentwicklung und Charakterdesign gehandhabt, manchmal stehen ein oder mehrere Charaktere schon relativ fest, während an der Story noch gearbeitet wird, manchmal ist es umgekehrt. Der am Häufigsten vorkommende Fall dürfte aber wohl sein, dass schon relativ früh beides parallel abläuft.

In dem konkreten Fall des zu dieser Arbeit gehörenden Praxisprojektes, der Arbeit an dem Teckel „Willy“, stand zunächst das Charakterdesign im Vordergrund, da dieser Charakter von Anfang an auf die Verwendung in einer möglichen Serie von Kurzfilmen ausgelegt war, wobei nur die Story des Pilotfilmes schon relativ klar war.

Dass auch in solchen Fällen schon ein Blick auf die möglichen Storyverläufe gelegt werden sollte, zeigt sehr gut das folgende Beispiel: In einer Szene des geplanten Pilotfilms wird „Willy“ bei ersten Flugversuchen mit einem Modellflugzeug von anderen Hunden beobachtet (Abb. 3.1). Durch die Planung dieser Szene zu diesem frühen Zeitpunkt war klar, dass neben Willy auch andere Hunde zu sehen sein würden. Dies ist einer der Gründe, warum insbesondere auch auf die leichte Wiederverwendbarkeit des im Rahmen der Arbeit erstellten Control-Rigs bei anderen Hundearten geachtet wurde.

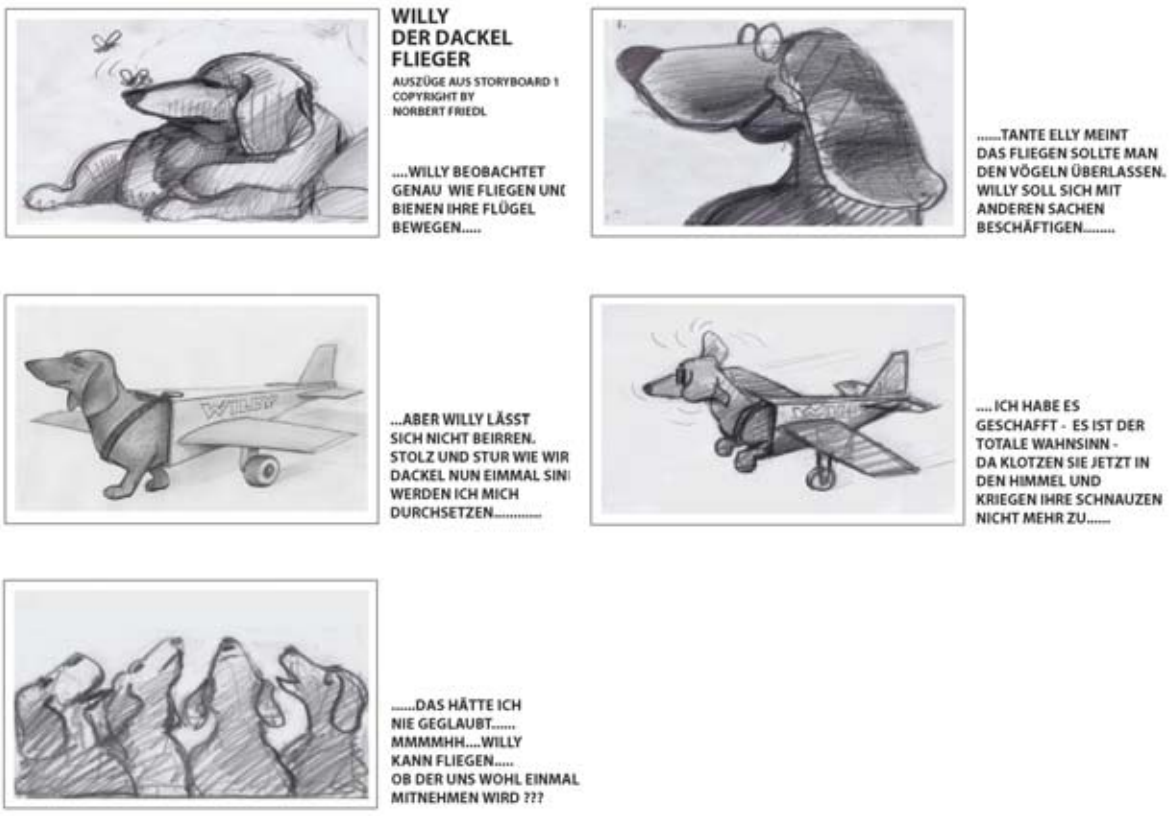

Abb. 3.1: Auszug aus dem Storyboard für den möglichen Pilotfilm mit der Hauptfigur „Willy“ und anderen Hunden (Copyright: Norbert Friedl)

Im oben angesprochenen Fall wurde zur Verdeutlichung und dem Entwurf einer Geschichte ein Storyboard verwendet, also eine sequentielle Aneinanderreihung verschiedener Skizzen, die unter anderem die Schlüsselmomente der Story, die vorherrschenden Positionen und Gesten der Figuren, die Bildgestaltung, Kamerabewegung und Ähnliches enthalten können. Zu jeder Skizze eines Storyboards kann auch eine mehr oder weniger umfangreiche textliche Beschreibung gehören (Maraffi2001).

Ein Storyboard ist sicher die bekannteste Möglichkeit zur Planung einer Geschichte, es gibt jedoch noch zahlreiche weitere Methoden. Das Zeichnen von Thumbnails etwa, also Zeichnungen, die noch etwas stärker abstrahiert sind als bei einem Storyboard, können gerade in einer sehr frühen Phase eine wirkungsvolle Alternative sein. Thumbnails müssen nicht zwangsläufig eine so komplette Beschreibung der Szene liefern wie ein Storyboard, auch kommen Thumbnails ohne eine textliche Beschreibung der Szenen aus (Maraffi2001).

Beim Planen einer Story für Filme werden häufig in einer etwas späteren Phase die Storyboards oder Thumbnails in sogenannte (2D-)Animatics umgesetzt, das heißt sie werden in der richtigen Reihenfolge und mit dem richtigen Timing hintereinander als Film abgespielt, unter Umständen wird auch schon eine vorläufige Tonspur angelegt. Ein Animatic dient damit zum Überprüfen der Sequenz von Szenen auf ihre Wirkung im Gesamtkontext und das richtige Timing (Maraffi2001, Williams2001).

Speziell in der 3D-Computergrafik wird nicht selten noch ein Schritt weitergegangen und ein sogenanntes 3D-Animatic erstellt. Dies ist eine stark vereinfachte Szene, die ohne Details der Umgebung oder der Charaktere auskommt und in der häufig nur eine grobe Animation der Figuren oder Objekte vorhanden ist. Dadurch, dass die Kamerabewegung und die Szenenfolge aber schon gut sichtbar sein können, hilft ein 3D-Animatic außerordentlich, die spätere Sequenz auf Ihre Wirkung hin zu überprüfen. Wichtig ist dabei, dass die Objekt-Hierachie der Modelle bereits korrekt ist, da sich aus dem 3D-Animatic die endgültige Animation entwickeln kann, viele der Animationen können dann eventuell beibehalten werden (Maraffi2001).

3.2.2 Charakterdesign

Wie in jeder anderen Form des Geschichtenerzählens auch, wird die Story eines Trickfilmes alleine den Zuschauer nicht bewegen, wenn die Charaktere flach und stereotyp sind. Die Bedeutung eines Charakters für eine Geschichte lässt sich sehr schön an einem simplen Beispiel zeigen, wie es einmal von Marcel Marceau aufgezeigt wurde (Thomas1981):

> „If a dignified man slips on a banana peel, it is funny. If it happens to a man who is down and out, it is not."

Dass Charakterdesign an sich alles andere als trivial ist, kann man sehr gut beobachten, wenn man die Entwicklung der wohl weltweit erfolgreichsten animierten Figur, Disney's Mickey Mouse, verfolgt. Sie wurde noch Jahrzehnte nach Ihrem ersten Auftreten in nicht unbedeuteten Ausmaß mit dem Ziel verändert, Emotionen besser veranschaulichen zu können (Thomas1981).

Es gibt verschiedene Möglichkeiten zum Entwickeln eines starken Charakters. Zu Anfang wird meistens mit einem Brainstorming begonnen, entweder visuell oder rein verbal, das dann meistens zu einer sogenannten Charakterbiografie führt, in der schriftlich die wichtigsten Eigenschaften des Charakters festgehalten werden. Wichtig ist dabei zu vermeiden, sich nur auf Äußerlichkeiten zu konzentrieren, charakterliche Tiefe einer Figur wird es später auch den für das Modeling und die Animation zuständigen Personen erleichtern, diese Figur mit Leben zu füllen. Auf der anderen Seite muss natürlich auch nach Bedeutung der Figur für die Story gewichtet werden, nicht jede einzelne Figur einer Massenszene muss genauso gut definiert werden wie eine Hauptfigur.

T. Helzle und C. Desse stellen in Ihrem Artikel (Helzle2003) eine kurze, relativ schnell durchführbare Checkliste vor, die hier helfen kann, einem Charakter Tiefe zu verleihen (Tabelle 3.1).

Bezeichnung	Mögliche Bestandteile
Art, Typ, Spezies	selbsterklärend
Morphologische Spezifikationen	Alter, Größe, Geschlecht, etc.
Lebensraum	selbsterklärend
Platz in der Nahrungskette und Überlebensstrategien	Jäger oder Gejagter, Feinde, Nahrung, etc.
Soziales Umfeld und Platz in der Gesellschaft	Finanzielle Situation, Familienverhältnisse
Psychogramm	Vorlieben, Abneigungen
Synthese und kurze Zusammenfassung	Selbsterklärend

Tabelle 3.1: Checkliste für den Charakterentwurf (Quelle: Helzle2003)

Im nächsten Schritt sollte der Figur ein Aussehen verliehen werden. Hier hilft oft nur ein visuelles Brainstorming, unter Umständen sind zahlreiche Rohzeichnungen nötig, um das endgültige Aussehen einer Figur zu erhalten. Hier ist es oft von großem Vorteil, wenn im vorhinein Referenzmaterial gesammelt wurde, etwa Fotografien von Tieren, Entwürfe anderer Künstler, Romane oder Filme.

In diesem Stadium der Planung ist es auch vonnöten, Überlegungen bezüglich des künstlerischen Stils der späteren Sequenz anzustrengen. Durch die Möglichkeiten der Computergrafik sind beinahe unzählige verschieden Stilistiken für Trickfilme entstanden, vom sogenannten „Toy-Look" etwa eines Filmes wie „Toy-Story" über einen Komik-Look bis zu einem beinahe fotorealistischen Aussehen etwa von Filmen wie „Final Fantasy". Dieser Look des Gesamtproduktes beeinflusst natürlich auch das Aussehen der einzelnen Figuren und umgekehrt (Maraffi2001).

3.2.3 Animationsstudien

Bereits bei der Konzeption des Charakters sollte mit der Planung der späteren Bewegungen begonnen werden, um sicherzugehen, dass diese Bewegungen mit der Figur, so wie sie konzipiert ist, auch durchführbar sind und glaubhaft wirken.

Auch für technische Überlegungen bezüglich des Aufbaus des Rigs ist es von Vorteil, die wichtigsten Bewegungen der Figur absehen zu können. So lässt sich dann beurteilen, welche Körperteile in welcher Form animierbar sein müssen und was vom Animator direkt gesteuert werden muss oder was automatisiert werden kann.

Zu diesem Zweck hat es sich bewährt, sogenannte „Model Sheets“ anzufertigen (Abb. 3.2). Diese wurden ursprünglich angefertigt, um in der Zeichentrickanimation, wo oft viele verschiedene Zeichner und Animatoren an einem Charakter arbeiten, das Aussehen desselbigen zu standardisieren. Auf ihnen sollte das Modell in der Standardposition aus mehreren Ansichten heraus dargestellt sein, sowie zusätzlich einige für den Charakter typische Schlüsselpositionen. Alle Zeichnungen sollten maßstabsgetreu angefertigt werden, dann eignen sie sich auch als Vorlage zum Modellieren (Thomas1981, Maraffi2001).

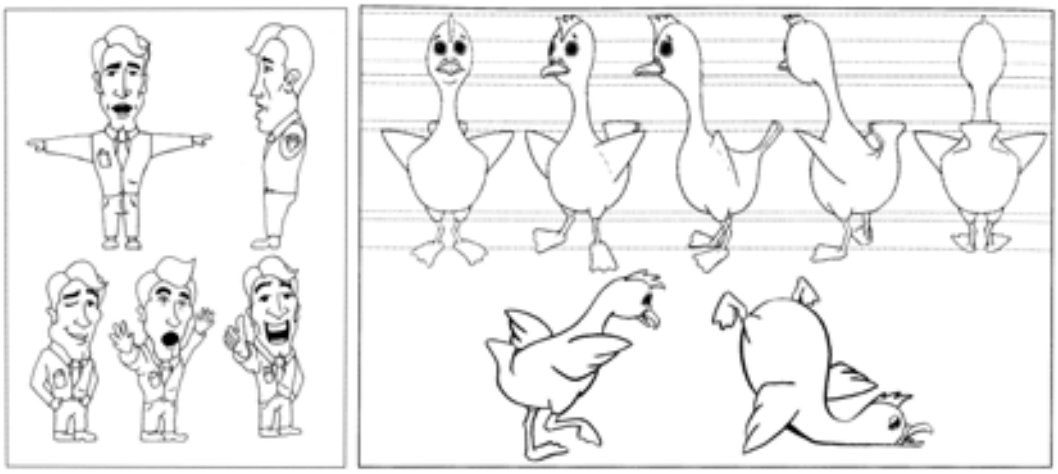

Abb. 3.2: Beispiele für Model Sheets (Quelle: Maraffi2001, S. 41,42; Zeichnungen rechts: Malin Tvedt)

Auch wenn es etwas unkonventionell erscheint, wird von vielen Animatoren empfohlen, dass der Animator die zu animierende Figur schauspielerisch nachvollzieht und sich dadurch bestmöglich in die Figur hineinversetzt. So wurden schon Animatoren dabei beobachtet, wie sie galoppierend oder trabend ihr Arbeitszimmer durchquerten, um den Bewegungsapparat eines Pferdes besser nachvollziehen zu können (Williams2001, Thomas1981).

Auch für den Film „The Hulk“, in dem die Hauptfigur computergeneriert ist, stellten die Animatoren selber verschiedene Bewegungen der Figur nach, während sie außerdem dabei gefilmt wurden (Quelle: Vortrag „The Hulk“ von Gerald Gutschmidt, Computer Graphics Super-visor, Industrial Light & Magic im Rahmen der eDIT|VES 2003, 28.09.2003).

Außerordentlich sinnvoll ist auch im Zusammenhang mit Bewegungsstudien das Sammeln von Referenzmaterialien bereits zu diesem Zeitpunkt zu beginnen. Meistens werden diese Referenzen nur als lose Anhaltspunkte verwendet, es gibt aber auch Fälle, in denen Realaufnahmen als genaue Vorgabe selbst für Details der Animationen verwendet werden. Die Realaufnahmen können hierfür während der Animation mithilfe einer, „Rotoscopy“ genannten, Technik „hinter“ den computergenerierten Charakteren dargestellt werden. Die Rotoscopy wird besonders gerne verwendet, wenn eine Überblendung zwischen computergenerierten Charakteren und Realaufnahmen erfolgen soll, wie etwa im Falle des Filmes „Fluch der Karibik“ (Quelle: Vortrag „Pirates of the Caribean“ von Hal T. Hickel, Animation Director, Industrial Light & Magic im Rahmen der eDIT|VES 2003, 28.09.2003).

Abb. 3.3: Im Buch von R. Williams findet sich beispielsweise auch diese Darstellung der Rückgratbewegung eines Hundes beim Laufen (Quelle: Williams2001, S. 331)

Auch in diversen Lehrbüchern zum Thema Zeichnen oder Animieren finden sich Anleitungen zum Zeichnen von Bewegungsabläufen, die natürlich genauso hilfreich auch für das Animieren mithilfe des Computers sind. (Abb. 3.3). Zur Modellierung und Animation von Muskeln, etwa für Gesichtsanimationen, sind Standardwerke der Zeichenkunst gut dienlich (etwa Faigin1990, Hogarth1990).

Im Falle des Teckels „Willy” haben sich sowohl an einem echten Teckel gedrehte Filmaufnahmen, als auch Fotoserien aus der Sammlung von Bewegungsstudien von Eadweard Muybridge aus dem Jahre 1887 als Referenzen geeignet (Abb. 3.4). Obwohl diese Abbildungen schon über 100 Jahre alt sind, werden sie noch immer sehr gerne zu Referenzzwecken beim Animieren eingesetzt (Muybridge1957, Muybridge1979).

Beim Sammeln der Referenzen kommt eine besondere Bedeutung den wohl am Häufigsten vorkommenden Bewegungen zu, dem Laufen, Gehen und Rennen. Das normale Gehen ist außerdem auch eine der Bewegungen, die einer Figur, egal ob menschlich oder nicht, die größte Individualität mitgibt. Dies wiederum macht das Gehen auch zu einer der am schwersten reproduzierbaren Bewegungen (Williams2001).

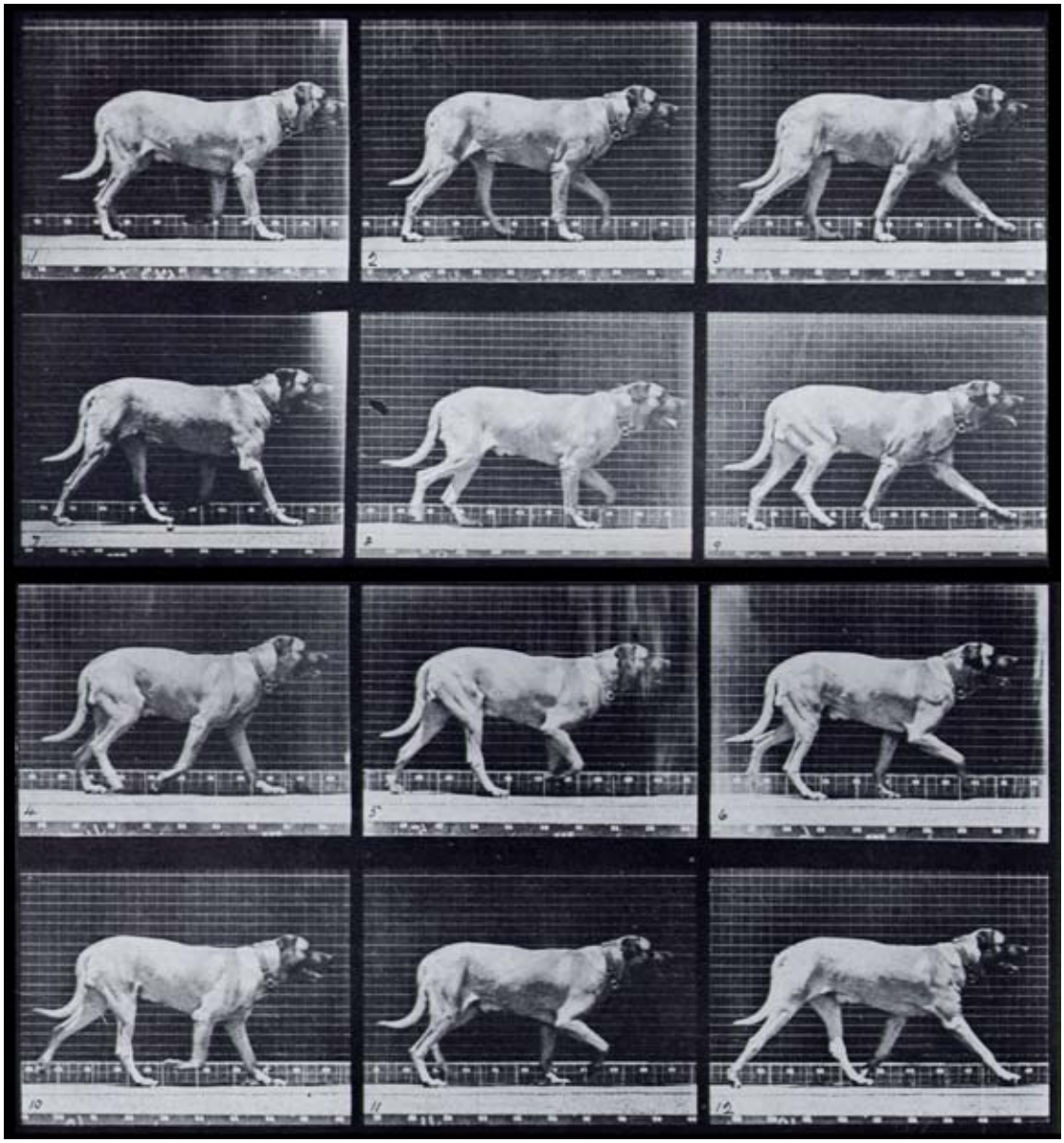

Abb. 3.4: Obwohl schon über 100 Jahre alt, sind die Fotoserien von Muybridge immer noch häufig eine gute Quelle für Animationsstudien von lebendigen Tieren (Muybridge1957, Tafel 113)

3.3 Modellierung

3.3.1 Modellierungstechniken

Schon seit geraumer Zeit ist in der Grafischen Datenverarbeitung zu beobachten, dass vor allem zwei Modellierungstechniken bevorzugt verwendet werden: Die sogenannte NURBS-Modellierung und die Polygon-Modellierung (Abb. 3.5). Die Polygon-Modellierung wurde in den letzten Jahren mit der sogenannten Subdivisions-Modellierung erweitert, die in dieser Arbeit, wie in den größten Teilen der Fachliteratur, als Ergänzung betrachtet wird, nicht als neue Technik.

In der Polygon-Modellierung werden die modellierten Objekte als sogenannte „Polygon Meshes“ oder auch nur Polygon-Objekte bezeichnet. Sie bestehen aus an ihren Ecken („Vertices“) und Kanten („Edges“) durch gerade Linien miteinander verbundenen Polygonen. Polygone wiederum sind geschlossene, flache Objekte, die aus den oben besprochenen Kanten und Ecken bestehen.

Eine erweiterte Form der Polygon-Modellierung, die sogenannte Subdivisions-Modellierung, die erst seit wenigen Jahren in professionellen Modellierungstools zu finden ist, hat sich seitdem ernorm verbreitet. Subdivisions-Objekte (oder „Subdivision Surfaces“) verwenden ein Polygon-Objekt als Hüllobjekt, welches ein Objekt mit höherer Geometriedichte steuert. Dadurch entstehen ohne großen zusätzlichen Aufwand aus Polygon-Objekten wesentlich glatter wirkende Modelle.

Die Algorithmen, welche die Unterteilung des Hüllobjektes steuern, werden „Subdivision Rules“ genannt, in Softimage|XSI sind gegenwärtig zwei verschiedene Methoden integriert, der sogenannte „Catmull-Clark“- und der „Doo-Sabin“-Algorithmus. In den meisten Fällen ergeben die beiden Algorithmen sehr ähnliche Ergebnisse, der Catmull-Clark-Algorithmus ergibt tendenziell etwas glättere Ergebnisse, während der Doo-Sabin-Algorithmus Ergebnisse produziert, die stärker den Formen des Hüll-Meshes folgen.

Im Bereich der künstlerischen Modellierung werden Subdivisions-Objekte inzwischen immer beliebter, gerade bei der Modellierung organischer Formen haben sie große Vorteile gegenüber der reinen Polygon-Modellierung. Im Folgenden wird deshalb mit dem Begriff der „Polygon-Modellierung“ auch das Modellieren mit Subdivisions-Objekten gemeint.

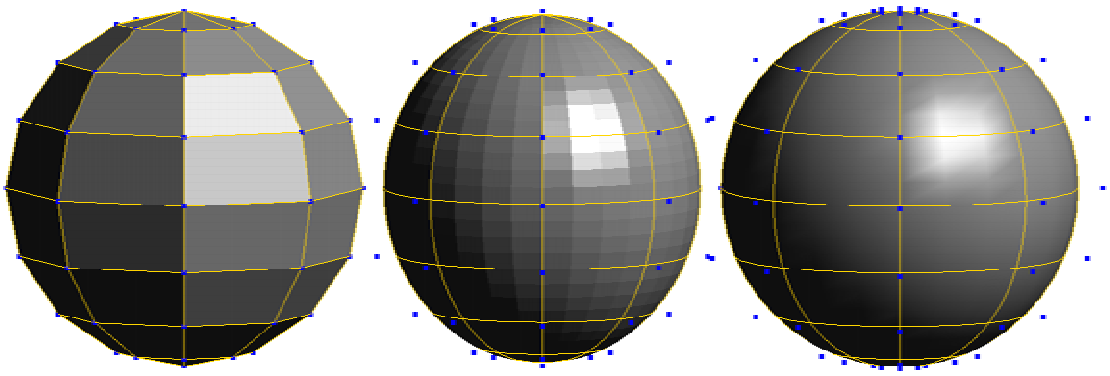

Abb. 3.5: Vergleich der verschiedenen Modellierungstechniken (Links: reine Polygon-Modellierung, Mitte: Subdivisions-Modellierung, Rechts: NURBS-Modellierung)

Die Modellierung mit der NURBS-Technik beruht dagegen auf einer völlig anderen Technik: NURBS steht für „Non-Uniform Rational B-Splines“, es sind, vereinfacht gesagt, mathematische Beschreibungen von kubischen Kurven. Aus diesen Kurven können dann auch Oberflächen gebildet werden. Diese Objekte werden dann NURBS-

Surfaces oder auch einfach nur Surfaces genannt, sie werden als eine Art Flickenteppich aus den kleineren, rechteckigen sogenannten Patches zusammengesetzt (Watt2002).

Die NURBS-Technik wurde ursprünglich für CAD- und CAE-Anwendungen entwickelt, es ist eine wesentlich technischere und auch genauere Herangehensweise an die Modellierung von Objekten als die Polygon-Modellierung. Der große Vorteil der NURBS-Technik ist, dass automatisch glatte und damit organisch wirkende Flächen erstellt werden. Da NURBS-Surfaces mathematisch beschrieben werden, bringt eine Annäherung keine Vergröberung der Struktur mit sich (Koenigsmarck2000).

Auch NURBS-Surfaces werden zur Rechenzeit wieder in Polygone umgerechnet (Tesselation), allerdings mit flexibler Auflösung, die etwa von der Entfernung zur Kamera abhängig gemacht werden kann. Dies führt im gerenderten Bild zu einer beliebig großen Kantenglätte.

Bei der Überlegung, welche Modellierungs-Technik in einem konkreten Anwendungsfall verwendet werden sollte, sind eine Reihe von Überlegungen anzustellen:

- NURBS-Surfaces benötigen weniger Punkte, um glatte Oberflächen darzustellen, während die Modellierung von Details mit Polygon-Objekten leichter fällt. Die neuen Möglichkeiten der Subdivisions-Technik machen allerdings viele früheren Vorteile der NURBS-Surfaces gegenüber der Polygon-Modellierung wieder wett, es können nun auch mit Polygonen relativ leicht glatte Oberflächen erstellt werden.
- Es gibt verschiedene Operatoren, die sehr elegant aus Kurven NURBS-Surfaces erstellen können, etwa durch das Rotieren einer Kurve um eine andere. Will man ein ähnliches Ergebnis als Polygon-Objekt erreichen, muss man das Surface-Objekt zunächst in ein Polygon-Objekt konvertieren.
- Bei manchen Softwarepaketen ist die Texturierung, konkreter das Zuweisen von Texturkoordinaten, mit NURBS-Surfaces wesentlich einfacher, weil diese von sich aus implizite UV-Koordinaten mitbringen. Bei Polygon-Objekten müssen dagegen zunächst explizite UV-Koordinaten erstellt werden, was Softimage|XSI allerdings gut leistet (Birn2001).
- Die Topologie von Polygon-Objekten kann wesentlich flexibler gestaltet werden als die von NURBS-Surfaces, die immer aus Patches zusammengesetzt werden müssen. Dies führt zu einer eher künstlerischen Arbeitsweise, die es erlaubt, ähnlich wie etwa beim Töpfern, mit einer groben Form anzufangen und nach und nach an den Stellen Details hinzuzufügen, an denen sie benötigt werden (der Prozess des sogenannten „gradual Refinement“). Außerdem können Polygon-Objekte leichter miteinander kombiniert werden, etwa durch den „MeshMerge“-Operator oder Boolesche Verknüpfungen.
- Bei Subdivisions-Objekten besteht die Möglichkeit, mit dem Hüll-Objekt zu arbeiten, etwa es zu deformieren, und damit direkt die höher aufgelöste Version zu

steuern. Dies kann natürlich viel Arbeit etwa beim Weighting oder Texturieren sparen, auch während der Modellierung selbst kann effizienter gearbeitet werden.

Ein wenig ist die Entscheidung, welche Technik angewendet werden sollte, natürlich auch eine Frage des persönlichen Geschmackes.

Im Fall des in dieser Arbeit besprochenen Charakters fiel die Wahl auf die Subdivisions-Modellierung. Das wichtigste Argument in diesem Fall war die kreativere Herangehensweise dieser Methode, sowie die Möglichkeit, verschiedene Subdivisions-Levels für die Darstellung zur Anzeige und für das Rendering zu wählen, was den Workflow beschleunigen kann (vergleiche nächstes Kapitel, „Effizienzüberlegungen“).

3.3.2 Effizienzüberlegungen

Es stehen in Softimage|XSI mehrere Methoden zur Erstellung eines Subdivisions-Objekt zur Verfügung. Es kann entweder ein neues Objekt erstellt werden, welches mit dem Hüll-Objekt weiterhin verknüpft ist, es können Teilbereiche eines Polygons unterteilt werden, oder es kann mit Hilfe der „Geometry-Approximation“ eine existierende Hülle verfeinert werden. Dies ist wohl auch die gebräuchlichste Herangehensweise. Hierbei wird für eine gegebenes Polygon-Objekt ein Subdivisions-Level angegeben, je höher dieser Wert, desto häufiger werden die Polygone unterteilt und desto glatter wirkt das Objekt.

Eine Besonderheit dieser Methode ist, dass der Subdivisions-Level separat für die Anzeige im Interface und für das Rendering eingestellt werden kann, was man sehr gut nutzen kann, um die Darstellung des Modells im Interface zu beschleunigen und den Workflow während dem Animieren damit zu verbessern.

Bereits zu diesem Zeitpunkt, unbedingt aber vor dem Texturieren, sollte überlegt werden, auf welchem Subdivisions-Level texturiert wird, auf welchem Level das Enveloping stattfinden soll und auf welchen Level nach dem Enveloping noch zur Darstellung erhöht werden soll. Wenn auf einem höherem Level gearbeitet werden soll, muss ein eigenes Subdivisions-Objekt erstellt werden, während bei der Arbeit mit Geometry-Approximation nur mit der ursprünglichen Hülle gearbeitet werden kann.

Je weniger Polygone für die Hülle benötigt werden, desto besser. Dies hat die Vorteile einer schnelleren Darstellung, eines leichteren Envelopings und vergrößert auch die Übersichtlichkeit beim Modellieren und damit die Möglichkeit zu späteren Änderungen an der Geometrie.

Allerdings ist zu beachten, dass „Weight Map-Painting“ für Envelopes, also das direkte Aufmalen der Gewichtung auf die Geometrie, bei Subdivisions-Objekten, die mithilfe der Geometry Approximation erstellt wurden, nur direkt auf die Hülle funktioniert. Es darf also auch nicht zu grob modelliert werden. Außerdem sollte die Polygondichte an deformierbaren Stellen, etwa. im Gesicht, hoch genug gewählt werden, um die Deformation gut darstellen zu können.

In diesem Fall wurde das Hüllobjekt ausreichend detailliert modelliert, so dass ohne eine Erhöhung des Subdivision-Levels texturiert und an die Deformer gebunden werden kann. Wenn anschließend erst der Subdivision-Level erhöht wird, hat das auch den Vorteil weicherer Übergänge und leichteren Weightings. Aus diesem Grund konnte komplett mit Geometry-Approximation gearbeitet werden, als Subdivisions-Algorithmus wurde der Catmull-Clark-Algorithmus eingesetzt.

3.3.3 Der Modeling-Prozeß

Softimage|XSI unterstützt das Verfahren der sogenannten „Rotoscopy“, es ermöglicht es also, Bilddateien in einen Viewport einzublenden und dadurch ständig eine Referenz des zu modellierenden Objektes zur Verfügung zu haben. Im Falle von real existierenden Tieren empfiehlt es sich natürlich, möglichst hochauflösende Fotos zu verwenden, die ohne eine allzu große perspektivische Verzerrung aufgenommen sein sollten (Abb. 3.6).

Abb. 3.6: Die für die Rotoscopy in der seitlichen Ansicht verwendete Fotografie eines echten Teckels

Als Referenz für das Modellieren von Details lebender Tiere hat es sich bewährt, Screenshots von angefertigtem Videomaterial zu verwenden. Im Vergleich zu der Methode, direkt Fotos zu machen, hat man in diesem Fall eine größere Auswahl von Motiven, besonders, da man während der Fotosession noch nicht unbedingt vorhersagen kann, für welche Stellen man später detaillierteres Anschauungsmaterial benötigt.

Die Modelling-Tools von XSI lassen ein äußerst effizientes Polygon-Modelling zu, die Arbeit wird in diesem Fall durch die Symmetrie des zu modellierenden Objektes weiter erleichtert (Rosa2003).

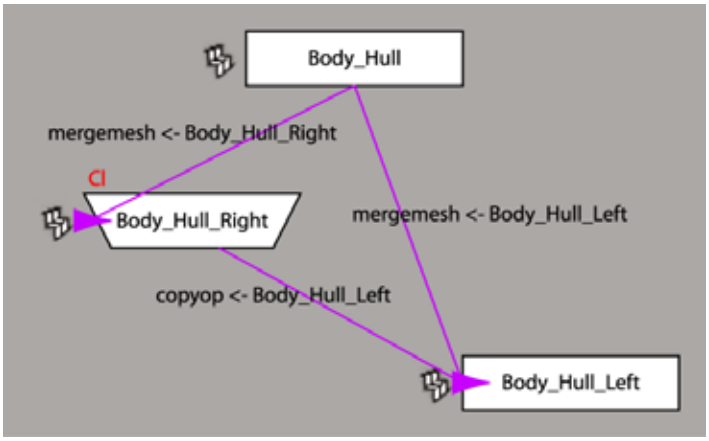

Abb. 3.7: Schematic View der Konstruktionshierarchie des Subdivisions-Objektes

Um diese auszunutzen, wurde nur an der linken Hälfte des Hundes modelliert, während die rechte durch ein Klonen der linken entstand. Anschließend wurden die beiden Hälften durch eine „MeshMerge"-Operation miteinander verschmolzen. Dieses Polygon-Mesh ist dann bereits das fertige Objekt, es kann durch Erhöhen des Subdivisions-Levels bereits jetzt geglättet werden, um eine exaktere Vorschau zu erhalten (Abb. 3.7).

Die Funktionsweise von XSI erlaubt es, diese Konstruktionshierarchie einmal am Anfang des Prozesses aufzubauen und während des ganzen Modellings unangetastet zu lassen. Änderungen an dem ursprünglichen Objekt, der linken Seite des Hundes, werden durch diese Objekthierarchie hindurch übertragen und das Ergebnis kann in Echtzeit auf dem endgültigen Mesh verfolgt werden (Abb. 3.8).

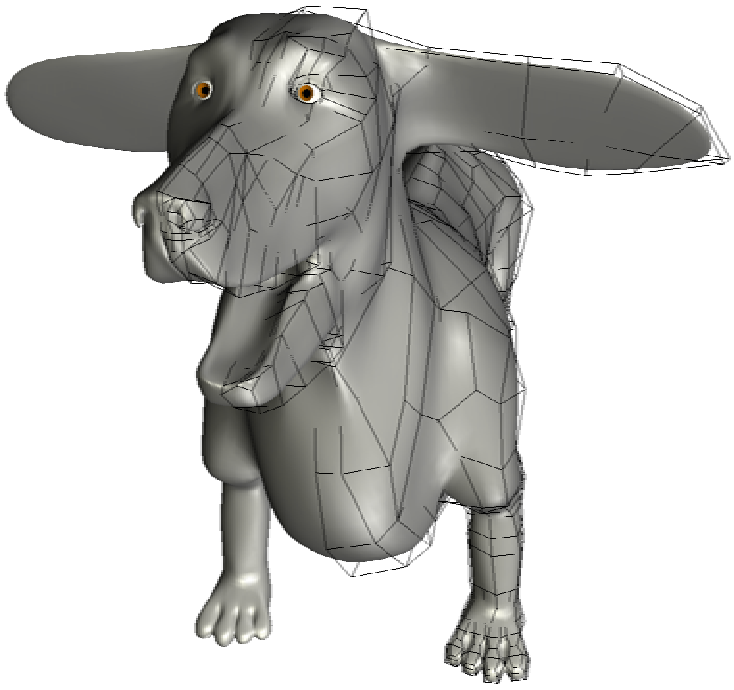

Abb. 3.8: Modellieren einer Hälfte des Objektes bei gleichzeitiger Anzeige des Endergebnisses

Während Torso und Kopf aus einem einzigen Objekt modelliert wurden, wurde die Zunge als eigenständiges Objekt modelliert, um das spätere Enveloping zu vereinfachen. Zähne und Krallen wurden ebenfalls als einzelne Objekte modelliert, um später die Texturierung zu vereinfachen, sie wurden allerdings später gemeinsam mit dem Körper in einem Envelope zusammengefasst.

Wie schon im vorigen Kapitel erwähnt, liegt der große Vorteil der Polygonmodellierung in der eher künstlerischen Herangehensweise an die Modellierung. Die Möglichkeit der Schrittweisen Verfeinerung („gradual Refinement“), also die Möglichkeit einem Objekt nach und nach an den benötigten Stellen Details hinzuzufügen, erinnert von der Arbeitsweise her etwas an das Töpfern und wird von den meisten Leuten als intuitiver als etwa das NURBS-Modeling empfunden (Abb. 3.9).

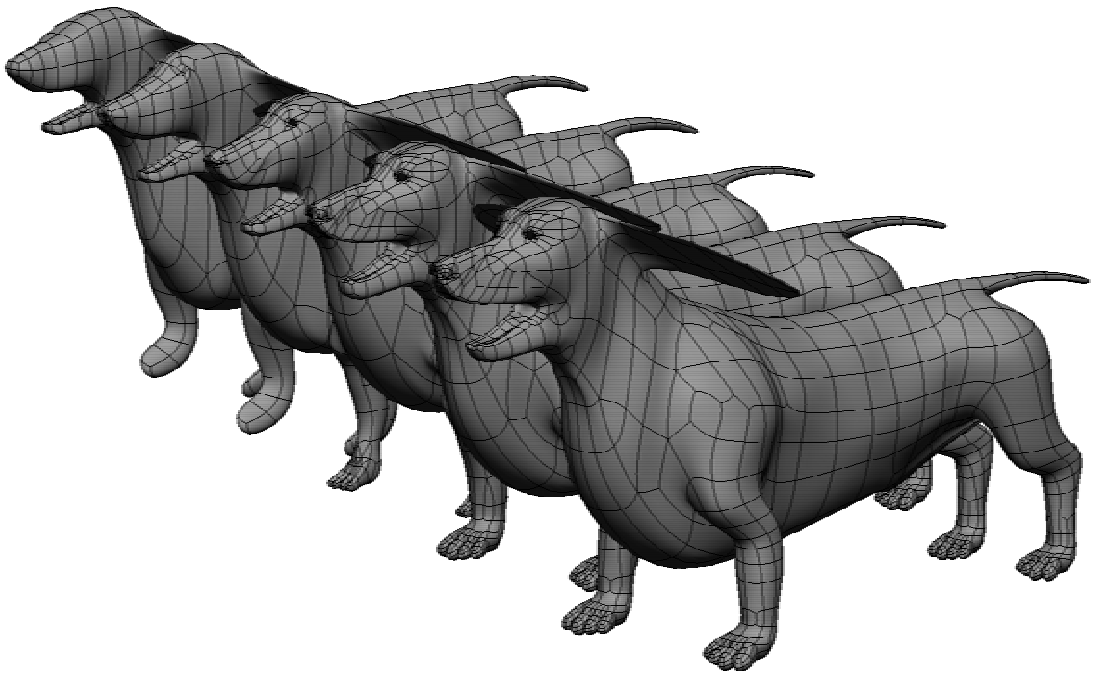

Abb. 3.9: Schrittweise Verfeinerung des Polygon-Objektes

Bei der Modellierung können die umfangreichen Polygon-Modellierungstools von Softimage|XSI voll ausgenutzt werden. Es sollte jedoch darauf geachtet werden, keine sogenannten „N-gons“, also Polygone mit mehr als vier Kanten zu erstellen, da diese bei einer späteren Unterteilung im Rahmen der Subdivisions-Technik zu Tesselierungsfehlern führen können, die besonders bei Animationen störend auffallen können. Am besten geeignet für die Subdivisions-Technik sind Polygon-Objekte die ausschließlich auf „Quads“, also Polygonen mit vier Kanten, beruhen, und damit auch auf Polygone mit drei Kanten verzichten. Ebenfalls von Vorteil sind Konstruktionsweisen, bei denen Punkte an genau vier Kanten liegen (XsiDoc2002, Modeling & Deformations).

Bei der Modellierung ist unbedingt zu beachten, dass Stellen, die später durch die Animation besonders deformiert werden, auch eine entsprechend höhere Geometrieauflösung erhalten. Dazu zählen beispielsweise die Teile des Envelopes, an denen später mal Gelenke liegen werden oder auch Teile des Gesichtes, die für Gesichtsausdrücke deformiert werden sollen. Aus diesem Grund ist es auch von großem Vorteil, wenn bereits im Charakterdesign-Prozess überlegt wurde, welche Emotionen die Figur überhaupt ausdrücken soll. Im speziellen Fall des Teckels „Willy“ war klar, dass ein relativ realis-

tischer Look, der erreicht werden sollte, sich nicht mit übermäßig deutlichen Gesichtsausdrücken verträgt (Rossano2002).

Der Charakter wurde mit abstehenden Extremitäten, wie etwa den Ohren und dem Schwanz, modelliert und dadurch ein späteres Weighting erleichtert. Aus dem Aussehen dieser Pose bei einem menschlichem Charakter, das bei ausgestreckten Armen und Beinen an den Buchstaben „T“ erinnert, leitet sich der für diese Haltung gebräuchliche Name „T-Pose“ ab.

Ebenfalls zur Vorbereitung einer einfachen Animation gehört, eine klare und übersichtliche Struktur der Objekte, die später einmal zum Envelope gehören werden, zu bilden. Besonders hilfreich ist hier das Bilden von Gruppen für jeweils alle Objekte, die später einmal zu einem Envelope gehören werden. Während der Arbeit am Weighting wird unter Umständen eine häufige Auswahl aller Envelope-Komponenten nötig werden, das Bilden dieser Gruppen kann die weitere Arbeit deshalb erleichtern. Im konkreten Fall wurde also eine Gruppe für alle Hüllobjekte des wichtigsten Envelopes gebildet (Körper, Zähne, Krallen) sowie eine weitere für die Zunge, die getrennt von den anderen Teilen an die Deformer gebunden wird.

Auch die Augen wurden im vorliegenden Fall komplett als funktionsfähige 3D-Konstruktion erstellt, sie wurden nach der von S. Efstathiou beschriebenen Methode aus mehreren teilweise geöffneten Kugeln zusammengesetzt, deren Öffnungswinkel parametrisiert wurden (Efstathiou2003, Abb. 3.10).

Abb. 3.10: Parametrisierbare Augäpfel als funktionsfähige Konstruktion

Durch dieses Vorgehen wurde eine Parametrisierung der Iris- und Pupillengröße ermöglicht, die bei Verwendung einer einfachen Texturierung nicht möglich gewesen wäre. Eine Alternative zur kompletten Modellierung wäre auch noch das Verwenden von prozeduralen Shadern gewesen, die verwendete Methode passt aber zu dem gewünschten realistischen Aussehen der Figur besser.

Es bleibt noch zu erwähnen, dass die Arbeit an der Geometrie des Modells zu diesem Zeitpunkt noch nicht zwingend abgeschlossen sein muss. Besonders nach dem Enveloping ist es unter Umständen notwendig, das Modell noch einmal zu überarbeiten. Mit Knochen, die das Modell deformieren, sieht es an einigen Stellen vielleicht nicht mehr so überzeugend aus wie ohne Knochen.

3.4 Rendering

3.4.1 Texturierung und Shader

Das endgültige Aussehen eines gerenderten Objektes wird, neben globalen Parametern wie etwa der Szenenbeleuchtung oder Rendereinstellungen, maßgeblich durch eine Kombination aus Shadern mit Texturen definiert. Die Möglichkeiten hier sprengen bei weitem den Umfang dieser Arbeit, hier sei auf Spezialliteratur wie etwa das Buch von J. McCann (McCann2003) verwiesen, es soll jedoch im Folgenden auf die Besonderheiten dieser Arbeit bei Charakteren eingegangen werden sowie auf die Texturierung in diesem konkreten Fall des Charakters „Willy“.

Ein Shader ist, kurz gesagt, ein Programmstück, welches auf eine bestimmte Weise das Renderergebnis von Mental Ray, der Render-Engine von Softimage|XSI, beeinflusst. Es gibt in XSI viele verschiedene Arten von Shadern für die unterschiedlichsten Verwendungen, wenn aber in diesem Kapitel von Shadern die Rede ist, sind immer die sogenannten „Surface-Shader“ gemeint, also Shader die das Rendering der Oberfläche eines Objektes beeinflussen. Verschiedene Shader können auch miteinander kombiniert werden, in XSI wird dies mithilfe des „Render Tree“ getan.

Jeder Surface-Shader wiederum kann nun eine große Anzahl von teilweise stark unterschiedlichen Input-Parametern besitzen, wie etwa Reflektivität, Transparenz oder Refraktion. Allen gemeinsam ist aber, dass mindestens ein Input-Parameter für einen Farbwert existiert, in den meisten Fällen sogar mehrere. Alle diese Parameter können auch über Maps beziehungsweise Texturen gesteuert werden, was bedeutet, dass eine Bilddatei als Quelle für die Farbwerte der Oberfläche an bestimmten Stellen verwendet wird. Je nachdem, welcher Parameter dabei durch eine Map gesteuert wird, nennt man diese Technik dann Farb-Mapping, Glanz-Mapping, Transparenz-Mapping usw. (Birn2001).

Für den Charakter „Willy“ wurde als wichtigster Surface-Shader der sog „Phong“-Shader verwendet, der mit verschiedenen Shadern kombiniert wurde. Die Farbwerte dieser Shader-Konstruktion wurden dabei durch eine Textur gesteuert (Abb. 3.11). Auf komplexe Modelle zur Simulation einer realistischen Haut, etwa durch das sogenannte „Subsurface Scattering“, konnte aufgrund der dichten Behaarung des Charakters verzichtet werden.

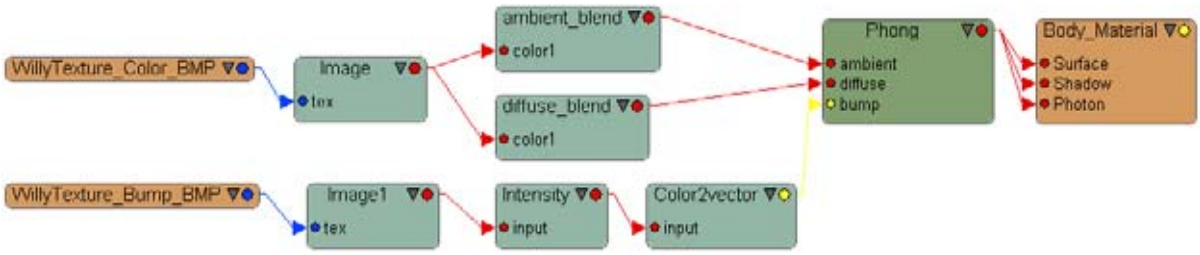

Abb. 3.11: Shader-Konstruktion für die Oberfläche des Körpers von „Willy"

Um nun eine Position auf einem 3D-Objekt einem Punkt auf einer Textur zuzuordnen, muss bei Polygon-Objekten eine sogenannte UV-Projektion, häufig auch UVW-Projektion genannt, erstellt werden. NURBS-Objekte verfügen über implizite UV-Koordinaten, können jedoch in XSI durch eine Projektion auch mit expliziten Koordinaten versehen werden (Birn2001).

Gerade bei lebenden Objekten muss bei der Texturierung besonders darauf geachtet werden, störende Übergänge zwischen Texturen zu vermeiden beziehungsweise geschickt zu verdecken. Diese Problematik ist im vorliegenden Fall besonders bedeutend, da der Teckel, im Gegensatz zu vielen anderen Charakteren, keine von sich aus gegebenen Möglichkeiten hat, Übergänge zu realisieren. Solche Möglichkeiten wären beispielsweise die Übergänge zwischen verschiedenen Kleidungsstücken oder zwischen einem Kleidungsstück und einer Hautpartie. In diesem Fall musste deshalb die Projektion der UV-Koordinaten zu großen Teilen aufwändig manuell korrigiert werden.

Für komplexe Objekte wie Charaktere empfehlen sich nach M. Isner von der „Softimage Special Projects Group“, prinzipiell die folgenden Methoden (Misner2003a):

a) Alle Projektionen eines Objektes werden in einer einzigen zusammengefasst. Hierbei werden meist zunächst mehrere Projektionen auf die Textur erstellt und diese dann später in einer einzigen UV-Projektion zusammengefasst. Diese Methode wird häufig bei Computerspielen verwendet, da sie Speicher spart. Es kann Texturspeicher gespart werden, da nur eine einzige Textur verwendet wird und es wird nur ein Set von UV-Koordinaten benötigt. Sie hat außerdem den Vorteil eines übersichtlichen Render Trees. Sie ist leicht zu handhaben bei sowieso vorhandenen Übergängen, schwerer allerdings, wenn diese nicht vorhanden sind. In diesem Fall muss mit Subprojektionen und manueller Nachbearbeitung gearbeitet werden.

b) Mehrere Projektionen werden angelegt, die jeweils für eine eigene Textur verwendet werden. Übergänge müssen manuell durch Transparenz-Maps oder Ähnliches erstellt werden. Bei XSI bietet sich zu diesem Zweck das „Color at Vertices"-Tool („CAV") an, das es erlaubt, Geometrie-Komponenten Zahlenwerte zuzuordnen, die dann in diesem Fall als Transparenz-Werte verwendet werden können.

c) Wenn möglich, kann das 3D-Objekt auch in mehrere Teile unterteilt werden die dann jeweils eine eigene Projektion bekommen. Häufig wird diese Technik für die Zähne oder Ähnliches verwendet. Bei dem Charakter „Willy" wurden die Zunge und die Krallen ebenfalls als eigene Objekte modelliert und texturiert. Weiche Übergänge zwischen Texturen sind mit dieser Technik allerdings nicht möglich.

d) Es wird die Methode des sogenannten „3D-Painting" angewendet. Hierbei wird das 3D-Modell in eine Spezialsoftware wie beispielsweise „Deep Paint" importiert. Diese Software erlaubt es, direkt auf das Objekt zu „Malen" und generiert dann eine Textur die durch eine Standardprojektion dargestellt werden kann. Diese Methode ist nicht perfekt wenn man Fotografien als Quellen benutzen will anstatt die Textur komplett selbst zu malen, außerdem ist der Export von 3D-Objekten aus Softimage|XSI heraus immer noch nicht ideal gelöst und es muss häufig der Umweg über Softimage|3D gegangen werden.

e) Ebenfalls den Umweg über eine Spezialsoftware wird gegangen, wenn das Modell in eine Software wie „YouMap" exportiert wird, die es erlaubt eine UV-Projektion durch eine Kombination aus mehreren einfachen Geometrieobjekten zu definieren.

Zum Erstellen einer UV-Projektion, die das Projizieren einer Textur mit wenig bis keinen sichtbaren Kanten ermöglicht, wurde im konkreten Anwendungsfall die folgende Vorgehensweise gewählt:

Zunächst wurde eine planaare Projektion für die linke Seite des Objektes erstellt, einschließlich einer Subprojektion für das linke Ohr, das als einziges Körperteil deutlich in einer anderen als der Projektionsebene ausgerichtet ist. Dieses führt zu einer großen Projektion für die gesamte linke Seite, die anschließend manuell optimiert werden musste, um eine an allen Stellen des Objektes möglichst gleich hohe Texturauflösung zu gewährleisten. Zum Abschluss konnten aufgrund der Symmetrie des Objektes die Projektionen der Polygone der rechten Seite des Objektes durch eine einfache Spiegelung aus den UV-Koordinaten der linken Seite gebildet werden (Abb. 3.12).

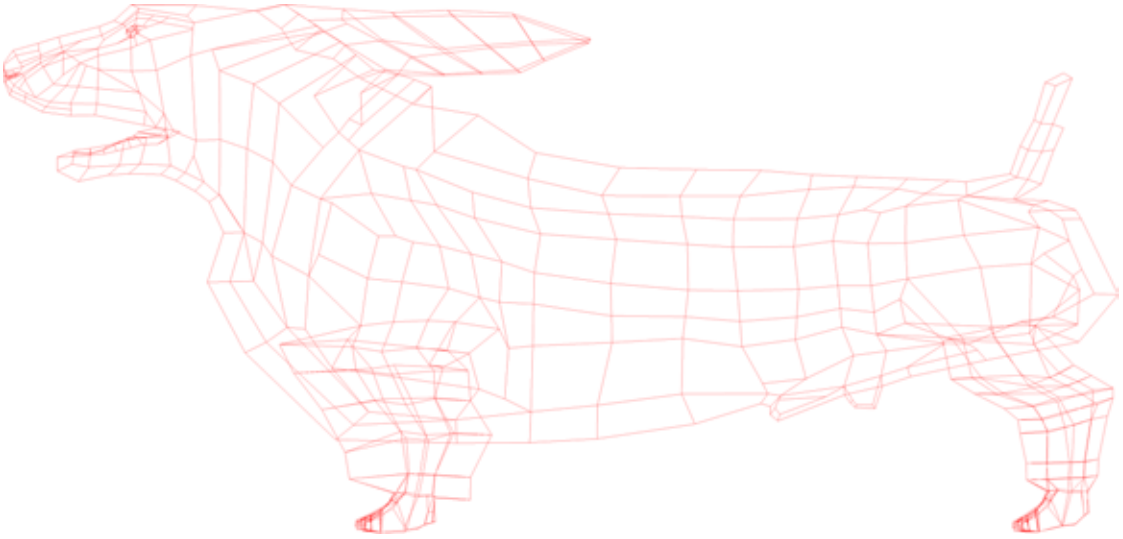

Abb. 3.12: Die fertige UV-Projektion der Texturkoordinaten

Dabei fiel leider eine Unzulänglichkeit von Softimage|XSI (Version 3.5) störend auf: Operatoren wie etwa der „MeshMerge“-Operator können offenbar, wenn die Topologie der Input-Objekte verändert wird, Ihre Geometriekomponenten nicht mehr einwandfrei zuordnen. Dies führt leider dazu, dass UV-Koordinaten bei der angewendeten Modellierungsmethode nicht absolut korrekt von den beiden Hälften des Hundekörpers auf das vereinigten Mesh übertragen werden.

Wie fast immer gilt auch bei der hier besprochenen Arbeit, dass Optimierungen der Darstellungs- und Rendergeschwindigkeit meistens zulasten der Qualität gehen und Qualitätsoptimierungen meistens zulasten der Darstellungs- und Rendergeschwindigkeit gehen. Das Geheimnis liegt in diesem Fall darin, einen guten Kompromiss zu finden. In dem konkreten Fall betraf dies eigentlich nur die Auflösung der verwendeten Textur, deren Speicherbedarf die Renderzeit etwas beeinflussen kann, andere aufwändige Shader- oder Rendereffekte wie Transparenz oder globale Beleuchtungsmodelle wurden nicht benötigt (XsiDoc2002, Lights & Cameras).

Die für die Farbwerte verwendete Textur entstand schließlich in dem Programm „Adobe Photoshop“ und basiert zu einem großen Teil aus Videomaterial und Nahaufnahmen eines lebendigen Tieres. Die Textur wurde, um auch überzeugende Nahaufnahmen zu ermöglichen, in einer relativ hohen Auflösung von etwa 4000 * 2000 Pixeln erstellt (Abb. 3.13). Als Bumpmap wurde eine ähnliche Textur erstellt, die durch eine etwas andere Maserung das Gefühl von Plastizität verstärkt. Durch die Funktion „Stamp UV Mesh“ von Softimage|XSI war es möglich, ein Abbild der UV-Koordinaten aus XSI heraus zu exportieren und in Photoshop als Vorlage zu verwenden.

Abb. 3.13: Die für die Farbwerte verwendete Textur

3.4.2 Haare und Fell

Erst seit wenigen Jahren ist eine qualitativ hochwertige Darstellung von Haaren und Fell in der Computergrafik möglich, Softimage|XSI ist dabei eines der wenigen Softwarepakete, das eine Haarsimulation direkt in die Animations- und Modellierungssoftware integriert hat. Das „Hair"-Modul von XSI zeichnet sich durch einfache Bedienbarkeit, komplette Integration in das Programm, etwa in das Simulationsmodul, sowie qualitativ hochwertige Ergebnisse bei akzeptabler Renderzeit aus, was XSI zur weltweit wohl führenden Umgebung in dem Bereich der Haarsimulationen macht.

Durch eine relativ hohe Texturqualität bei dem im Rahmen dieser Arbeit erstellten Charakter wäre eine Haarsimulation für Ansichten aus größerer Entfernung nicht unbedingt erforderlich, ein sichtbarer Unterschied wäre kaum zu erkennen. Um allerdings auch Nahaufnahmen überzeugend darstellen zu können, wurde ergänzend die Haarsimulation eingesetzt.

Ein weiterer Vorteil dabei ist, das leicht störende Textur-Artefakte an der Symmetrieachse des Körpers von „Willy" durch die Haarsimulation aufgelockert und verdeckt werden. Diese Artefakte sind unvermeidbar, wenn eine Textur für beide Seiten des Objektes verwendet werden soll und entstehen durch die Spiegelung der Textur (Abb. 3.14, Abb. 3.15).

Textur und Haarsimulation als Kombination haben sich dabei sehr bewährt. Dadurch, dass die Haarsimulation nur als Ergänzung benötigt wird und es erwünscht ist, dass die Textur durch die Haare hindurch zu sehen ist, können relativ wenige Haare verwendet werden, was wiederum Rechenzeit spart. Die Textur kann mit dieser Technik außerdem als Map für die Haarfarbe eingesetzt werden, was die Arbeit erspart, die mit dem Definieren der Haarfarbe, etwa für Pfoten und Gesicht, einhergeht.

Abb. 3.14: „Willy" ohne Haarsimulation: Durch Spiegelung der Textur entstandene Artefakte sind gut zu erkennen (linke untere Ecke)

Abb. 3.15: „Willy" mit Haaren: Artefakte werden überdeckt, Textur und Haarsimulation ergänzen sich

Beim Arbeiten mit Haaren ist unbedingt zu beachten, dass eine Deformation eines Haar-Objektes äußerst rechenaufwändig ist. Haar-Objekte sollten also, wenn sie beispielsweise wie im vorliegenden Fall durch eine Envelope-Operation ständig deformiert werden, nur zur Renderzeit dargestellt werden, nicht in einem Viewport. Die Haare, die das Fell von „Willy" bilden, sind standardmäßig nicht in den Viewports sichtbar, lassen sich aber durch die entwickelte Charaktersteuerung sichtbar machen (Kapitel 6, „Entwicklung einer grafischen Benutzeroberfläche zur Animationssteuerung").

Zum Modellieren der Länge und Ausrichtung der Haare wurden die sogenannten „Guide-Hairs", anhand derer Transformationen und Deformationen die anderen, letztendlich renderbaren, Haare interpoliert werden, mit den standardmäßigen Haar-Bearbeitungstool von XSI in Form gebracht. Dabei haben sich die in der Version 3.5 von XSI neu hinzugekommenen Tools wie der „Recomb"-Operator und das sogenannte „Flex-Styling" bewährt. Die Möglichkeit, Haare mithilfe der Physik-Simulation von XSI zu stylen, so wie es das XSI-Handbuch vorschlägt, hat sich dagegen nicht als effektiv erwiesen. Bei Objekten mit einer größeren Anzahl von Guide-Hairs, wie in diesem Fall, ist die dynamische Haar-Simulation entschieden zu rechenaufwändig und damit zu langsam für eine sinnvolle Anwendung zu diesem Zweck (XsiDoc2002, Simulation).

Die wichtigste Neuerung des Haar-Moduls bei Version 3.5 von XSI ist allerdings die Unterstützung von Subdivision-Surfaces als Haaremitter. Bis einschließlich Version 3.0 konnte ein Polygon-Objekt, welches als Subdivisions-Hüllobjekt definiert war, zwar Haare emittieren, diese wuchsen allerdings immer direkt von der Hülle des Objektes aus. Da Subdivision Surfaces die Eigenschaft haben, die Oberfläche des Objektes etwas zu verkleinern, standen diese Haare damit quasi „in der Luft". Mit Version 3.5 ist es

nun möglich, im Haarobjekt den Subdivisions-Level anzugeben, der dann bei der Emittierung der Haare berücksichtigt wurde.

Da die vorliegende Arbeit zunächst auf der Version 3.0 von Softimage|XSI geplant war, wurde ein Workaround entwickelt, das vorsah, ein eigenes Subdivision-Objekt in der benötigten Auflösung zu generieren und dieses dann als Haaremitter zu verwenden. Wenn die Konstruktionshierarchie nicht eingefroren würde, würde eine Envelope-Deformation des eigentlichen Hüllobjektes auch auf das Subdivisions-Objekt übertragen werden, dieses müsste selbst nicht in den Envelope übernommen werden. Die Anzeige des Subdivisions-Objekt könnte dann unterdrückt werden, einzig die von ihm emittierten Haare würden für das Rendering sichtbar gemacht. Natürlich ist diese Vorgehensweise beim Arbeiten mit Softimage|XSI 3.5 nicht mehr nötig und unnötig kompliziert, so dass sie in der endgültigen Version von „Willy“ auch keine Verwendung mehr fand.

Abschließend soll angemerkt werden, dass nicht in jedem Fall eine Haarsimulation die perfekte Methode ist, um den Effekt echter Haare zu erwecken. So ist beispielsweise für sehr dicke Haare oder auch Haarstränge, wie die sogenannten „Dreadlocks“, oft eine Cloth-Simulation die bessere Wahl.

3.4.3 Das Rendern

An dieser Stelle soll, teilweise vorbereitend, schon einmal ein wenig auf das Rendern unseres Charakters eingegangen werden. Prinzipiell gibt es bei dem Rendern von Charakteren keine allzu großen Besonderheiten, so dass nur kurz auf einige Erfahrungen eingegangen werden soll, die speziell mit dem Charakter „Willy“ gemacht wurden (Abb. 3.16, Abb. 3.17).

Abb. 3.16: Der Charakter „Willy", bereit animiert zu werden

Trotz der Haarsimulation ergeben sich für „Willy“ keine allzu langen Renderzeiten, auf einem durchschnittlichen Grafik-PC liegen die Rechenzeiten für ein Frame bei etwa 120 Sekunden, was bedeutet, dass zum späteren Rendern einer Animation für etwa eine Sekunde animiertes Material etwa 30 Minuten benötigt werden. Berechnungsgrundlage dafür ist eine Bildrate von 15 Frames pro Sekunde, was für Trickfilme durchaus ausreichend und auch gebräuchlich ist.

Um diese Zeiten zu erreichen, wurde auf globale Beleuchtungsmodelle wie „Global Illumination“ oder „Final Gathering“ verzichtet, diese Modelle sind leider immer noch mit einem zu großen Rechenaufwand verbunden, um sie in einem größeren Umfang einzusetzen. Für den erwünschten Stil der Animation waren diese Effekte allerdings auch abdingbar.

Abb. 3.17: „Willy" in Nahaufnahme

3.5 Zusammenfassung

Im Laufe dieses Kapitels wurde ein Charakter entworfen, seine Animation geplant, er wurde modelliert und schließlich texturiert. Dies alles sind vorbereitende Tätigkeiten, bei denen die spätere Animation selbst nie aus den Augen verloren werden darf. Modellierung und vor allem Texturierung müssen dabei nicht zwangsläufig bereits zu diesem Zeitpunkt abgeschlossen sein, sondern können sogar noch nach der Animation fertiggestellt werden. Die grundsätzliche Vorgehensweise bei diesen Arbeitsgängen sollte aber zu diesem Zeitpunkt schon feststehen und auf Durchführbarkeit geprüft werden.

Am Ende dieses Kapitels steht sozusagen die leblose Hülle des Charakters, die nächsten Kapitel werden sich mit dem Animieren, also dem „Beleben“ des Charakters beschäftigen. Als Vorbereitung dazu werden zunächst als kleiner Exkurs in Kapitel 4 die technischen Mittel vorgestellt, die zum Aufbau eines Control-Rigs nötig sind.

4 Technische Mittel zum Aufbau des Control-Rigs

4.1 Einleitung

Im letzten Kapitel wurde die Vorbereitung eines Charakters auf die Animation besprochen. Bevor im nächsten Kapitel 5 der „Aufbau des Control-Rigs“ besprochen wird, sollen zunächst die wichtigsten technischen Voraussetzungen besprochen werden, die für diese Arbeit nötig sind.

Dabei muss dieses Kapitel ganz speziell die Arbeit mit Softimage|XSI ansprechen, dennoch sind die meisten hier besprochenen Konzepte, zumindest in ähnlicher Form, auch in anderen vergleichbaren Softwarepaketen enthalten.

„Hierarchien und Constraints“ (Kapitel 4.2) werden dabei hauptsächlich für das Gerüst benötigt, das zur Animation nötig ist, während „Properties in Softimage|XSI“ (Kapitel 4.3) und „Scripting in Softimage|XSI“ (Kapitel 4.4) auch später für die Programmierung einer grafischen Benutzeroberfläche zur Steuerung des Charakters benötigt werden.

4.2 Hierarchien und Constraints

Auf Hierarchien und Constraints wurde bereits in Kapitel 2.3.4, „Inverse Kinematik“, hingewiesen. Das Thema soll hier kurz wiederholt und ergänzt werden, außerdem soll besonders auf die Unterschiede zwischen diesen beiden Techniken und ihre Kombinationsmöglichkeiten eingegangen werden. Beide Techniken können dazu verwendet werden, Strukturen aufzubauen, in denen die Transformationen von Elementen Einfluss auf die anderer Elemente haben. Jede dieser beiden Methoden hat jedoch ganz spezielle Vorteile der jeweils anderen gegenüber.

Bei Hierarchien stehen Elemente, die miteinander verbunden sind, zueinander in einer „Parent-Child-Beziehung“, wobei das in der Hierarchie weiter oben stehende Element als „Parent“ und die ihm direkt untergeordneten Elemente als „Children“ bezeichnet werden. Ein Parent kann mehrere Children besitzen, ein Child aber nur ein direktes Parent.

Werden die Transformationen des Parents einer Hierarchie verändert, verändern sich im Normalfall im gleichen Maße auch die entsprechenden globalen Transformationen der Children. Dieses Verhalten kann durch Abschalten der Option „Child Compensation“ unterdrückt werden. Die lokalen Transformationen eines Elements beziehen sich auf das Parent-Objekt und werden daher nicht verändert.

Parent-Child-Beziehungen können in Softimage|XSI sowohl im Explorer als auch in der Schematic View sichtbar gemacht werden, weshalb diese Beziehung teilweise auch nur zum Ordnen von Elementen in der Hierarchie der gesamten Szene eingesetzt werden (vergleiche „Anhang D: Die Schematic View“, Abb. 4.1).

Verschiedene Tools und Operatoren von Softimage|XSI erzeugen automatisch eine Hierarchie, beispielsweise das „Draw-Chain-Tool“, außerdem sind etwa beim Erstellen einer neuer Szene automatisch alle darin enthaltenen Objekte Children des Szene-Objektes.

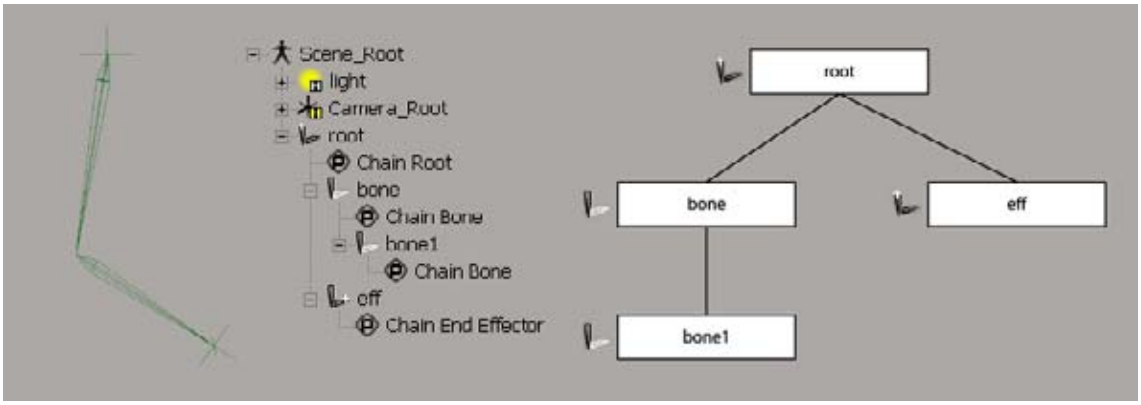

Abb. 4.1: Hierarchie einer trivialen IK-Kette (Links: Darstellung in einem Viewport, Mitte: Darstellung im Explorer, Rechts: Darstellung in der Schematic-View)

Eine Struktur mit ähnlichen Eigenschaften wie eine Parent-Child-Hierarchie von Objekten kann ebenfalls durch Constraints konstruiert werden. In diesem Fall besteht der größte Unterschied zu Hierarchien darin, dass explizit nur ganz bestimmte Transformationen, wie etwa Skalierung, Rotation oder Translation, weitergegeben werden können. Bestimmte Constraints beeinflussen die Transformationen der Zielobjekte auf andere Art, indem sie beispielsweise das Objekt auf ein anderes Ausrichten .

Ein weiterer Unterschied besteht darin, dass ein Element, welches durch ein Constraint mit einem anderen verbunden ist, sich nicht notwendigerweise im lokalen Koordinatensystem des anderen befinden muss. Außerdem sind Verknüpfungen durch Constraints im Explorer sowie in der Schematic View nicht so offensichtlich zu erkennen, sie eignen sich also nicht zum Ordnen von Elementen.

Es gibt in Softimage|XSI eine große Auswahl von unterschiedlichen Constraints, etwa das „Position“-Constraint, das „Rotation“-Constraint, das „Direction“-Constraint, welches ein Objekt auf ein anderes ausrichtet, oder das „2 Points“-Constraint, welches ein Objekt zwischen zwei anderen Objekten platziert (Abb. 4.2). Für eine Beschreibung dieser und aller weiteren Constraints sei an dieser Stelle auf die Dokumentation zu Softimage verwiesen (XsiDoc2002, Animation).

Beim Arbeiten mit Constraints ist zu beachten, dass die Objekte, auf die Constraints angewendet werden, im globalen Koordinatenraum transformiert werden. Deshalb werden alle Einstellungen, die im lokalen Koordinatenraum getroffen wurden, überschrie-

ben, wenn sie vor der Einrichtung eines Constraints getroffen wurden. Außerdem ist beim Arbeiten mit Constraints darauf zu achten, sogenannte „Constraint-Dependency Cycles“ zu vermeiden, also Schleifen von Constraints. Dies kann zu unvorhersehbaren Ergebnissen führen (XsiDoc2002, Animation).

Im Gegensatz zu Hierarchien, in denen ein Objekt immer nur ein direktes Parent hat, kann ein Objekt durchaus mehrere Constraints besitzen. Diese Constraints wirken gleichzeitig auf das Objekt ein und können untereinander übergeblendet werden.

Außerdem können Constraints mit einem Offset versehen werden. Das heißt, dass der oder die vom Constraint beeinflussten Parameter bei dem/den beeinflussenden Objekt(en) und dem beeinflussten Objekt nicht genau gleich sind, sondern ein fester Wert als Unterschied beibehalten wird. Hierzu kann das von einem Constraint beeinflusste Objekt mit aktivierter „Constraint Compensation“ transformiert werden, der Offset kann aber auch direkt in der Property Page des Constraints angegeben werden.

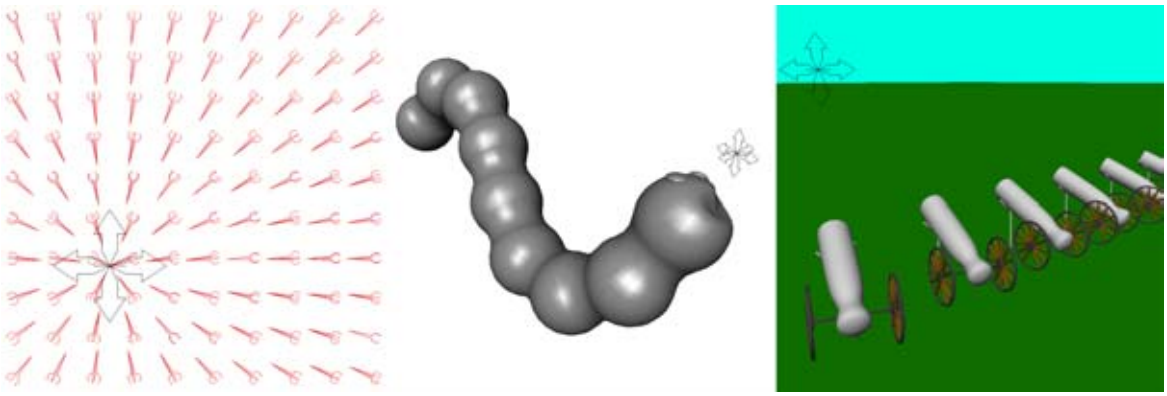

Abb. 4.2: Beispiele für Konstruktionen, deren Ausrichtung mithilfe von Constraints allein durch die Position eines Objektes geändert werden kann (Links: Orientation-Constraints, Mitte: Distance-Constraints, Rechts: Orientation- und Direction-Constraints)

Auch Constraints werden von verschiedenen Tools in Softimage|XSI automatisch verwendet. An erster Stelle sei hier das „Spline-Tool“ erwähnt, das ideal zum Aufbauen von IK-Ketten ist, die als Rückgrat oder Ähnlichem genutzt werden. Es erlaubt, eine Kette von beliebig vielen Knochen durch die Position und Rotation von zwei anzugebenden Objekten zu steuern. Es kann außerdem sehr gut zum Konvertieren von IK-Chains in Kurven und dann gegebenenfalls in andere IK-Chains benutzt werden. Diese Funktionalität ist beispielsweise praktisch, um eine IK-Kette mit drei Knochen, die durch den Animator gesteuert wird, in eine Kette mit sechs Knochen zu konvertieren, die dann zum Enveloping genutzt wird (Misner2003b, vergleiche auch Kapitel 5.5.4, „Schwanz“).

4.3 Properties in Softimage|XSI

Jedes Element, das in einer Softimage|XSI-Szene vorkommt, hat sogenannte „Properties“ oder auch „Parameter“, also Eigenschaften, die es von anderen unterscheiden. Diese Properties werden in sogenannten „Property Editors“ oder auch „Parameter Sets“

gesammelt und gruppiert. Properties können etwa die Farbe oder die Größe eines Objektes definieren, sie können den Zustand eines Constraints bestimmen oder auch die Translationen eines Objektes. Die Property Editoren für die Translationen etwa finden sich im Explorer unter der „Kinematics“-Gruppe des jeweiligen Objektes und nennen sich „Global Transform“ und „Local Transform“.

Properties können nicht nur zu einzelnen Objekten gehören, sie können auch ganzen Hierarchien zugewiesen sein, das heißt sie werden dann von einem Parent-Objekt auf seine Children vererbt.

Jedes Element einer Szene kann zusätzlich auch mit sogenannten „Custom Parameter Sets“ ausgestattet werden, die „Custom Parameter“ enthalten können. Damit lassen sich also für jedes Objekt zusätzliche Parameter definieren. Außerdem können natürlich Objekte wie etwa Null-Objekte nur für den Zweck der Datenspeicherung erstellt werden, als Properties könnte man beispielsweise die Translationen des Objektes verwenden. Softimage|XSI kennt in dem Sinne keine lokalen Variablen, auf Parameter auf die grundsätzlich zugreifbar ist kann von jedem Element in der Szene aus zugegriffen werden.

Eine besondere Form der Custom Parameter sind die sogenannten „Proxy Parameter“. Diese sind direkt mit einer anderen Property verlinkt. Das bedeutet, dass jede Operation, die mit einem Proxy Parameter durchgeführt wird, genau die gleichen Auswirkungen hat, die sie hätte, wenn sie mit dem verknüpften Property direkt ausgeführt werden würde. Dies gilt auch umgekehrt, das heißt Änderungen an der verlinkten Property wirken sich auch auf den Proxy Parameter aus.

Custom Parameter können als verschiedene Typen erstellt werden, im einzelnen als Fließkommazahl, als Integerzahl, als Boolescher Wert oder als „Text“. Bis auf den Parametertyp „Text“ lassen sich die verschiedenen Parametertypen mithilfe von Expressions ineinander konvertieren. Die Parameter des Typs Fließkommazahl und Integerzahl müssen mit einem Wertebereich versehen werden, also mit einem Minimal- und einem Maximalwert. Die Parameter aller Typen außer des Typs „Text“ lassen sich weiterhin als animierbar beziehungsweise nicht animierbar kennzeichnen, alle Typen lassen sich als schreibgeschützt definieren.

Das Aussehen aller Parametersets, also auch der Custom Parameter Sets wird durch sogenannte „SPDL-Dateien“ definiert. In Ihnen wird beispielsweise die Reihenfolge der Parameter definiert, das weitere Layout des Parametersets und Ähnliches. Indem man also die SPDL-Dateien von Custom Parameter Sets mit einem externen Editor editiert, besteht eine Möglichkeit, diese genauer anzupassen als es die Standard-Tools erlauben. Es stehen dann zum Beispiel auch weitere Eingabemöglichkeiten wie etwa Combo-Boxen oder Radio-Buttons zur Verfügung.

4.4 Scripting in Softimage|XSI

4.4.1 Expressions

Expressions sind in Softimage|XSI kurze Programmstücke, die jeweils einem Parameter zugewiesen werden. Dies kann jeder beliebige Parameter sein, der auch animierbar ist. Sobald eine Expression einen Parameter steuert, kann dieser nicht mehr selbst verändert werden, weder durch Skripts oder Ähnliches, noch interaktiv aus der GUI von Softimage|XSI heraus. Einzige Ausnahme hier sind die sogenannten „Linked Parameters“, eine Sonderform der Expressions (siehe unten).

Ein Beispiel für eine simple Expression ist etwa die folgende. Sie wird dem Intensitätswert einer Lichtquelle zugewiesen, der damit durch einen Lichtschalter („switch“) gesteuert wird (Quelle: Rossano2003)

```
[MyLight.light.soft_light.intensity =]
cond (switch.kine.local.roty > 45 , (.3 + (3 *
(switch.kine.local.roty / 360))) , 0)
```

In dieser Notation ist die eigentliche Expression nur der Teil des Ausdrucks, der hinter dem „ist-gleich“-Zeichen steht. Der Teil des Ausdruckes, der in eckigen Klammern steht, verweist auf den Parameter, den die Expression steuert.

Auf Properties wird mit der „Punkt-Notation“ zugegriffen, an erster Stelle steht dabei (gegebenenfalls) der Name des Models, dann der Name des Objektes und anschließend die betreffende Property, die ebenfalls noch weiter unterteilt sein kann (etwa „kine.local.roty“). Es stehen Funktionen für die üblichen mathematischen Anwendungen zur Verfügung, außerdem Funktionen um auf die Werte von Funktionskurven zu bestimmten Zeitpunkten zurückzugreifen.

Die Einschränkungen von Expressions bestehen darin, dass sie nur aus einem einzigen Ausdruck bestehen, der dadurch unter Umständen sehr komplex und schwer zu lesen wird. Steuerungsstrukturen wie Schleifen sind nicht vorhanden, einzig eine rudimentäre Fallunterscheidung steht zur Verfügung. Variablen können, mit Ausnahme von vordefinierten Variablen, wie etwa Pi, ebenfalls nicht innerhalb von Expressions definiert werden. Da jedoch alle Parameter innerhalb eine XSI-Szene global sind, können Custom Parameter oder Standard-Parameter von extra für diesen Zweck angelegten Dummy-Objekten für diesen Zweck verwendet werden.

Obwohl die Struktur von Expressions relativ einfach ist, sind sie äußerst nützlich, um die Animation sich wiederholender Vorgänge oder von Vorgängen, die auf mathematischen Funktionen beruhen, zu vereinfachen. Im Rahmen dieser Arbeit werden Expressions vorwiegend benutzt, um dem Charakter „Willy“ eine größere Komplexität seiner Bewegungen zu verleihen und die Steuerung seiner Animation zu vereinfachen.

Die sogenannten „Linked Parameters“ („verlinkte Parameter“) sind eine spezielle Form von Expressions, auch wenn diese Verwandtschaft bei Benutzung der GUI von XSI nicht unbedingt sofort und deutlich klar wird. Einem Parameter, der mit einem oder

mehreren anderen Parametern verlinkt ist, wird automatisch die Funktion „l_fcv()“ als Expression zugewiesen. Außerdem wird für diese Expression eine Funktionskurve generiert, die allerdings nicht, wie sonst bei Funktionskurven, den Wert eines Parameters im Verlauf der Zeit (der Frames) darstellt sondern den Parameter über den mit ihm verlinkten Parameter aufträgt. Dadurch ist es möglich, eine nichtlineare und beliebig komplexe Zuordnung der Parameterwerte zu definieren, die mit mathematischen Funktionen und den Möglichkeiten von Expressions alleine kaum definierbar wäre.

4.4.2 Scripted Operators

„Scripted Operators“ sind eine andere Form des Scriptings in Softimage|XSI, sie stellen dem Programmierer etwas mehr Möglichkeiten zur Verfügung als Expressions.

Scripted Operators können beliebig lange Programmstücke sein, die entweder in VBScript oder in JScript verfasst sind. Daher erlauben sie auch die von diesen Skriptsprachen unterstützten Kontrollstrukturen sowie das Definieren von lokalen Variablen, die nur innerhalb des Scripted Operators Gültigkeit haben, in welchem sie definiert wurden (Kapitel 4.4.5, „ Die Skriptsprachen“).

In Scripted Operators darf nur ein bestimmter Teil der Funktionen und Kommandos verwendet werden, die in Skripts verwendet werden dürfen, und zwar nur eine Untergruppe der sogenannten „Model Statements“. „Commands“ dürfen nicht verwendet werden. (Für eine Erläuterung dieser beiden Begriffe siehe Kapitel 4.4.4, „Gegenüberstellung des Command- und des Object-Models“)

Um innerhalb eines Scripted Operators auf Daten zuzugreifen, muss zunächst explizit eine sogenannte „Connection“ erstellt werden. Je nachdem, ob Daten gelesen oder geschrieben werden sollen, muss diese Connection eine Input- oder eine Output-Connection sein.

Es bestehen noch weitere Unterschiede zwischen Expressions und Scripted Operators. So kann eine Property, der eine Expression zugeordnet wurde, nicht manuell verändert werden. Nicht so bei Scripted Operators, diese können zusätzlich zu einer bestehenden Funktionskurve erstellt werden. Anders als bei Expressions kann ein Scripted Operator außerdem mehreren Parametern zugeordnet werden, genauso wie ein Parameter mehreren Scripted Operators zugeordnet werden kann.

4.4.3 Scripts

Die sogenannten „Scripts“ bieten wiederum etwas mehr Möglichkeiten als die Scripted Operators, haben allerdings auch ein etwas anderes Anwendungsgebiet.

Scripts können entweder bei bestimmten Ereignissen aufgerufen werden, etwa bei einem Wechsel der Auswahl, sie können in einer Partikelsimulation das Verhalten der Partikel steuern, sie können im Batch-Mode ohne ein Starten der GUI aufgerufen werden oder mit DHTML kombiniert in der „Net View“ (nur bei Softimage|XSI unter Win-

dows-Betriebssystemen vorhanden) eingesetzt werden. Außerdem können Scripts als Kommandos in XSI registriert werden und dann über Buttons aufgerufen werden.

Im Rahmen dieser Arbeit werden Scripts in der sogenannten „Synoptic View“ eingesetzt. Dort werden sie durch die Synoptic View aufgerufen, um bestimmte Elemente des Charakters und der Szene zu steuern und so die Charaktersteuerung zu übernehmen. Für eine genauere Definition der Synoptic View und den Einsatz von Skripten für diesen Zweck siehe Kapitel 6.4.

Skripts können sowohl mit Command-Model-Statements und -Notation geschrieben werden, als auch mit Object-Model-Statements und -Notation. Die Kombination beider Möglichkeiten ist ebenfalls möglich (Kapitel 4.4.4).

Außerdem können Scripts in jeder von Softimage|XSI unterstützen Skriptssprache verfasst werden, dazu zählt zum Beispiel VBScript, Jscript, PerlScript, Python ActiveX Scripting und alle anderen ActiveX-kompatiblen Skriptsprachen. Vermischt werden können diese Skriptsprachen aber innerhalb eines Skriptes nicht (Kapitel 4.4.5).

4.4.4 Gegenüberstellung des Command- und des Object-Models

Softimage|XSI unterstützt zwei verschiedene APIs (Application Program Interface), das Command- und das Object-Model. Beide Programmierweisen können beliebig kombiniert werden.

Das Command-Model basiert auf den Kommandos, die auch die GUI von XSI zu Verfügung stellt. So gut wie jede Aktion, die über das GUI von XSI ausgeführt werden kann, ruft eines oder mehrere Kommandos auf. Scripting mit dem Command-Model bedeutet das manuelle Aufrufen dieser Kommandos innerhalb eines Skripts. In der sogenannten „Command Box“ von XSI werden alle ausgeführten Kommandos protokolliert, was das Scripting vereinfachen kann.

Beispiel für das Command-Model:

```
GetPrim „Null“, „MyNull“
```

Das Object-Model dagegen basiert auf der inneren Darstellung der Objekte selbst in Softimage|XSI, also auf einer objektorientierten Herangehensweise. Es gibt Klassen und Subklassen wie etwa „Geometry“, „Point“ oder „ShapeClip“, die Methoden und Eigenschaften haben. Scripting mit dem Object-Model bedeutet zunächst ein Objekt zu identifizieren und dann einen Parameter entweder zu lesen beziehungsweise zu setzen oder eine Methode aufzurufen (Rabiler2001).

Beispiel für das Object-Model:

```
Set mySceneRoot = Application.ActiveProject.ActiveScene.Root
set myNull = mySceneRoot.AddNull(„MyNull“)
```

Programmieren mit dem Command-Model fällt in den meisten Fällen wesentlich einfacher, die Programmsequenzen laufen allerdings meistens langsamerer ab als vergleich-

bare Sequenzen, die im Object-Model programmiert sind. Deshalb empfiehlt sich für Scripts, die sehr häufig aufgerufen werden, wie etwa für solche, die das Verhalten von Partikeln beeinflussen, unbedingt das Object-Model. Die meisten Anwendungen von Scripts, insbesondere die hier verwendete Steuerung eines Charakters, sind aber weniger zeitkritisch. So ist beispielsweise für den User weniger wichtig, ob das Selektieren eines Steuerungsobjektes 0,1 sec oder 0,15 sec dauert. Aus diesem Grund sind die im Rahmen dieser Arbeit erstellten Skripts zum größten Teil im Commad-Model verfasst.

Soll ein Script oder Teile daraus jedoch weiter auf die Laufzeit hin optimiert werden, so hat es sich bewährt, zunächst eine grobe Version des Skriptes mithilfe des Command-Models zu programmieren, und danach Stück für Stück die besonders zeitaufwändigen Passagen in das Object-Model umzuformulieren (Misner2001).

4.4.5 Die Skriptsprachen

Wie schon erwähnt, können für Scripts in XSI alle ActiveX-kompatiblen Scriptsprachen verwendet werden. Dazu gehören etwa VBScript, JScript, PerlScript und Python ActiveX Scripting. Eine Ausnahme ist die Synoptic View, Scripts, die für sie geschrieben sind, können nur in VBScript oder JScript verfasst sein. Standardskriptsprache für alle Anwendungen ist VBScript.

Für die vorliegende Arbeit wurden Skripts nur in der Synoptic View eingesetzt. Da innerhalb einer Synoptic View nur eine einzige Skriptsprache zum Einsatz kommen kann, wurde ausschließlich VBScript eingesetzt. Prinzipiell unterscheiden sich die möglichen Sprachen jedoch nicht sehr, zumindest für die doch relativ begrenzte benötigte Funktionalität sind sie relativ ähnlich. Für VBScript sprach deshalb vor allem die freie Zugänglichkeit des Entwicklerpaketes und der Dokumentation, die auf der Internetseite für Softwareentwickler von Microsoft erreicht werden kann. Dadurch, dass es die Standardskriptsprache von Softimage|XSI ist, wird VBScript selbst außerdem automatisch mit XSI mitinstalliert.

VBScript ist eine objektorientierte Skriptsprache, welche die üblichen Steuerungsstrukturen bietet, eine umfangreiche Bibliothek von Operatoren und Funktionen mitliefert und ActiveX unterstützt. Parameter werden in VBScript mithilfe des Konzeptes „Passing by Reference“ an Funktionen übergeben, als Datentyp kennt es nur einen, den sogenannten „Variant“. Dieser hat jedoch zahlreiche Subtypen, die automatisch zugewiesen werden. Unter anderem gibt es neben den typischen (Sub-)Typen wie „Integer“, „Double“ und „String“ auch den Typ „Objekt“ (Microsoft2001, Hayes2003).

VBScript bietet außerdem als Besonderheit den anderen Sprachen gegenüber neben den üblichen Schleifenkonstruktionen auch die „For Each....Next“-Konstruktion, die automatisch einen Programmteil auf alle Elemente einer Liste von Objekten anwendet, was für die hier benötigte Funktionalität relativ praktisch ist.

4.5 Zusammenfassung

Softimage|XSI stellt dem Entwickler zahlreiche Möglichkeiten zur Verfügung, um durch Scripting ein gewünschtes Verhalten zu erzielen. Die verschiedenen Möglichkeiten ergänzen sich gegenseitig und haben alle ihre Daseinsberechtigung in speziellen Anwendungsfällen. In den nächsten zwei Kapiteln dieser Arbeit werden alle hier besprochenen Techniken in mehr oder weniger großem Umfang eingesetzt, um zunächst ein Control-Rig aufzubauen, welches den Charakter steuert (Kapitel 5), und anschließend eine Benutzeroberfläche zu Verfügung zu stellen, die es dem Animator ermöglicht, den Charakter effizient zu animieren (Kapitel 6).

5 Aufbau des Control-Rigs

5.1 Einleitung

Dieses Kapitel befasst sich mit dem wohl technisch schwierigsten und vielleicht auch wichtigstem Arbeitsgebiet, das beim Aufbau eines computergenerierten Charakters benötigt wird. Der Aufbau eines Control-Rigs, welches die Bewegung eines Charakters ermöglicht und steuert (Kapitel 2.3.9, Character-Control-Rig), wird in der Praxis deshalb meistens von spezialisierten Fachkräften übernommen. Diese Fachkräfte werden häufig als Technical Directors (TDs) bezeichnet, ihr Aufgabengebiet umfasst in diesem Zusammenhang das Erstellen des Setups für einen Charakter, welches es erlauben muss, diesen Charakter glaubwürdig zu animieren und dabei noch möglichst zeitsparend arbeiten zu können (Rossano2002).

Ein gutes Control-Rig sollte dem Animator also ermöglichen, den Charakter schnell in gute und gewünschte „Posen" zu bringen und gleichzeitig die Möglichkeit geben, die „inneren Werte" eines Charakters für feinere, detaillierte Steuerung zugänglich zu machen (Murtak2003).

5.2 Nachbilden des Bewegungsapparates

5.2.1 Skelettstruktur

Während bei einem menschlichen Charakter häufiger das von XSI zur Verfügung gestellte Standardskelett verwendet werden kann, sind die Unterschiede, was Bewegungsart und Skelettaufbau angeht, bei Vierbeinern untereinander so groß, dass es meistens nötig wird ein eigenes Rig zu entwickeln.

Teile der bereits in Kapitel 3.2, „Entwurf des Charakters", vorgestellten, gesammelten Materialien zur Anatomie und Bewegung des Charakters konnten als Referenzmaterial für die Anatomie von Hunden auch beim Aufbau der Skelettstruktur verwendet werden. Besonders bei real existierenden Charakteren ist für diesen Zweck auch die Darstellung des Skeletts eines lebendigen Tieres praktisch. Im Falle des Teckels „Willy" konnte ein vergleichbares Hundeskelett als Referenz verwendet werden.

Im Idealfall sollten Fotografien eines Referenzmodells gemacht werden, bei denen das Modell die Gelenke an beispielsweise Armen und Beinen einmal anwinkelt und einmal entspannt hält. Für den Film „Terminator 3", für den ein sogenanntes „Digital Double", also eine digitale Kopie, des Hauptdarstellers angefertigt wurde, wurde beispielsweise eine große Anzahl von Fotos des Darstellers gemacht, die ihn in den verschiedensten Extremstellungen zeigten (Quelle: Vortrag „Terminator 3" von Pablo Helman, VFX-Supervisor, Industrial Light & Magic im Rahmen der eDIT|VES 2003, 28.09.2003).

Dieses Vorgehen dient dazu, die genaue Position der Gelenke in den Extremitäten erfassen zu können, um absolut überzeugend wirkende Bewegungen zu ermöglichen.

Andererseits ist zu bedenken, dass in machen Fällen die Positionen der echten Knochen und Muskeln nur als Anregung zu verstehen sein sollte, nicht unbedingt als Einschränkung (Abb. 5.1). Als Beispiel sind hier etwa die Positionen der Rippen beim Charakter „Willy“ zu nennen. Hier wurden, anders als die Realität es vorgab, die Rippen über fast die komplette Länge des Rückgrats angelegt, um auch den hinteren Bereichen des Körpers einen Halt für das Enveloping zu geben. Für ein realistisches Verhalten der Rippen beim Atmen dürfen die Rippen dabei natürlich nur in Höhe des Brustkorbes bewegbar sein (vergleiche Kapitel 5.5.1, „Wirbelsäule und Brustkorb“). Rotation Limits wurden im Übrigen nicht eingesetzt, die meisten Quellen empfehlen deren Verwendung nicht, weil sie unter Umständen auch ansonsten mögliche und sinnvolle Bewegungen unterdrücken können und weil deren Anwendung zu Unsauberkeiten in der Animation führen kann (Rossano2002, Maraffi2001).

Abb. 5.1: Skelett eines echten Hundes (links) und konstruiertes Skelett von „Willy" (rechts) (Quelle für linke Abb.: Rosa2003)

Das Skelett des Charakters sollte, während es Stück für Stück aus Konstruktionen von Hierarchien und Constraints aufgebaut wird, dabei regelmäßig auch schon in Bewegung gestestet werden. Zwar kann durch sorgfältige Planung der Arbeit viel Zeit und Energie gespart werden, viele Feinheiten können jedoch häufig erst während der Arbeit durch Überprüfen der resultierenden Bewegungen optimiert werden.

Um Kinematikketten miteinander zu verbinden, muss prinzipiell nur der Effektor der übergeordneten Kette den Ursprung der untergeordneten steuern. Dazu kann der Ursprung entweder als Child des Effektors eingesetzt werden, oder durch ein Constraint mit diesem verbunden werden.

Hilfsobjekte, Steuerungsobjekte und Deformer-Objekte wurden bei „Willy“ größtenteils in eine einzige Hierarchie integriert, um einen übermäßigen Einsatz von Constraints zu vermeiden, der unter Anderem die Übersichtlichkeit verschlechtern kann. Außerdem kann durch die Verwendung von Parent-Child-Beziehungen der Vorteil der lokalen Koordinaten genutzt werden. Diese Hierarchie hat ihren Ursprung im sogenannten „COG-Steuerungsobjekt“, wobei COG für „Center of Geometry“ steht. Mit diesem Steuerungsobjekt kann das Rig in seiner Gesamtheit rotiert und verschoben werden.

Einige Teile des Rigs, die besonders gut zu separieren waren, wurden jedoch von der Hierarchie der Steuerungsobjekte getrennt und in einer separaten „HelperHierarchie“ abgelegt. Dies betraf vor allem diverse IK-Elemente und Hilfsobjekte und diente dazu, die Hierarchie etwas „abzuflachen", eine zu umfangreiche Hierarchie verschlechtert die Übersichtlichkeit wieder.

Bei der Konstruktion ist weiterhin zu beachten, dass noch keine Steuerungsobjekte direkt eingebaut werden sollten, sondern zunächst nur Null-Objekte. Alle Elemente, die später einmal mit den betreffenden Steuerungsobjekten verbunden sein sollen, sollten also zunächst an den entsprechenden Nulls verankert werden. Später werden dann die Steuerungsobjekte wiederum an den Nulls verankert. Durch dieses Vorgehen kann während der Entwicklung flexibler gearbeitet werden und beispielsweise auch die Steuerungsobjekte leicht nachträglich ausgetauscht werden (Kapitel 5.3.2, „Die Steuerungsobjekte“).

Eventuell können bereits während der Entwicklung verschiedene Gruppen gebildet werden, in welchen die verschiedenen Elemente des Rigs, wie etwa Deformer-Objekte, Hilfsobjekte oder Steuerungsobjekte einsortiert werden. Durch Verändern der Parameter für die Sichtbarkeit dieser Gruppen kann die Szene übersichtlich gehalten werden. Später wird die Sichtbarkeit dieser Gruppen durch die Synoptic View gesteuert. Neben der Methode des Einteilens in Gruppen wäre auch eine Organisation in „Layern“ denkbar gewesen. Dies hat aber den Nachteil, dass diese Layer dann für jedes in eine Szene importierte derartige Rig in der betreffenden Szene mit hinein importiert werden würden, was die Übersichtlichkeit, speziell bei mehreren Rigs innerhalb einer Szene, wieder verringern würde.

Wenn bereits Animationen vergleichbarer Charaktere vorliegen und möglicherweise verwendet werden sollen, sollte von Anfang an auf eine Kompatibilität des Rigs mit dem entsprechenden bereits animierten Rig geachtet werden. Dies kann entweder eines der beiden Standardskelette von Softimage|XSI oder eine eventuelle hausinterne Konstruktion sein. Die Befragung von Experten im Rahmen dieser Arbeit hat allerdings ergeben, dass in den meisten Fällen ein Rig auf eine spezielle Anwendung hin erstellt wird („Anhang B: Ausgewählte Expertenmeinungen“116).

5.2.2 Überlegungen zum Enveloping und zum Weighting

Während der Entwicklung der Skelettstruktur sollte das Modell selbst, also die Geometrie, schon zumindest grob existieren. Dadurch kann während der Entwicklung der Skelettstruktur immer schon die Anzahl und die Position der Deformer-Objekte kontrolliert werden.

Beim Einbinden der Deformer-Objekte ist eine grundsätzliche Überlegung bezüglich der Anzahl dieser, besser gesagt der Dichte dieser, anzustellen. Je mehr Deformer-Objekte beim Enveloping berücksichtigt werden, desto besser wird im Allgemeinen das standardmäßige Weighting, das XSI beim ersten Zuweisen der Hüllobjekte zu den De-

former-Objekten berechnet, desto weniger muss also auch manuell am Weighting ausgebessert werden.

Allerdings wird das Überarbeiten des Weightings, das dann doch manuell durchgeführt werden muss, mit steigender Anzahl von Deformer-Objekten zunehmend aufwändiger, unübersichtlicher und schwieriger. Mehr Deformer-Objekte führen außerdem zu einer Verschlechterung der Performance beim Animieren. Besonders bei einer Kombination aus hoher Dichte von Deformer-Objekten und hoher Geometrieauflösung ergibt sich des Weiteren ein Problem damit, weiche Übergänge zwischen den Gewichtungen auf der Hülle zu bekommen. Zu abrupte Übergänge können zu Brüchen des Envelope-Objektes während der Bewegung führen (Abb. 5.2). Im Falle von „Willy“ wurde speziell dieses Problem dadurch gelöst, dass das Hüllobjekt zum Enveloping genutzt wurde und eine ausreichend hohe Geometrieauflösung durch „Geometry Approximation“ erreicht wurde (vergleiche dazu auch Kapitel 3.3.2, „Effizienzüberlegungen“).

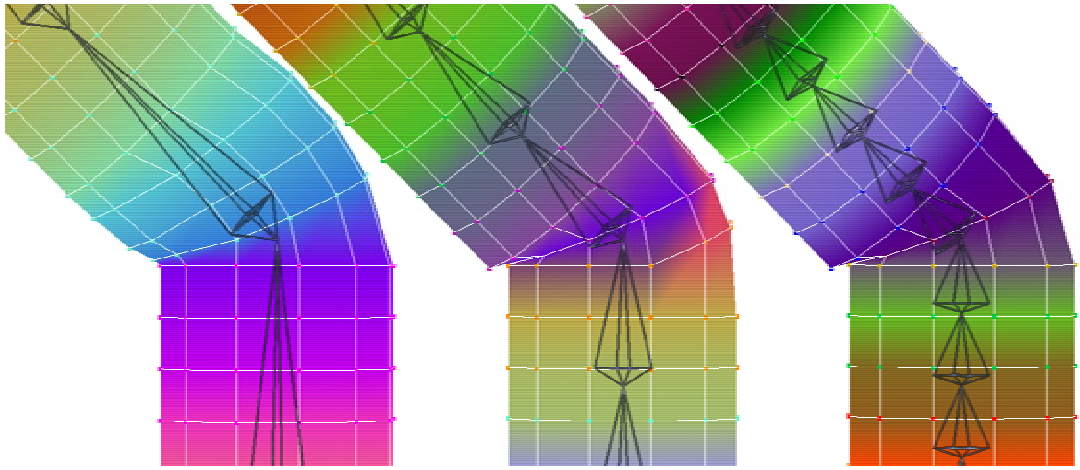

Abb. 5.2: Die Auswirkungen einer größeren Anzahl von Deformer-Objekten. Gut ist hier die zunehmende Härte des Überganges zwischen den Einflußbereichen einzelner Deformer-Objekte zu erkennen, die zu „Brüchen“ der Oberfläche des Envelope-Objektes führt

Auf der anderen Seite dürfen auch auf keinen Fall zu wenige Deformer-Objekte eingesetzt werden. Neben der bereits erwähnten, schlechteren Ergebnisse beim standardmäßigen Weighting würden dann die Fähigkeiten des Rigs unter Umständen nicht voll ausgenutzt.

Für den Charakter „Willy“ wurden relativ viele Deformer-Objekte eingesetzt, dadurch konnte das manuelle Weighting auf ein Minimum reduziert werden. Dies geschah auch vor dem Hintergrund, dass das erstellte Rig leicht für andere Charaktere wiederzuverwenden sein sollte und der Arbeitsaufwand, der zum Anpassen des Rigs nötig ist, minimiert werden sollte. Für die Anzahl der Deformer-Objekte, die in die Berechnung der Standardgewichtung für einen bestimmten Punkt der Geometrie einbezogen wird, wurde ein Wert von drei angegeben.

Nach dem Weighting hat es sich bewährt, den sogenannten „Smooth Envelope Weights“-Operator anzuwenden, dadurch kann der für das Weighting benötigte Arbeitsaufwand weiter verringert werden (XsiDoc2002, Animation).

Zähne und Krallen wurden, anders als der Körper, nicht per flexible Enveloping an Deformer-Objekte gebunden, sondern durch eine Parent-Child-Beziehung starr an die entsprechenden Knochen, also Pfoten und Schnauze, gebunden. Dadurch, dass die Gewichtung der direkt angrenzenden Geometrie des Körpers zu nahezu 100% diesen Deformer-Objekten zugewiesen ist, ist das Ergebnis das gleiche.

Es sollten auch zur Vereinfachung des Envelopings verschiedene Gruppen gebildet werden, ähnlich und teilweise überlappend denen, die zur besseren Übersicht über die Konstruktion dienen. Durch solche Gruppen für deformierende und deformierte Elemente wird eine einfache Selektion der entsprechenden Komponenten beim Enveloping ermöglicht, was unter Umständen während der Entwicklung zahlreiche Male nötig sein wird.

5.3 Das Control-Rig

5.3.1 Generelle Vorgehensweise

Da die Steuerung des Charakters sowohl über Steuerungsobjekte als auch über Slider möglich sein sollte, und da das Rig leicht auf andere Charaktere skalierbar und ergänzbar sein sollte, wurde im Rahmen dieser Arbeit das Konzept einer „unteren Steuerungsebene“ entwickelt, die das Rig sozusagen „unter der Oberfläche“ steuert. Auf diese Steuerungsebene greift der Benutzer nicht direkt zu, sondern über eine der möglichen Steuerungsmethoden die dafür erstellt wurde.

Da das Arbeiten mit Expressions, die zu einem großen Teil zur Steuerung des Rigs verwendet wurden, gewisse Einschränkungen mit sich bringt, wie etwa die fehlende Variablen- oder Konstantendeklaration, den beschränkten Umfang und die beschränkte Übersicht über komplexe Ausdrücke, wurden drei verschiedene Klassen für Custom Parameter konstruiert. Die strenge Einhaltung der speziellen Eigenschaften und Beschränkungen dieser Parameterklassen sowie der Nomenklatur musste dabei dem Entwickler überlassen werden.

Die Parameter, die der angesprochenen untersten Steuerungsebene angehören und die später direkt durch eine der Steuerungsmethoden angesprochen werden, wurden der Klasse „Variablen“ zugeordnet. Sowohl die Namen der Parametersets, die diese Parameter enthalten, als auch die Namen der Parameter selbst wurden zur Kenntlichmachung mit dem Präfix „var_“ versehen.

Der Präfix „const_“ wurde den Parametern und Parametersets der Klasse „Konstanten“ zugeordnet. Die betreffenden Parameter sind in Softimage|XSI als „nicht animierbar“ gekennzeichnet, ihre primäre Funktion ist die Bereitstellung von konstanten Werten für diverse Berechnungen zur Bewegung des Rigs. Editiert werden sollten diese Werte nur zur Anpassung des Rigs an andere Charaktere.

Die letzte Parameterklasse dient der Vereinfachung gewisser Zusammenhänge, die Klasse „Interne Variablen“ dient als eine Art Zwischenspeicher, um komplexe Ausdrü-

cke sinnvoll zu unterteilen. Sie sind ausnahmslos mit Expressions belegt und können daher nicht direkt animiert werden. Der Präfix für diese Klasse von Parametern wurde als „int_“ definiert.

Eine Auflistung aller im Rahmen dieser Arbeit verwendeten Parameter, unterteilt nach den verschiedenen Klassen, kann im Anhang zu der Arbeit gefunden werden (Anhang C: Parametertabellen).

5.3.2 Die Steuerungsobjekte

Die Steuerungsobjekte wurden während der Entwicklung zunächst noch nicht direkt in das Rig implementiert, an ihrer Stelle wurden Nullobjekte in die Hierarchie eingebunden. Auch alle Referenzen auf etwa Positionen oder Orientierungen von Steuerungsobjekten innerhalb von Expression werden nicht direkt von den Steuerungsobjekten abgenommen, sondern von den dazugehörigen Null-Objekten. Die einzige Ausnahme hierbei wird bei den Steuerungsobjekten für den Schwanz gemacht, hier müssen die zwei Variablen für die horizontale und vertikale Bewegung direkt von den Steuerungsobjekten genommen werden, da das hier verwendete „Frame_At“-Kommando nicht mit Expression sondern nur mit Funktionskurven arbeiten kann (Kapitel 5.5.4).

Die angesprochenen Null-Objekte wurden dann später als Children in eine Beziehung zu den eigentlichen Steuerungsobjekten gestellt (Abb. 5.3). Dieses Vorgehen erlaubte eine flexiblere Arbeitsweise während der Entwicklung, da beispielsweise die Steuerungsobjekte jederzeit ausgetauscht werden konnten.

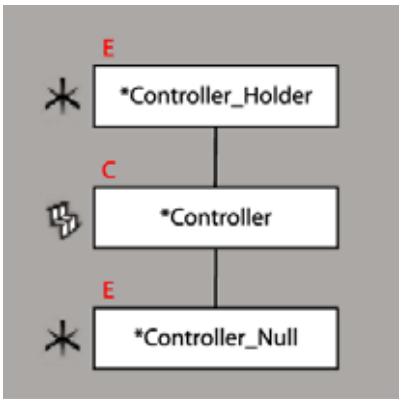

Abb. 5.3: Steuerungsobjekt-Hierarchie, Nomenklatur

Die Steuerungsobjekte wiederum sind selbst Children anderer Null-Objkete: Null-Objekte, denen als Name der Name des entsprechenden Steuerungsobjektes zuzüglich des Suffixes „_Holder“ zugewiesen wurden, dienen als Parents. Sie haben die gleichen Transformationswerte wie ihre Steuerungsobjekte und dienen dazu, die Standardpositionen und –rotationen der Steuerungsobjekte zu definieren. Diese können durch diese Konstruktion jederzeit zurückgesetzt werden, indem einfach alle lokalen Transformationswerte auf null gesetzt werden. Dies erleichtert die Animation mit dem Rig ernorm. Es wurde durch Expressions mit konstanten Werten an den lokalen Transformationsparametern der jeweilige Holder sichergestellt, dass es nicht möglich ist, das Rig versehentlich zu verändern, selbst wenn die Child Compensation einmal aktiviert sein sollte.

Die Steuerungsobjekte selbst sind Polygon-Objekte, für deren Darstellungsart der Drahtgitter-Modus festgelegt wurde, damit sie in der Darstellung in einem Viewport im schattierten Modus nicht die Sicht auf den Charakter verdecken. Da das Anzeigen der individuellen Darstellungsart von Objekten bei Softimage|XSI häufig versehentlich durch das sogenannte „Override Objects Property“ unterdrückt wird, wurde später eine

Funktion in der zum Rig gehörenden Synoptic View implementiert, welche die „Override Objects Property“ wieder deaktiviert (vergleiche Kapitel 6.5.3, „Untermenü ‚Face’“). Außerdem wurden alle Steuerungsobjekte als nicht renderbar gekennzeichnet.

Als Form der meisten Steuerungsobjekte wurde die Form einer umgekehrten Pyramide gewählt, da durch diese Form der Rotationsursprung, der sich an der Spitze der Pyramide befindet, nicht verdeckt wird (Abb. 5.4). Eine häufig gewählte Alternative für die Verwendung von Polygon-Objekten als Steuerungsobjekte wäre im Übrigen die Wahl von sogenannten „Implicits“ gewesen, das sind geometrische Grundformen, deren Darstellung auf einer mathematischen Formel beruhen. Diese sind von Natur aus nicht schattiert oder gerendert darzustellen, ihre Form kann allerdings nicht editiert werden. Eine weitere Möglichkeit wäre die Konstruktion von Steuerungsobjekten aus Kurvenobjekten gewesen. Auch diese sind nicht schattiert oder gerendert darzustellen, eine entsprechende Konstruktion aus einem einzigen Kurvenobjekt herzustellen, wäre aber ungleich aufwändiger gewesen.

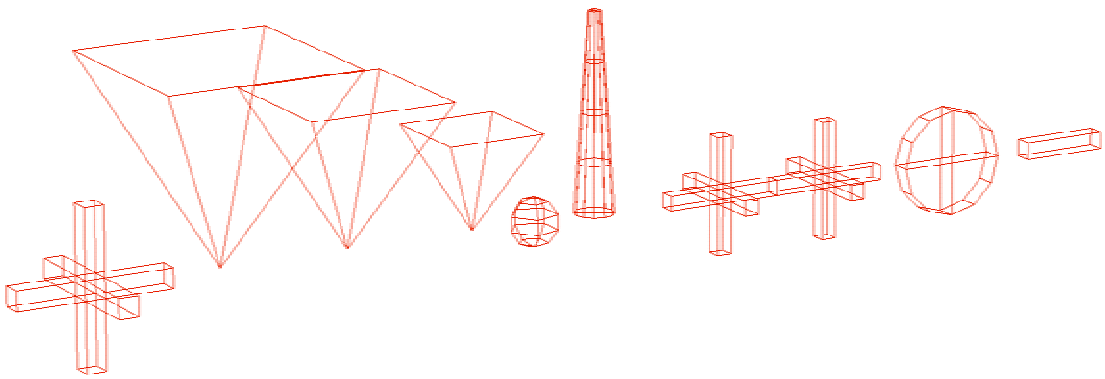

Abb. 5.4: Die unterschiedlichen Formen der verwendeten Steuerungsobjekte

Der besseren Übersicht diente es auch, die Steuerungsobjekte in drei Gruppen zu unterteilen, denen unterschiedliche Farben für die Darstellung im Drahtgitter-Modus zugewiesen wurden. Die erste Gruppe umfasst alle Steuerungsobjekte, die zur Steuerung des Gesichtes dienen. Die Steuerungsobjekte zur Steuerung des restlichen Körpers wurde ihrer Wichtigkeit nach in die beiden anderen Gruppen eingeteilt (Abb. 5.5).

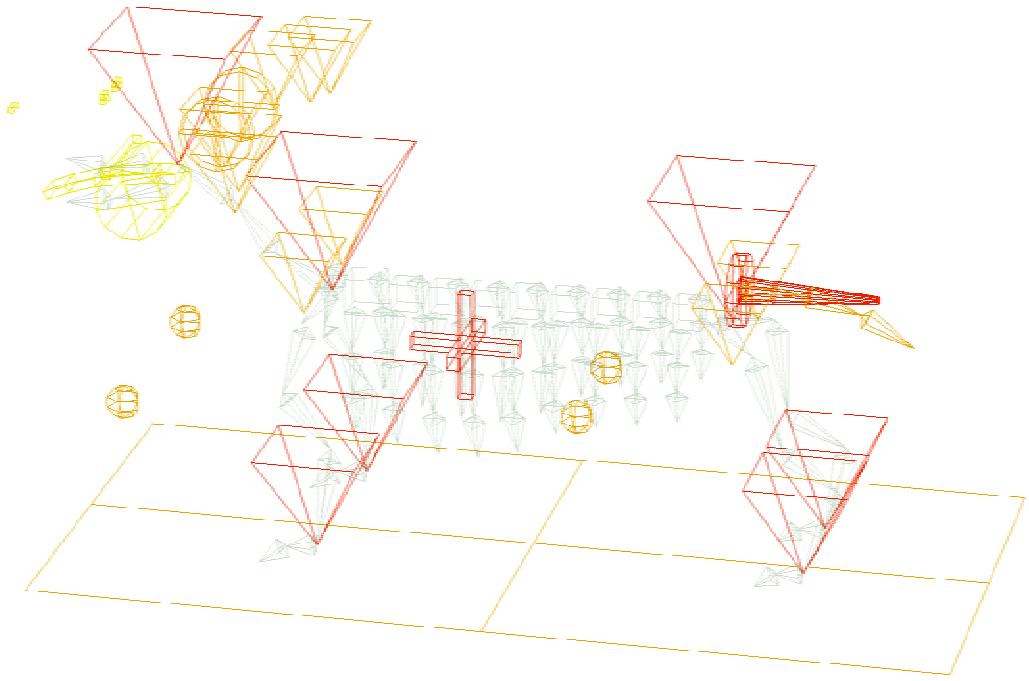

Abb. 5.5: Die Positionen der Steuerungsobjekte im Rig

Um das Arbeiten mit den Steuerungsobjekten für den Animator zu vereinfachen, wurden die Steuerungsobjekte mit Position- und Rotation-Limits versehen, die alle nicht vorgesehenen Transformationen blockieren. Außerdem wurden sogenannte „Marking Sets“ und „Transformation Setups“ zugewiesen. Marking Sets dienen dabei als Referenz, welche Parameter bei Aufruf des Kommandos „SaveKey“ mit einem Key versehen werden sollen und erleichtern damit auch den Aufruf des entsprechenden Kommandos aus der Synoptic-View. Außerdem bilden sie die alternative Slider-Ebene für jedes Steuerungsobjekt (siehe Kapitel 6.3, „Realisierung der zwei möglichen Steuerungsmethoden“). Mit „Transform Setups“ lassen sich dagegen Transformationstools als Standard definieren, die bei Selektion des entsprechenden Objektes automatisch aktiviert werden. So wurde beispielsweise für das Steuerungsobjekt für den Schwanz von „Willy“ festgelegt, dass bei Selektion automatisch das Rotationstool von Softimage|XSI aufgerufen wird, voreingestellt auf den lokalen Transformationsmodus nur für die beiden Koordinatenachsen, um die sich das Objekt auch rotieren lässt (XsiDoc2002, Fundamentals).

Bezüglich der Bewegungsfreiheit der Steuerungsobjekte fiel die Entscheidung, dass sämtliche Steuerungsobjekte für den Kopf und das Gesicht des Rigs über ein zwischengeschaltetes Null-Objekt immer am Hals des Rigs befestigt bleibt und nicht weiter von der Brust wegbewegt werden kann, als der Hals des Tieres lang ist. Eine ähnliche Entscheidung fiel für die Steuerungsobjekte des Schwanzes, die an der Hüfte befestigt wurden.

5.3.3 Parametrisierung

Der Begriff der „Parametrisierung“ meint in diesem Zusammenhang das Zusammenfassen vieler Parameter, welche die Bewegung des Rigs beeinflussen, zu möglichst Wenigen, die den gleichen oder zumindest einen ähnlichen Einfluss auf das Rig haben.

Sowohl Parametrisierung als auch Automation haben das Ziel, den Arbeitsaufwand für den Animator zu verringern. Während die Automation aber versucht, die Anzahl notwendiger Keys zu verringern (vergleiche Kapitel 5.3.5), ist mit der Parametrisierung das Vermindern der Parameteranzahl selbst gemeint.

Ein gutes Beispiel dafür wurde im Rahmen dieser Arbeit etwa mit dem Brustkorb des Rigs erstellt; Ein einziger Parameter steuert hier die Ausdehnung des Brustkorbes bei der Atmung, indem er die Rotation aller Rippen steuert. Weitere Beispiele für Parametrisierungen werden in Kapitel 5.5 besprochen.

Die Parametrisierung betrifft aber nicht nur animierbare Parameter, auch bei Parametern der Klasse „Konstanten“ wurde versucht, die Anzahl der Parameter zu verringern, um die Anpassung des Rigs an andere Charaktere zu vereinfachen. Damit liefert die Parametrisierung hier einen Beitrag zur Skalierbarkeit (siehe Kapitel 5.3.4). Auch hier kann der Brustkorb von „Willy“ als Beispiel dienen, hier definiert ein einziger Parameter die Länge der beiden Rippen eines Rippenpaares, ein weiterer den Rotationsspielraum dieses Paares bei der Atmung.

5.3.4 Skalierbarkeit

Mit „Skalierbarkeit“ ist die Möglichkeit gemeint, das fertige Rig im Anschluss an die Arbeit eventuell an andere Charaktere anzupassen, also etwa an andere anatomische Verhältnisse oder gegebenenfalls sogar Bewegungsabläufe.

Besonders im Zusammenhang mit der Skalierbarkeit wurde die Parameterklasse „Konstanten“ entwickelt, die diejenigen Parameter enthält, die einer einfachen Anpassung sowohl der Länge einzelner Bones, als auch der Berechnung von Bewegungsabläufen dienen. Weiterhin ist die Länge des Rückgrats, des Halses und des Schwanzes sowie der Durchmesser des Schädels skalierbar.

Veränderungen der Positionen von Roots von kinematischen Ketten müssen jedoch durch Translation beziehungsweise Rotation der betreffenden Objekte durchgeführt werden. Veränderungen der bevorzugten Winkel von Kinematikketten müssen weiterhin mithilfe des „Move Joint/Branch“-Tools ausgeführt werden.

Um die Standardpositionen der Steuerungsobjekte zu verändern, müssen die entsprechenden Parents (Suffix: „_Holder“) angepasst werden (vergleiche Kapitel 5.3.2, „Die Steuerungsobjekte“63). Um dies zu vereinfachen wurden auch diese in einer Gruppe zusammengefasst.

Weiterhin wurde auf Shape-Animationen am Modell selbst komplett verzichtet, da die sogenannten „Shape Keys“ für jedes Modell komplett neu erstellt werden müssten.

Stattdessen wurde eine alternative Methode eingesetzt, die einen Umweg über Deformer-Objekte geht, die an mit Shape Keys versehenen Kurven ausgerichtet sind (Kapitel 5.4, „Shape-Animationen“).

Die schon erwähnte Gruppe, die alle im Rig enthaltenen Deformer-Objekte enthält, erleichtert das Weighting anderer Modelle an das Rig, was auch der Skalierbarkeit dient.

In das fertige Modell des Rigs ist auch eine sogenannte „Annotation“, also eine Anmerkung, mit dem Namen „Setup-Help“ eingebunden, die in englischer Sprache die Namen der für die Anpassung benötigten Parametersets angibt, sowie weitere Hinweise zur Anpassung des Rigs gibt. Eine genauere Erklärung der entsprechenden Konstanten findet sich im Anhang zu dieser Arbeit („Tabelle C.2: Die verwendeten Konstanten“).

5.3.5 Automatisierung

Im Rahmen der Automatisierung von Bewegungsabläufen wird versucht, dem Animator Arbeit abzunehmen. Die Automatisierungen beziehen sich dabei hauptsächlich, aber nicht ausschließlich, auf die Sekundäranimationen (vergleiche Kapitel 2.3.7, „Primär- und Sekundäranimation“). Damit dienen Automatisierungen auch dazu, die Bewegungen eines Charakters wesentlich authentischer zu gestalten. Die bereits besprochenen Animationsstudien, speziell die Videoaufnahmen des lebenden Tieres, geben dabei den erwünschten Effekt aller Automatisierungen vor.

Bereits in den vorherigen Kapiteln wurde der Brustkorb von „Willy“ für eine Erläuterung der verschiedenen Zielsetzungen des Rigs herangezogen. Die Automatisierung in diesem Zusammenhang bezieht sich auf die Atmung. Es wurde eine Möglichkeit entwickelt, den Parameter, der die Ausdehnung des Brustkorbes steuert, wiederum durch eine Sinus-Funktion zu steuern. Der Benutzer kann durch Beeinflussen der Amplitude und der Frequenz der Funktion Einfluss auf die automatische Atmung nehmen. Indem er die Amplitude auf null reduziert, kann er die Automatisierung ausschalten und mithilfe eines dritten Parameters die Ausdehnung des Brustkorbes direkt steuern (Abb. 5.6).

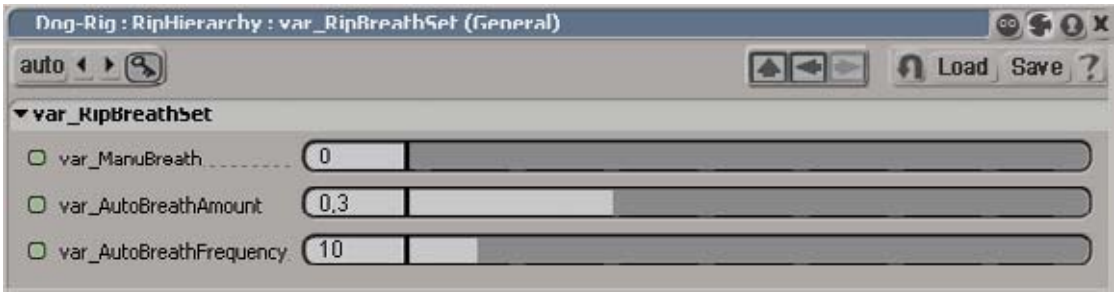

Abb. 5.6: Die drei Parameter, welche die Atmung steuern

Ein weiteres Beispiel für eine eingesetzte Automatisierung ist etwa die automatische Rotation der Hüfte und der Brust in Abhängigkeit der Pfoten. Diese und weitere Automatisierungen werden näher erläutert in Kapitel 5.5, „Das erstellte Control-Rig im Einzelnen“.

Wichtig im Zusammenhang mit Automationen ist, dass alle ihre Auswirkungen stufenlos skalierbar sein müssen, bis zur totalen Deaktivierung jeder Wirkung. Idealerweise sollte der entsprechende Parameter selbst animierbar sein. Es sollte immer bedacht werden, dass während der Konstruktion nie alle möglichen Anwendungen des Rigs geplant werden können. Nur auf diese Weise stehen dem Animator sämtliche Möglichkeiten des Rigs offen (Rossano2002).

Die Automatisierungen des hier besprochenen Rigs wurden hauptsächlich durch Expressions erreicht, die zum größten Teil Parameter der Klasse „Konstanten“ in die Berechnungen einbeziehen. Damit kann durch Verändern der betreffenden Parameter die Bewegung des Rigs angepasst werden.

Auch Parameter der Klasse „Interne Variablen“ fließen in die Berechnungen ein. Für beispielsweise die Automatisierungen, welche die Position und Rotation von Hüfte, Brust und Schulterknochen betreffen, wurde eine Messkonstruktion aus Null-Objekten gebaut, die mit Constraints an verschiedene Positionen an Hüfte, Brust und Beinen gebunden wurden. Durch eine Organisation dieser Null-Objekte in einer Hierarchie konnten verschiedene Entfernungen und Positionen der Null-Objekte zueinander leicht abgelesen werden und in Parametern zur weiteren Verwendung durch Expressions zwischengespeichert werden („Tabelle C.3: Die verwendeten internen Variablen“).

Einige der Automatisierungen des Rigs betreffen direkt die Steuerungsobjekte: So sind etwa die Steuerungsobjekte für die Schultern über das Distance-Constraint an die Steuerungsobjekte für die Hüfte und die Brust gebunden. Außerdem sind die Steuerungsobjekte für die Pfoten durch ein Bounding-Plane-Constraint an ein Objekt gebunden, das während der Animation den Boden simulieren kann und ein Durchdringen desselbigen durch die Pfoten verhindern soll.

5.4 Shape-Animationen für Gesichtsausdrücke

Für die Umsetzung von Gesichtsausdrücken und Ähnlichem wird im Allgemeinen die Technik der Shape-Animation eingesetzt. Die übliche Herangehensweise dabei ist, die Geometriekomponenten, die beeinflusst werden sollen, in Cluster zu gruppieren, und diese Cluster dann, durch Operatoren, die nicht die Topologie beeinflussen, derart zu verändern, dass sie die gewünschten Formen darstellen. Für diese Formen werden dann Shape Keys gespeichert und zwischen diesen dann mithilfe des Animation Mixers in Softimage|XSI interpoliert (Abb. 5.7).

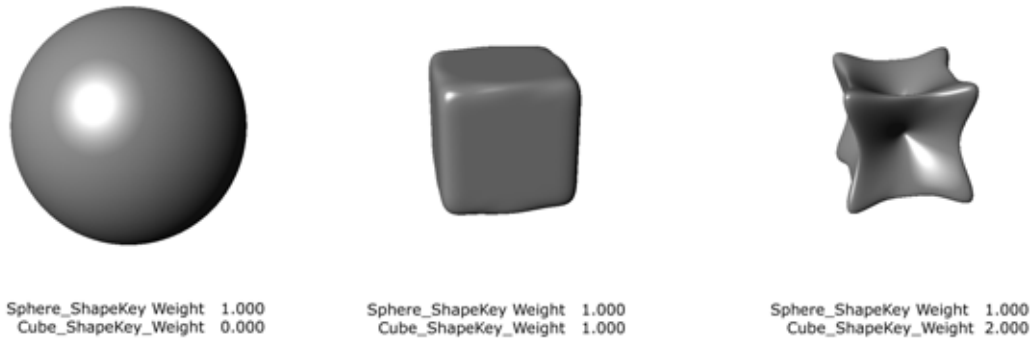

Abb. 5.7: Beispiel für eine Shape-Animation, hier dargestellt das Verhalten im Additiven Modus

In seinem Buch über Charakteranimation beschreibt A. Rossano eine alternative Methode, die im Rahmen dieser Arbeit in die Praxis umgesetzt und verwendet wurde (Rossano2002). Sie beruht darauf, nicht die Geometrie des Modells selbst durch Shape-Animation zu beeinflussen, sondern stattdessen Kurven, welche Abstraktionen der Gesichtsmuskulatur darstellen. Diese Kurven sind Teil des Rigs und beeinflussen die Geometrie nicht direkt, sondern über Deformer-Objekte.

Prinzipiell erlaubt es Softimage|XSI, jedes beliebige Objekt als Deformer-Objekt für einen Envelope zu verwenden. Bei allen Arten von Objekten, mit Ausnahme der Bones, wird allerdings die Form des Objektes für die Envelope-Berechnungen komplett vernachlässigt, hierfür werden allein die Transformationen des Koordinatenmittelpunktes des Objektes einbezogen. In diesem Fall war genau das gefragt, es wurden also keine Bones als Deformer verwendet, sondern Implicits für die, allein aus Gründen der Optik, die Kugel als Form gewählt wurde.

Diese Gruppe von Objekten wurde, weil sie eher Muskelpartien eines echten Hundes als Knochen nachbildeten, „Muscle-Deformer“ genannt. Sie wurden im Übrigen nicht ausschließlich als Hilfsmittel zur Shapeanimation im Gesichtsbereich eingesetzt, auch etwa für das Nachbilden des Schwanzes oder der Beinmuskulatur wurden sie eingesetzt.

Das konkrete Vorgehen bestand darin, zunächst die Kurven zu erstellen, an denen die Muscle-Deformer ausgerichtet werden sollten. Dabei konnte sich etwas an der Lage der Gesichtsmuskeln in einem menschlichen Gesicht orientiert werden, etwa was die kreisförmigen Muskeln um die Augen herum anging (Maraffi2001).

Insgesamt wurden bei dem Rig für „Willy“ nicht allzu viele Kurven verwendet, zwei pro Auge, eine für die Stirn und drei für den Bereich um die Schnauze. Das System ließe sich beliebig ausbauen, da allerdings ein relativ realistisches Aussehen des gesamten Charakters erwünscht war, wurde auf weitere Details verzichtet. Die Erfahrung zeigt, dass eine zu starke „Vermenschlichung“ der Gesichtsausdrücke von Tieren sehr leicht unrealistisch aussieht. Außerdem war keine Lippensynchronität des Charakters gefragt, die wesentlich mehr Ausdrucksmöglichkeiten speziell im Bereich der Schnauze nötig gemacht hätte (Maraffi2001).

Anschließend wurden die Muscle Deformer an den Kurven platziert. Dazu wurden einzelne Punkte der Kurven als Cluster gespeichert, anschließend wurden die Muscle-Deformer mithilfe des „Object to Cluster“-Constraint an diese Cluster gebunden.

Die Kurven wurden nun kopiert und jede Kopie in eine der gewünschten Formen gebracht, die später die Shape Keys ergeben sollten. Dies ermöglichte dann, die Shape Keys mithilfe des Kommandos „Select Shape" zu erstellen und mit den Kopien der Kurven zu verlinken, so dass die Kopien der Kurven beibehalten und an andere Modelle angepasst werden können.

Alle Shape Keys konnten nun im Animationsmixer als Clips instanziiert und arrangiert werden. Im Mixer des Modells wurde die Normalisierung ausgeschaltet, er wurde also auf die Arbeit im Additiven Modus eingestellt. Dieser erlaubt eine größere Vielfalt von Formen und größeren gestalterischen Einfluss, weshalb er für Shape-Animationen besser geeignet ist. Anschließend wurden noch simple Ausdrücke erstellt, welche die Gewichtungen aller Shape-Clips jeder Kurve nur noch von einem Parameter abhängig machten (Maraffi2001).

Das Enveloping der Muscle-Deformer kann gemeinsam mit allen anderen Deformern durchgeführt werden (Abb. 5.8). Da die automatische Gewichtung im Normalfall anhand der Entfernung der Punkte der Geometrie zu den Deformern entscheidet, welche Deformer auf einen Punkt einwirken, müssen noch diverse Muscle-Deformer quasi als „Gegengewicht“ im Gesicht und entlang des Schädels verteilt werden, die nicht durch die Kurven animiert werden. Ansonsten würden animierte Muscle-Deformer nach der Standard-Gewichtung einen zu großen Einfluss auf die Geometrie des Kopfbereiches nehmen. Um diese zusätzlichen Muscle-Deformer zu platzieren, wurde aus einem Polygon-Objekt ein simples Modell eines Schädels konstruiert, an welchem sie mithilfe von „Object to Cluster“-Constraints gebunden werden.

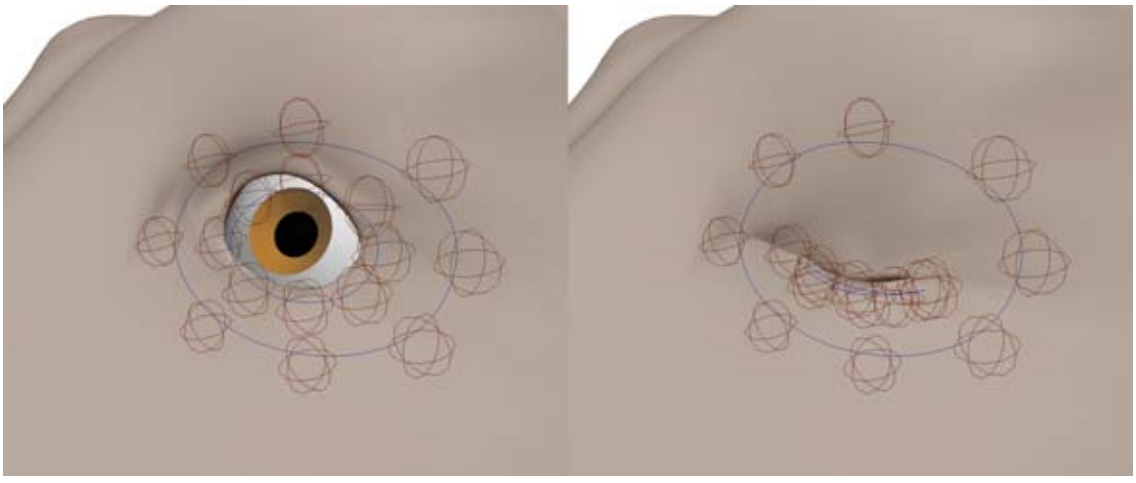

Abb. 5.8: Animierte Kurve für die Bewegung eines Augenlids

Diese Methode, die Geometrie also nicht direkt zu modifizieren sondern Deformer-Objekte zu verwenden, hat den Vorteil, dass sie es ungemein erleichtert, das Rig auf andere Modelle zu übertragen. Während bei der ursprünglichen Methode die verschiedenen Shape Keys jedes Mal erneut modelliert werden müssten, kann die Gesichtsmuskulatur nun mit dem üblichen Enveloping, das sowieso durchgeführt werden muss, auf das Modell übertragen werden. Die Shape-Animation funktioniert weiterhin, auch wenn die Kurven transformiert werden, um beispielsweise ihre Lage im Gesicht des Hundes zu verändern. Selbst im Extremfall, wenn die Ausdrücke selbst sich unterscheiden, müssen nur die Kopien der Kurven, die mit den Shape Keys verlinkt sind, angepasst werden.

Die Methode erlaubt außerdem eine Vorschau der Animation ohne Darstellung der Geometrie. Wenn auf langsameren Systemen animiert wird, auf denen die Darstellung der Geometriedeformation in Echtzeit nicht möglich ist, kann mit dieser Methode zumindest die Animation der einzelnen Kurven beobachtet werden. Ein entscheidender Nachteil der Methode kann sich allerdings ergeben, wenn sehr detaillierte Gesichtsausdrücke erwünscht sind. In diesem Fall sind die Möglichkeiten, gestalterischen Einfluss zu nehmen, bei dieser Methode doch etwas begrenzt. Es empfiehlt sich in diesem Fall, die Geometrie selbst in die gewünschten Formen zu bringen.

Zu beachten ist beim Zusammenhang von Shape-Animationen mit referenzierten Modellen, in denen ja explizit angegeben werden kann, welche Einstellungen betreffend der Modelle in der Szene mitgespeichert werden, dass der Animationsmixer dann nicht in einer Szene mitgespeichert werden darf, da sonst alle zur Steuerung des Rigs vordefinierten Shape-Tracks überschrieben werden. Dies heißt leider auch, dass die Länge der einzelnen Shape-Clips nicht auf herkömmliche Weise im Animationsmixer an die Länge der neuen Szene angepasst werden kann. Die Möglichkeit, alle Shape-Clips auf eine sehr große Länge zu skalieren, die in der Praxis kaum erreicht werden dürfte, hat sich ebenfalls als ungeeignet erwiesen, unter Umständen kann dabei nämlich die Darstellungsgeschwindigkeit ernorm sinken.

Allerdings kann die Länge von Shape-Clips in einem Skript skaliert werden, ohne dass ein Aktivieren des Speicherns der Mixer von referenzierten Modellen erforderlich ist. Aufgrund dessen wurde in der Synoptic View eine Funktion implementiert, welche bei ihrem Aufruf die Shape-Clips auf die volle Länge der aktuellen Szene skaliert (Kapitel 6.5.3).

5.5 Das erstellte Control-Rig im Einzelnen

5.5.1 Wirbelsäule und Brustkorb

Prinzipiell wird die Animation des Rückgrats nur durch Position und Rotation der Steuerungsobjekte für Brust und Hüfte gesteuert. Dabei ist das von Softimage|XSI zur Verfügung gestellte „Spine-Tool“, das automatisch eine Konstruktion aufbaut, die zwei gegebene Referenzobjekte mit einer Kurve verbindet. Die entstandene Kurve wird neben den Translationen der Referenzobjekte durch zwei Null-Objekte definiert, die den Einflussbereich der Referenzobjekte angeben. Diese Null-Objekte konnten in diesem Fall einmalig relativ zu Brust und Hüfte platziert und anschließend versteckt werden, eine weitere Justierung war nicht nötig (Abb. 5.9).

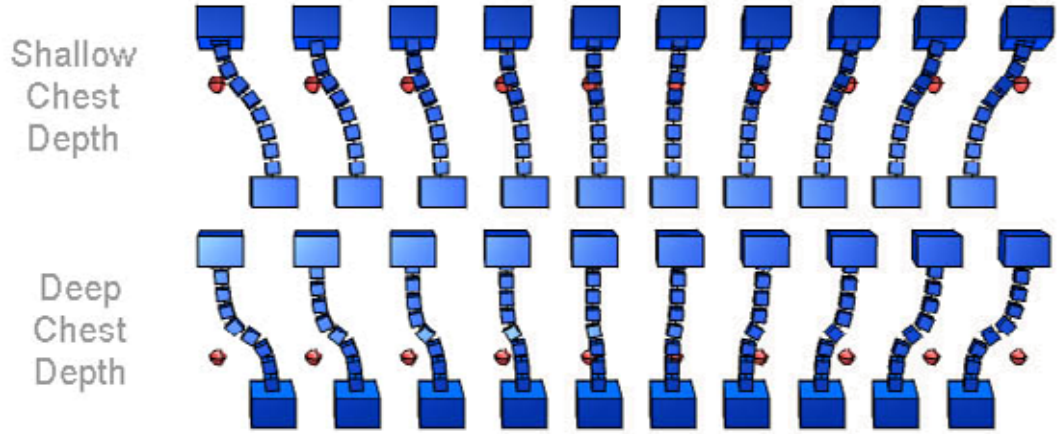

Abb. 5.9: Auswirkungen der Objekte, welche den Einflussbereich der Hüfte und der Brust angeben (Quelle: Misner2003b)

Entlang der Kurve werden automatisch Objekte platziert, die als Wirbel dienen und die auch, mit einem anteilsmäßig über die Kurve verlaufenden Rotationsgrad um sich selbst rotieren, wenn es die Referenzobjekte tun. Dadurch kann eine Drehung der gesamten Wirbelsäule erreicht werden (Abb. 5.10, Misner2003b).

Abb. 5.10: Rotation der Wirbel um die Wirbelsäule (Quelle: Misner2003b)

Mit einem Constraint an die beiden äußersten Wirbel des Rückgrats gebunden sind die Konstruktionen des Schwanzes und des Halses. Sie bilden damit eine Fortsetzung des Rückgrats. Der Hals selbst ist ähnlich wie das Rückgrat aufgebaut.

Die Rippen, die den Brustkorb bilden und den gesamten Torso beim Enveloping stützen, wurden mithilfe des Kommandos „Create Chain from Curve“ erstellt. Dazu wurden zunächst Kurven erstellt, welche die Form der Rippen der linken Körperhälfte definieren. Diese wurden geklont und auf die rechte Körperseite gespiegelt. Dadurch, dass die

IK-Chains mit den Kurven verknüpft bleiben, genügt es, bei Anpassung der Rippenform auf einen anderen Charakter für ein Rippenpaar jeweils eine Kurve anzupassen.

Die Länge jedes Knochens eines Rippenpaares wird durch einen Parameter des Types „Konstante“ gesteuert, das Parameterset, in dem alle diese Konstanten zusammengefasst sind, ermöglicht damit eine weitere Anpassung der Form des Brustkorbes.

Die Atmung, also die Ausdehnung des Brustkorbes kann durch eine Rotation der Kurven, an die Chains gebunden sind, erreicht werden. Der Rahmen, in welchem jedes Rippenpaar bei der Atmung um das Rückgrat rotiert, ist wiederum durch Parameter des Types „Konstanten“ definiert.

Die Rippen wurden dabei entlang der gesamten Wirbelsäule platziert, um den kompletten Torso beim Enveloping zu „halten“ und eine etwa gleichbleibende Dichte von Deformer-Objekten zu gewährleisten. Um eine realistische Atmung zu erhalten werden für die Ausdehnung des Brustkorbes aber nur die vorderen Rippen auf Höhe des Brustkorbes beeinflusst. Bei allen anderen Rippen ist der entsprechende Parameter auf einen sehr kleinen Wert beziehungsweise auf Null gesetzt (Abb. 5.11).

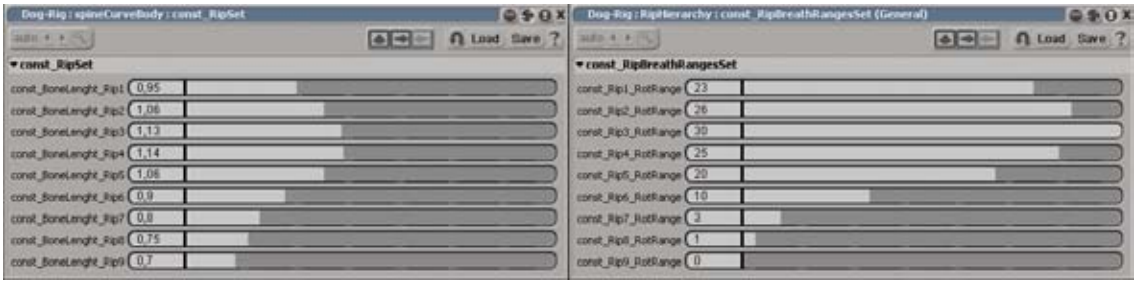

Abb. 5.11: Parametersets für die Konstanten zur Definition der Länge der Rippen (links) und des Rotationsbereiches der Rippen bei der Atmung (rechts)

In Kapitel 5.3.5 wurde die Atmung bereits als Beispiel für eine sinnvolle Automatisierung angegeben. Zur Umsetzung dieser Automation wurde ein Parameter der Klasse „Intern“ erstellt, der effektiv die Rotation der Wirbel steuert. Dieser interne Parameter wiederum wird durch drei Parameter der Klasse „Variablen“ gesteuert.

Die Parameter für die Amplitude („var_AutoBreathAmount“) und die Frequenz („var_AutoBreathFrequency“) steuern die automatische Atmung, der dritte Parameter („var_ManuBreath“) gibt als Offset die Möglichkeit der manuellen Steuerung der Ausdehnung des Brustkorbes. Indem der Benutzer die Amplitude auf null reduziert, kann er die Automatisierung ausschalten und mithilfe dieses dritten Parameters die Ausdehnung des Brustkorbes nur manuell steuern.

Für die Erzeugung periodischer Signale empfiehlt sich natürlich die Verwendung einer Sinus- oder Kosinus-Funktion, in diesem Fall wurde eine Sinusfunktion zur Steuerung der absoluten Ausdehnung des Brustkorbes in Abhängigkeit der Zeit, beziehungsweise des aktuellen Frames, und der drei oben angegebenen Parameter verwendet:

Expression für den internen Parameter, der die Ausdehnung des Brustkorbes steuert:

```
[ Dog-Rig.RipHierarchy.internal_RipBreathSet.
internal_RipBreath = ]
( Dog-Rig.RipHierarchy.var_RipBreathSet.
var_ManuBreath * 0.7 )
+ ( Dog-Rig.RipHierarchy.var_RipBreathSet.
var_AutoBreathAmount )
+ sinus( Dog-Rig.RipHierarchy.var_RipBreathSet.
var_AutoBreathFrequency, Dog-Rig.RipHierarchy.
var_RipBreathSet.var_AutoBreathAmount, 0 )
```

Für die Umsetzung dieses Parameters auf die Rippen wurden die Kurven mit folgender Expression belegt (hier beispielhaft nur das erste Rippenpaar):

Expression für die Kurven, welche die Rippen der linken Seite definieren:

```
[ Dog-Rig.Rip1-l.kine.local.roty = ]
( 90 + ( Dog-Rig.RipHierarchy.
const_RipBreathRanges.const_Rip1_RotRange / 2 ) )
- ( Dog-Rig.RipHierarchy.var_RipBreathset.var_RipBreath
* Dog-Rig.RipHierarchy.const_RipBreathRanges.
const_Rip1_RotRange )
```

Expression für die Kurven, welche die Rippen der rechten Seite definieren:

```
[Dog-Rig.Rip1-r.kine.local.roty = ]
( 90 - ( Dog-Rig.RipHierarchy.
const_RipBreathRanges.const_Rip1_RotRange / 2 ) )
+ ( Dog-Rig.RipHierarchy.var_RipBreathset.var_RipBreath *
Dog-Rig.RipHierarchy.const_RipBreathRanges.
const_Rip1_RotRange )
```

5.5.2 Brust und Hüfte

Mit Brust und Hüfte sind im Rahmen dieser Arbeit die, entsprechend dem Terminus der Graphischen Datenverarbeitung, „Roots“, der Vorderläufe als auch der Hinterläufe gemeint. Im Zusammenhang des erstellten Rigs sind dies die beiden Enden des Rückgrats. Die Umsetzung der Automationen der Hüft- und Brustbewegungen erfolgt prinzipiell gleich, allein konstante Faktoren führen zu einem Unterschied in der Bewegung.

Die im Folgenden in diesem Unterkapitel beschriebenen Methoden zur Automation der Hüftbewegung sind teilweise in Anlehnung an die in den Büchern von A. Rossano und C. Maraffi erläuterten Methoden zur Steuerung der Hüfte bei Zweibeinern, genauer Menschen, entstanden, teilweise aus Überlegungen zur Anatomie von Hunden (Rossano2002, Maraffi2001).

Im Rahmen der Automation von Brust und Hüfte wurde eine interne Messkonstruktion aufgebaut („Dog-Rig.FeetMeasureHierarchy“), welche die Beziehungen der Pfoten zueinander misst. Dazu wurde mit Position-Constraints an die Brust, Hüfte und Pfoten jeweils ein Null-Objekt gebunden. Diese Null-Objekte wurden dann in einer Parent-Child-Hierarchie derart zueinander in Beziehung gesetzt, dass die benötigten Werte einfach an den lokalen Transformationen abgelesen und in Parametern des Typs „Intern“ zwischengespeichert werden können.

`Dog-Rig.FrontFeetMeasure_Anchor.internal_FrontFeetMeasureSet.`

beziehungsweise

`Dog-Rig.BackFeetMeasure_Anchor. internal_BackFeetMeasureSet.`

$$\texttt{internal\_FeetAverageForward} = \begin{cases} > 0, \text{ falls Pfoten relativ zur Hüfte eher vorne liegen} \\ < 0, \text{ falls Pfoten relativ zur Hüfte eher hinten liegen} \end{cases}$$

$$\texttt{internal\_FeetAverageUp} = \begin{cases} > 0, \text{ falls linke Pfote über rechter liegt} \\ < 0, \text{ falls rechte Pfote über linker liegt} \end{cases}$$

$$\texttt{internal\_FeetAverageSide} = \begin{cases} > 0, \text{ falls Pfoten relativ zur Hüfte eher rechts liegen} \\ < 0, \text{ falls Pfoten relativ zur Hüfte eher links liegen} \end{cases}$$

$$\texttt{internal\_FeetAverageUp2} = \begin{cases} > 0, \text{ falls Pfoten relativ zur Hüfte eher oben liegen} \\ < 0, \text{ falls Pfoten relativ zur Hüfte eher unten liegen} \end{cases}$$

Die Rotationen von Brust und Hüfte werden primär durch direkte Rotation der Steuerungsobjekte und einem geringen Anteil („HipsRotationAmount“, „ChestRotationAmount“) addierter Automatisierung gesteuert. Die oben angesprochenen Messergebnisse wurden so verwendet, dass der Automationsanteil ein „Lehnen“ der Hüfte, beziehungsweise der Brust, in Richtung der Pfoten bewirkt. Die Automation ist durch Setzen des Faktors auf Null natürlich auch wieder komplett abschaltbar.

Praktisch wurde dies derart gelöst, dass der Rotationsanteil, der aus der Automatisierung hervorgeht, einem Null-Objekt zugewiesen wird, das Child eines anderen Objektes ist, welches exakt die Transformationen des Steuerungsobjektes für die Hüfte, beziehungsweise der Brust, besitzt.

Mit dieser Technik war es nicht nötig, Transformationen innerhalb einer einzigen Expression zu addieren. Stattdessen wurden Parent-Child-Beziehungen aufgebaut, in denen das Child die Transformationen seines Parents erbt und zusätzliche Parameter aufaddiert bekommt. Das letzte Child der Kette besitzt dann die endgültigen Koordinaten. Diese Technik wurde bei allen derartigen Automationen verwendet.

Die Positionen von Brust und Hüfte werden ebenfalls primär durch Steuerungsobjekte gesteuert, ein Algorithmus für die Automatisierung der Position ist als Automatisierung implementiert. Es werden die, durch die oben angesprochene Messkonstruktion ermittelten, internen Parameter für die Hüft- beziehungsweise Brustrotation für diesen Zweck verwendet. Je nach Position der Beine zu Brust oder Hüfte wird eben dieses Element und die dazugehörigen Schultern verschoben. Der Offset wird im „internal_ChestPositionSet" berechnet, mit dem entsprechendem Parameter aus dem Parameterset „var_AutomationsAmountSet" multipliziert und durch den entsprechenden Parameter des Typs „Konstante“ aus dem Parameterset „const_ChestPositionLimitSet" begrenzt.

Zu einem Problem kann bei zu starker Verwendung dieser Automation werden, dass bei einer zusätzlichen oder später folgenden Bewegung des Hüft-Steuerungsobjektes dieser Offset zunächst wieder ausgeglichen wird, weshalb man dann nicht die volle Kontrolle über die Hüftanimation hat.

5.5.3 Schultern, Beine und Pfoten

Um eine Bewegung der Schultern zu ermöglichen, wurde für jede Schulter ein eigener Schulterknochen in das Rig eingebunden. Die Ausrichtung dieses Knochens ist durch ein Steuerungsobjekt manuell steuerbar, eine zusätzliche automatische Bewegung kann aufaddiert werden.

Dazu wurden die Schulterknochen auf ein Null-Objekt ausgerichtet, welches die Koordinaten des dem Steuerungsobjekt zugeordneten Nullobjektes erbt, addiert mit den Ergebnissen der Automation. Dieser Offset wird durch verlinkte Parameter bestimmt („const_ShoulderOffsetSet“), die mit Parametern des Typs „Intern“ verknüpft sind („internal_ShoulderAutomatationsSet“). Diese wiederum beziehen Ihre Werte aus der in Kapitel 5.5.2 angesprochenen Messkonstruktion. Durch den Einsatz von verlinkten Parametern war eine bessere Feinsteuerung der Ergebnisse der Automation möglich als es allein mit Expressions möglich gewesen wäre.

Für die Steuerung der Beine, die ihren Ursprung am Ende des jeweiligen Schulterknochens haben, wurde zusätzlich zu dem natürlich auf jeden Fall erforderlichen Steuerungsobjekt, das den Effektor der Kette steuert, ein Up-Vector-Steuerungsobjekt erstellt (Abb. 5.12). Nur dadurch ist eine volle Steuerung der Beine möglich, indem man nämlich durch den Up-Vector die Ausrichtung der Y-Achse des ersten Knochens festlegt. Dies ist deshalb nötig, weil 2D-Chains, wie der Name schon sagt, von alleine nur in einer Ebene rotieren können.

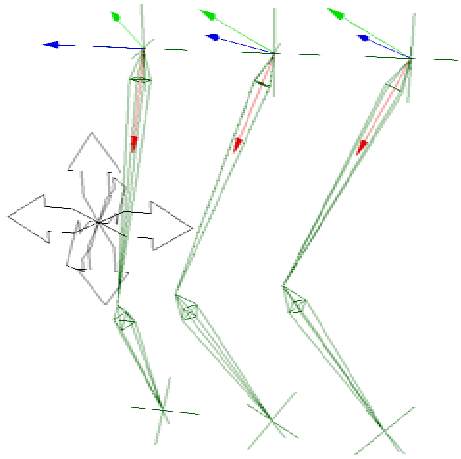

Abb. 5.12: Up-Vector-Constraints. Die Drehung aller drei kinematischen Ketten werden durch die Position eines Objektes gesteuert

Für eine noch genauere Steuerung der Beine wurde ein Slider erstellt, der die Funktion des IK/FK-Blending von XSI, die standardmäßig in den Optionen der Kinematikkette versteckt ist, in den Vordergrund rückt und dem Animator besser zugänglich macht. Der Slider kann von dem Animator später über die erstellte Synoptic View aufgerufen werden.

Die Technik des Überblenden zwischen Inverser- und Vorwärtskinematik erlaubt es dem Benutzer, Animationen sowohl durch Bewegen der Steuerungsobjekte, als auch durch Rotieren der einzelnen Knochen zu erstellen. Mithilfe des Sliders kann dann weich zwischen den beiden Animationen übergeblendet werden. Die Arbeit mit Vorwärtskinematik ist besonders sinnvoll, wenn absolute Kontrolle über jeden einzelnen Knochen gewünscht ist, etwa wenn „Willy“ sich hinlegen soll und kein Teil des Beines den Boden durchdringen soll.

Als weitere Automation wurde eine Ebene erstellt, deren einziger Zweck es ist, als Begrenzungsebene für die Pfoten-Steuerungsobjekte zu dienen. Dazu wurden die Pfoten-Steuerungsobjekte mit einem Bounding Plane-Constraint versehen, der verhindert, dass sie durch diese „FloorDummy“ genannte Ebene hindurchdringen können (Abb. 5.13). Dies kann die Animation etwa einer Laufbewegung von „Willy“ ernorm vereinfachen. Wenn diese Hilfe nicht gewünscht wird, etwa bei unebenem Boden, kann sie natürlich auch über die Synoptic View deaktiviert werden.

A. Rossano empfiehlt für diesem Zweck eine Expression, da diese seiner Beobachtung nach schneller berechnet werden kann als ein Constraint. Da eine darstellbare Ebene aber für den Benutzer besser zu visualisieren ist und etwa auch rotiert werden kann, wurde diese Methode gewählt (Rossano2002).

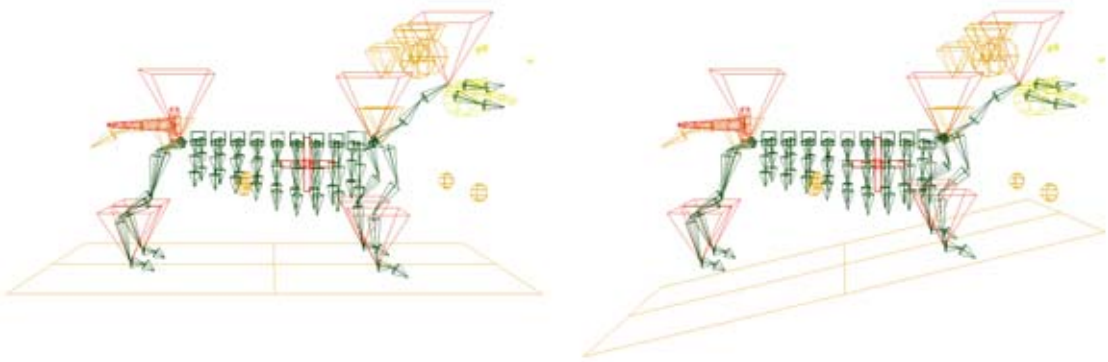

Abb. 5.13: Auch eine Rotation des „FloorDummy" ist möglich und beeinflusst die Steuerungsobjekte für die Pfoten

Die Pfoten selbst wurden aus relativ simplen kinematischen Ketten mit zwei Knochen erstellt. Diese Knochen können zueinander, zumindest über die konstruierte Steuerung, nicht bewegt werden. Wohl aber kann natürlich die gesamte Kette transformiert werden. Was die X- und Y-Achse der Kette angeht ist diese über eine Expression mit dem Null-Objekt des Steuerungsobjektes verbunden. Bei der Z-Achse, die das Rollen der Pfoten angibt, wurden bessere Ergebnisse erreicht, indem der Up-Vector der Kette durch ein

Path-Constraint an eine halbkreisförmige Kurve gebunden wurde und die Position auf der Kurve mithilfe einer Expression durch das Null-Objekt des Steuerungsobjektes gesteuert wurde.

5.5.4 Schwanz

Zur Animation des Schwanzes von „Willy“ wurde sicherlich eine der sinnvollsten und auch innovativsten Automationen des Rigs konstruiert. Ziel war es, eine Möglichkeit zu schaffen, dem Schwanz mit wenigen Parametern eine natürliche Bewegung zu ermöglichen und gleichzeitig ein exaktes Platzieren des Schwanzes zu ermöglichen, beispielsweise um Durchdringungen des Schwanzes mit Böden oder Wänden zu vermeiden.

Zu Beginn des Projektes war für den Schwanz eine andere Konstruktionsmethode als die später verwendete in der näheren Auswahl. Diese sah vor, Bones als Deformer-Objekte an eine simple Geometrie zu binden, die durch eine Cloth-Simulation deformiert wurde. Diese Geometrie würde dabei nur als Hilfsobjekt dienen, da die Cloth-Simulation von sich aus natürlich nicht auf Bones direkt angewendet werden kann.

Für Rigs, bei denen weniger Wert auf die Echtzeitdarstellung gelegt wird, ist diese Methode dennoch zu empfehlen. Sie hat insbesondere der direkten Physik-Simulation des Schwanzes, also dem Anwenden des Simulations-Operators auf die endgültige Geometrie des Charakters gegenüber einige Vorteile. So umgeht sie mögliche Fehler der Cloth-Simulation, beispielsweise Überschneidungen von Polygonen, durch den Umweg über Bones. Die Rotationen der Bones könnten außerdem einmal in Aktionen gespeichert und dann auf andere Modelle übertragen werden. Da die Hilfsgeometrie im Allgemeinen weniger Polygone als das endgültige Modell hat, ist die Simulation zusätzlich wesentlich schneller zu berechnen

Diese Möglichkeit hat sich aber leider aufgrund der langen Rechenzeiten der Simulation und der damit verbundenen Untauglichkeit zur Echtzeitdarstellung der Animation für das im Rahmen dieser Arbeit erstellte Rig als nicht praxistauglich erwiesen. Sie hätte weiterhin den Nachteil gehabt, dass die Länge des Schwanzes nach der Konstruktion und dem Zuweisen des Simulations-Operators nicht mehr verändert werden konnte, was die Skalierung des Rigs an andere Charaktere ernorm erschwert hätte.

Die letztendlich verwendete Methode ging deshalb einen völlig anderen Weg. Sie basiert darauf, grob vereinfacht dargestellt, zwei kinematische Ketten zu erstellen. Dabei ist die erste für eine automatische Bewegung des Schwanzes, gesteuert durch Expressions zuständig (im Folgenden „Expressions-gesteuerte Kette“ genannt) und die andere für die manuelle Festlegung der Schwanzposition („Manuell-gesteuerte Kette“). Die späteren Deformer-Objekte können nun zwischen diesen beiden kinematischen Ketten interpoliert werden. Damit hat diese Konstruktion prinzipielle Verwandtschaft mit dem von Softimage|XSI implementierten FK/IK-Blending, besprochen in Kapitel 5.5.3, „Schultern, Beine und Pfoten“.

Zunächst soll der Aufbau der Expressions-gesteuerten Kette erläutert werden: Das Prinzip eines durch Expressions gesteuerten Schwanzes oder eines ähnlichen organischen Fortsatzes ist nicht ganz neu. A. Rossano hat etwa in seinem Buch Beispiele dafür gegeben (Rossano2002, Rossano2003).

Die natürlich wirkende Bewegung beruht dabei darauf, dass Rotationen sich durch den Schwanz hindurch fortpflanzen und dabei abschwächen. Um zeitlich verzögerte Bewegungen zu erreichen, kann bei XSI die Funktion „at_frame“ in Expressions verwendet werden. Es liest eine vorzugebende Funktionskurve zu einem bestimmten Frame aus. Durch Angabe des Ausdrucks „Fc – X“ für das auszulesende Frame kann der Wert der Kurve X Frames vor dem aktuellen Frame (Fc) ausgelesen werden. Die Verwendung dieses Befehls setzt daher zwingend eine vorhandene Funktionskurve, also mindestens einen für den betreffenden Parameter gesetzten Key, voraus.

Die Parameter des Typs „Konstanten“, welche die zeitliche Verzögerung für jedes Element der Kette festlegen („TailTimeOffsetSet“), wurden derart gewählt, dass die Verzögerung vom Schwanzansatz aus kontinuierlich zunimmt. Die Faktoren mit denen die Rotationen multipliziert werden („const_TailFlexibilitySet“) nehmen gleichsam ab.

Aufgrund der Beobachtung, dass der Schwanz eines Teckels eine natürliche Biegung hat, die immer beibehalten wird, wurden diese Expressions für das Rig von „Willy“ noch etwas modifiziert. Parameter des Typs „Konstanten“, die im Parameterset „const_TailBendSet“ zu erreichen sind, legen diese natürliche Biegung für jedes einzelne Element des Knochens fest. Diese Biegung wird, multipliziert mit einem vom Benutzer einstellbaren Faktor, auf die Rotationen jedes einzelnen Elementes der Expressionsgesteuerten Kette aufaddiert.

Da das erste Element der Expressions-gesteuerten Kette noch nicht alle Elemente der Expression beinhaltet, hier beispielhaft die Expressions für das zweite Element:

```
[ Dog-Rig.Bone_ExprTail_2.kine.local.roty = ]
( Dog-Rig.TailHierarchy.const_TailBendSet.
const_TailBendBone2 * Dog-Rig.TailHierarchy.
var_TailSet.int_TailBend )
+ ( at_frame( Fc - Dog-Rig.TailHierarchy.
const_TailTimeOffsetSet.const_TailTimeOffsetBone2,
Dog-Rig.TailController.kine.local.rotx ) / 4 )

[Dog-Rig.Bone_ExprTail_2.kine.local.rotz = ]
( at_frame( Fc - Dog-
Rig.TailHierarchy.const_TailTimeOffsetSet.
const_TailTimeOffsetBone2, Dog-Rig.
TailController.kine.local.rotz ) / -4 )
* Dog-Rig.TailHierarchy.
const_TailFlexibilitySet.const_TailFlexibilityBone2
```

Die Manuell-gesteuerte Kette wurde auf eine völlig andere Art und Weise konstruiert. Zunächst soll die Notwendigkeit der Steuerung dieser Kette erläutert werden. Es wäre natürlich durchaus denkbar, auf eine aufwändige Konstruktion zu verzichten, da die Kette ohnehin manuell gesteuert wird.

Dazu ist zu bedenken, dass eine weiche Bewegung des Schwanzes nur mit relativ vielen Bones erreicht werden kann. In diesem Fall wurde eine Anzahl von acht Bones gewählt. Die Technik, die nun aber zum Überblenden zwischen der Expressions-gesteuerten Kette und der Manuell-gesteuerten Kette verwendet wird, funktioniert einwandfrei nur bei gleicher Anzahl von Elementen. Da dem Animator nicht zugemutet werden kann, acht einzelne Elemente der Manuell-gesteuerten Kette zu animieren, musste eine andere Möglichkeit gefunden werden.

Dazu wurde eine kinematische Kette mit drei Bones erstellt, welche tatsächlich vom Animator gesteuert werden. Dann wurde das bereits in Kapitel 5.5.1 erwähnte „Spine-Tool“ verwendet um diese Kette mit drei Elementen in eine mit acht Elementen zu konvertieren (XsiDoc2002, Animation).

Dabei macht man sich der Tatsache zunutze, dass das Spine-Tool eine Kurve erstellt, die von einem der zwei anzugebenden Enden des Spines zum anderen geht (Abb. 5.14). Als Enden sind dazu die Root und der Effektor der zu konvertierenden kinematischen Kette anzugeben. Die Platzierung der Wirbel kann dabei komplett vernachlässigt werden, es empfiehlt sich also, den Parameter für die interne Skalierung des Spines, der automatisch angelegt wird, auf den kleinstmöglichen Wert von 0,01 zu stellen. Ein zu kleiner Wert ist hier nicht von Bedeutung da er keinen Einfluss auf die resultierende Kurve hat, durch einen zu großen Wert kann aber eine andere Form der Kurve als beabsichtigt entstehen.

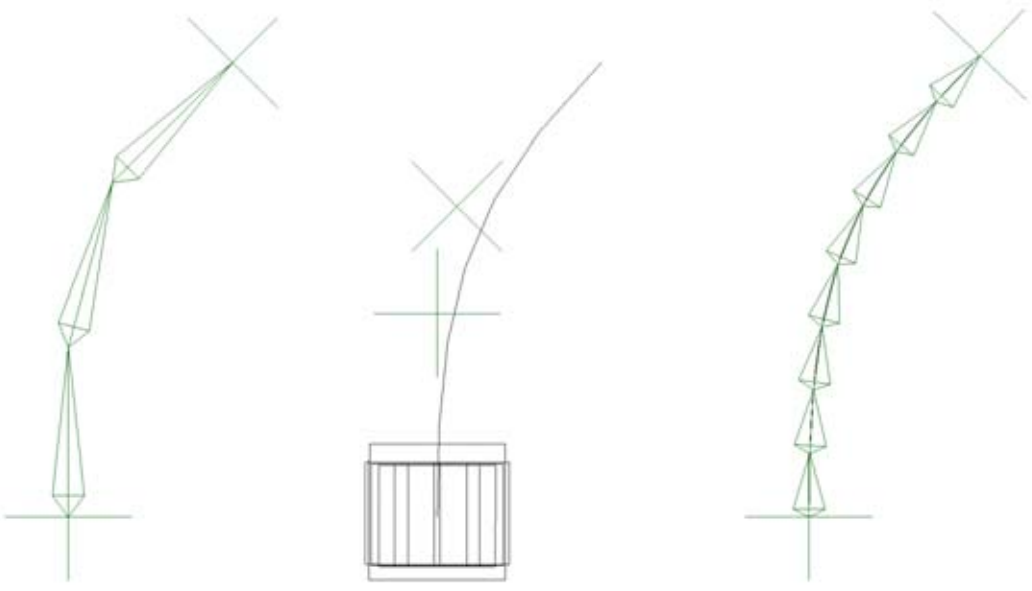

Abb. 5.14: Konvertieren einer kinematische Kette in eine andere mithilfe des Spine-Tools:
1.) Kinematische Kette mit drei Bones (links)
2.) Spine mit einer Skalierung auf 0,01 (mitte)
3.) Konvertierte Kurve, daran ausgerichtete Kette mit acht Bones (rechts)

Um die durch das Spine-Tool mithilfe eines Scripted Operators erstellte Kurve in den normalen Objekttyp „Curve“ von XSI zu konvertieren wird nun der „Fit on Curve“ Operator angewendet. Auf die resultierende Kurve wird nun wiederum der „Create Chain from Curve“-Operator angewendet und die endgültige Manuell-gesteuerte Kette mit acht an die Kurve gebundenen Bones erstellt (XsiDoc2002, Animation).

Um nun zwischen diesen beiden kinematischen Ketten mit je acht Bones zu überblenden, werden die schon in Kapitel 5.4 für die Gesichtsmuskulatur verwendeten Muscle-Deformer verwendet. Keine der beiden Ketten, weder die Expressions-gesteuerte Kette noch die Manuell-gesteuerte Kette, wird selbst als Deformer-Objekt verwendet. Stattdessen wurde eine dritte Konstruktion aus Muscle-Deformern aufgebaut.

Deren Positionen werden durch 2-Point-Constraints zwischen den Positionen der entsprechenden Bones der beiden Ketten gemittelt. Ein weiches Überblenden zwischen den beiden Ketten wird mithilfe der „Distance-Percentage“-Variablen der Constraints erreicht („var_TailSet.var_TailExprManuSwitch“), die zwischen beiden Positionen gewichten. Eine geringe Veränderung der Gesamtlänge des Schwanzes, die bei dieser Methode während des Überblendens entsteht, kann hier in Kauf genommen werden (Abb. 5.15).

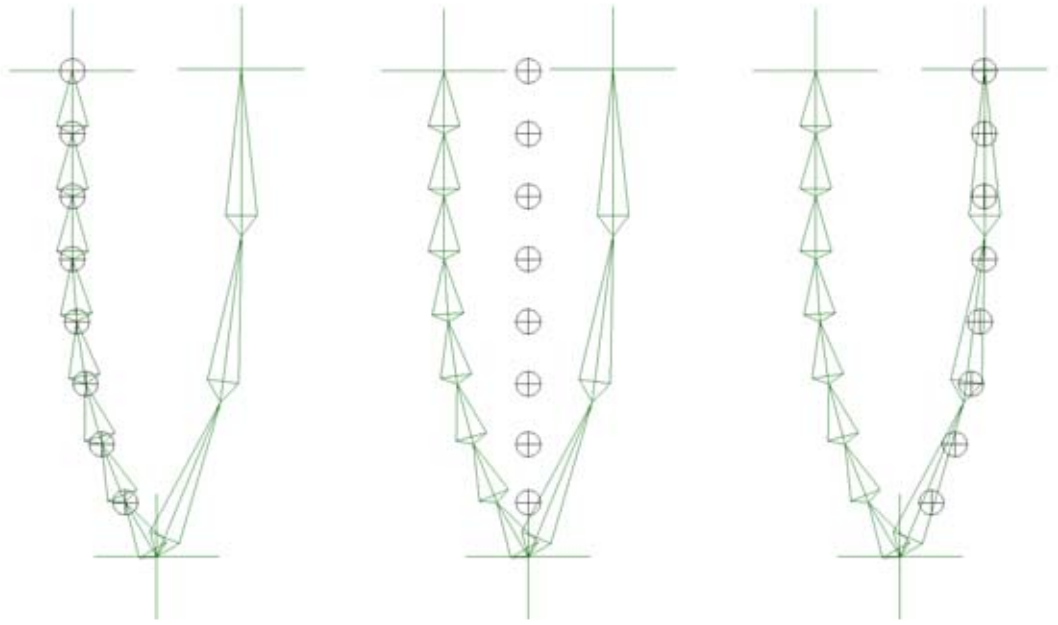

Abb. 5.15: Überblenden zwischen den beiden kinematischen Ketten:
1.) Wert von „var_TailExprManuSwitch“ = 0 (links)
2.) Wert von „var_TailExprManuSwitch“ = 50 (mitte)
3.) Wert von „var_TailExprManuSwitch“ = 100 (rechts)

Ein Rollen des Schwanzes um die eigene Achse wurde durch Ausrichten jedes der MuscleDeformer auf seinen Nachfolger mithilfe des Direction-Constraints und Parametrisieren der „Roll“-Variable des Constraints erreicht.

Da die Länge beider, als Basis der Überblendung herangezogenen, Ketten parametrisierbar ist, gilt dies natürlich auch automatisch für die Konstruktion aus Muscle-Deformern. Insgesamt ist die Länge des Schwanzes also voll parametrisierbar („const_TailLenghtSet“).

Im Gegensatz zu allen anderen Parametern des Rigs wurden einige Variablen des Schwanzes, wie etwa die Rotation des Schwanzansatzes, nicht von einem Custom Parameter abgenommen. Es wurde also teilweise keine untere Steuerungsebene implementiert, wie in Kapitel 5.3.1 beschrieben. Dies war nötig, weil die in den Expressions für die Expressions-gesteuerte Kette verwendete „at_frame“-Funktion eine Funktionskurve als Input benötigt und auch mit einer Expression, die nur den Wert des Steuerungsobjektes auf einen Parameter überträgt, nicht arbeiten kann.

Stattdessen wurden die benötigten Werte dann direkt von den betreffenden Steuerungsobjekten abgenommen. Für die Steuerung der Expressions-gesteuerten Kette fiel die Wahl auf ein kegelförmiges Objekt, für die Manuell-gesteuerte Kette auf die XSI-Standardsymbole für Bones. Mithilfe eines zylinderförmigen Steuerungsobjektes kann durch Verschieben die Biegung und durch Drehen die Rollbewegung des Schwanzes animiert werden (Abb. 5.16). Für die Überblendung zwischen den beiden Steuerungsmethoden wurde ein Slider implementiert, der über die Synoptic View aufgerufen werden kann.

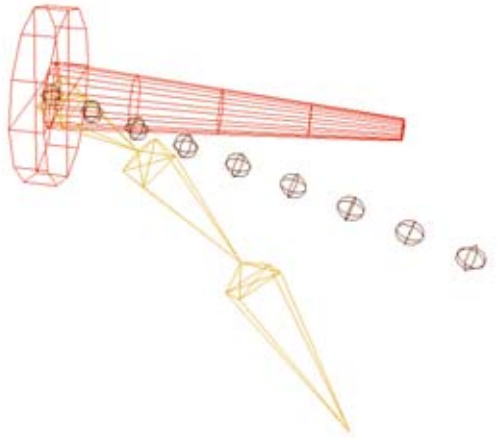

Abb. 5.16: Die für die Steuerung des Schwanzes konstruierten Steuerungsobjekte

Eine weitere Einschränkung der Konstruktion ist, dass die Expression „Frame_at“ für Nonlineare Animation mit Action-Clips nur funktioniert, wenn als zu speichernde Quellen „Store-Transformations-All Sources“ gewählt wird, was normalerweise aufgrund des wesentlich größeren Speicherbedarfs vermieden werden sollte.

5.5.5 Muskelbereiche

Die punktförmigen Deformer-Objekte, sprich Muscle-Deformer, kommen in dem entworfenen Rig neben der Verwendung als Gesichtsmuskeln und für den Schwanz noch bei anderen Anwendungen zur Verwendung. Sie bieten dem Envelope zusätzlichen Halt und garantieren eine etwa gleichbleibende Dichte von Deformer-Objekten etwa an den Beinen des Rigs, am Gesäß und am Hals.

Einige dieser Muscle-Deformer sind nicht nur statisch an IK-Elemente gebunden, sondern besitzen eine eigene, automatisierte Animation. So sind etwa die Muscle-Deformer des Gesäßes mithilfe eines 3-Point-Constraints an Schulter, Oberschenkel und Schwanzansatz gebunden. Damit wird die Bewegung einer oder mehrerer der genannten Komponenten auf das Gesäß übertragen und die Bewegung wirkt insgesamt natürlicher.

Die Verwendung von punktförmigen Deformer-Objekten ist in der Charakteranimation bereits üblich zur Simulation von Muskelsträngen wie etwa Bizeps oder Trizeps. Im Falle des Rigs von „Willy“ wurden sie in dieser Form als Bizeps für den Oberschenkel der Vorderbeine verwendet (Maraffi2001).

Dabei wurde eine Kurve erstellt und ein, per Path-Constraint an diese Kurve gebundener Muscle-Deformer, entlang der Kurve bewegt. Die Position auf der Kurve wurde dabei über einen verlinkten Parameter mit der Rotation des zweiten Knochens der kinematischen Kette verknüpft, also mit dem Beugungswinkel des Beines (Abb. 5.17). Die Verwendung von, über eine Funktionskurve, verlinkten Parametern vereinfacht hierbei wieder die Verwendung nichtlinearer Zuordnungen. Zum Anpassen der Muskelbewegung an andere Charaktere kann einfach diese Funktionskurve, sowie die Kurve an welche der Muscle-Deformer gebunden ist, modifiziert werden.

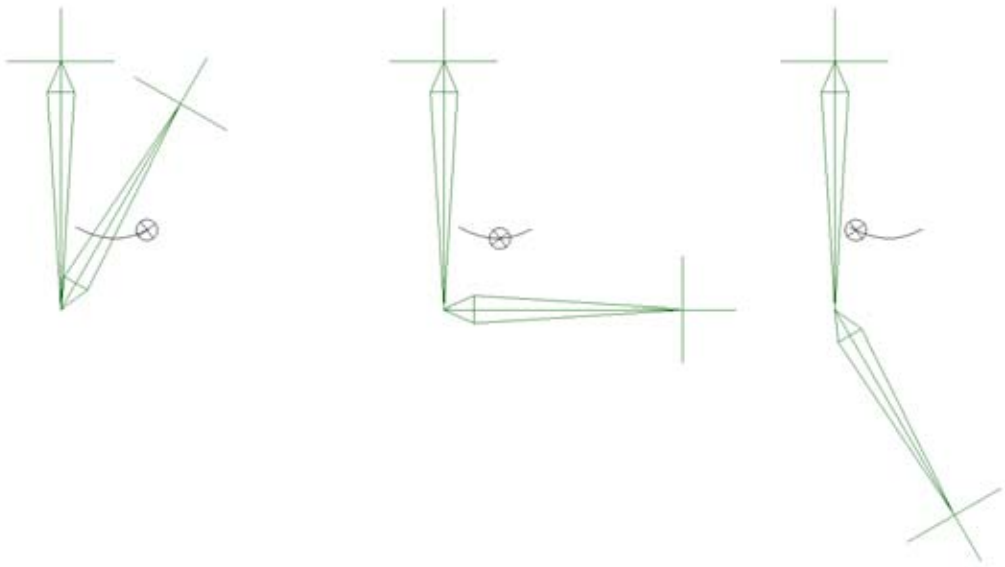

Abb. 5.17: Die Position des Muscle-Deformers auf der Kurve ist über einen verlinkten Parameter mit dem Winkel zwischen dem ersten und dem zweiten Bone verbunden

Eine Alternative zur Verwendung von Deformer-Objekten zur Simulation von Muskelsträngen wäre im Übrigen das Erstellen von Shape Keys und das Verknüpfen der Gewichtungen dieser Shape Keys mit der Rotation der Knochen. Der Nachteil dieser Methode wäre, dass die Geometrie des Objektes zum Anlegen der Shape Keys modifiziert werden müsste. Analog zum Verwenden von Shape Keys für Gesichtsausdrücke (Kapitel 5.4) müssten die Shape Keys damit für jeden neuen Charakter erneut erstellt werden. Eine Wiederverwendbarkeit des Rigs wäre nicht gewährleistet.

5.5.6 Gesichtsbereich

Ein großer Teil des Gesichtsbereiches von „Willy“ wurde mithilfe von Shape-Animationen animiert. In Kapitel 5.4 wurde die hierbei verwendete Technik erläutert, die darauf beruht, Kurven mit Deformer-Objekten zu verbinden und für die Kurven, nicht für die Geometrie selbst, Shape Keys zu erstellen.

Im Einzelnen wurden pro Auge zwei kreisförmige Kurven um das Auge herum gezogen, eine weitere Kurve wurde für die Stirn erstellt, außerdem je eine um die feinen Formveränderungen der Schnauze, des Ober- und des Unterkiefers zu ermöglichen.

Die Kurven für Ober- und Unterkiefer wurden dabei nur für geringe Formveränderungen verwendet, wie etwa zum Aufblasen der Backen oder Anziehen der Mundwinkel. Die entscheidende Bewegung des Unterkiefers, das Öffnen und Schließen des Mundes, wurde durch eine Kinematikkette aus zwei Knochen erreicht. Dabei wird der Effektor der Kette, die den Unterkiefer steuert, entlang eines Pfades mithilfe des entsprechenden Steuerungsobjektes gesteuert (Abb. 5.18).

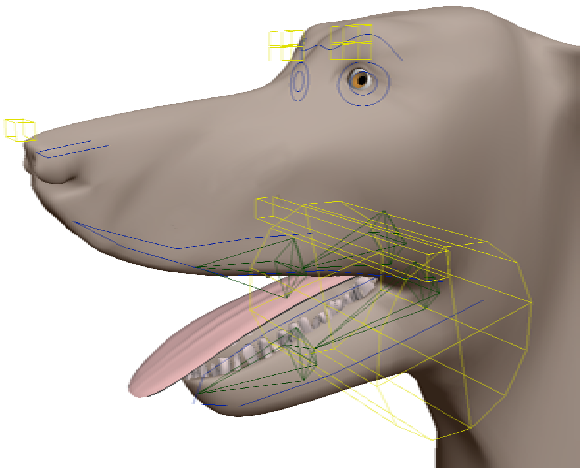

Abb. 5.18: Kurven zur Nachbildung von Muskeln, Bones für Ober- und Unterkiefer sowie die entsprechenden Steuerungsobjekte

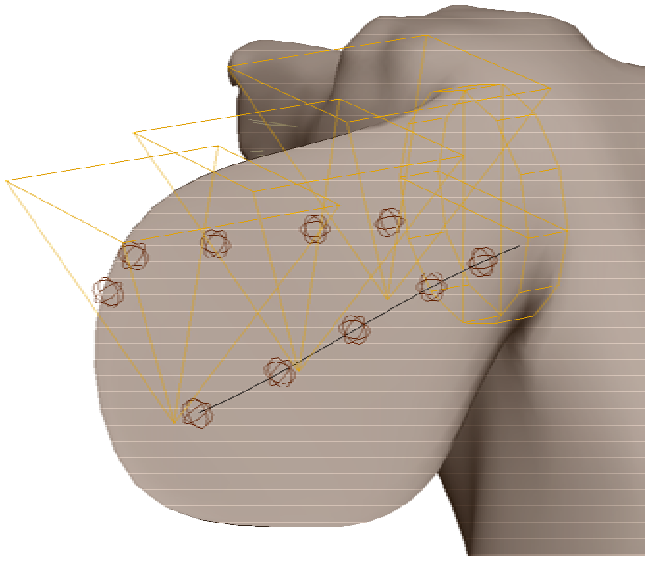

Abb. 5.19: Die Steuerungsobjekte zu Steuerung des Ohres, außerdem sichtbar die Kurve und die in zwei Reihen an ihr angeordneten Muscle-Deformer

Die Ohren von „Willy“ entstanden aus einer dem Schwanz ähnlichen Herangehensweise. Muscle-Deformer wurde an eine mithilfe des „Spine-Tools“ und des „Fit to Curve“-Operators erstellte Kurve gebunden. Als Anfangs- und Endpunkt des Spines wurde die Root und der Effektor einer kinematischen Kette gewählt, die vom Animator über drei Steuerungsobjekte gesteuert werden kann (Abb. 5.19).

Eine Rotation der Muscle-Deformer um die Kurve wird durch Animation des Up-Vector-Parameters des Path-Constraints, gesteuert durch zwei zylinderförmige Steuerungsobjekte, erreicht.

Die Muscle-Deformer rotieren dabei, abhängig von Ihrer Position auf der Kurve, unterschiedlich stark. Dieser Rotationsfaktor wird gesteuert durch die Parameter des Typs „Konstanten“ im Parameterset „const_EarsRollflexibilitySet“.

Die Zunge wurde separat vom restlichen Körper Willys behandelt. Das Geometrie-Objekt ist, genau wie etwa die Zähne oder die Krallen, als separates Objekt modelliert worden. Durch ein separates Enveloping wird ein manuelles Verändern der Gewichtung vermieden, was sonst nötig wäre, da im Gesichtsbereich eine relativ hohe Dichte von Deformer-Objekten existiert und die Zunge relativ nahe an den Ober- und Unterkieferknochen liegt.

Die Deformer-Objekte der Zunge wurden in einer den Ohren ähnlichen Technik an ein Spine gebunden, welches von einer kinematischen Kette gesteuert wird, deren Bones auch als Steuerungsobjekte dienen. Durch diese Konstruktion ist die Länge der Zunge zur Anpassung an andere Charaktere voll skalierbar („const_TongueDefaultLenghtSet“). Eine einfache Automation der Zungenbewegung wird erreicht, indem ein Faktor des Rotationswinkels des ersten Bones der Rotationswinkel des Unterkiefers ist.

Die Augen von „Willy“ sind das einzige Element des Charakters, bei dem aus praktischen Gründen nicht klar zwischen Hüllobjekten und Rig mit Deformer-Objekten getrennt wurde. Die Augen wurden separat erstellt und sind voll parametrisierbar, sowohl die Iris- als auch die Pupillengröße lässt sich einstellen (Kapitel 3.3.3). Da diese Steuerung allerdings nicht über das Rig erfolgt, wurde sie nicht in der Synoptic View implementiert, das entsprechende Parameterset („Eye_Left.EyeParameterSet“, „Eye-Right.EyeParameterSet“) muss manuell geöffnet werden. Die Richtung, in welche die Augen blicken, wird durch ein Direction-Constraint gesteuert. Die Null-Objekte, welche die Richtung angeben, sind über Constraints mit dem entsprechendem Steuerungsobjekt verbunden. Dabei wurden sie bewusst links beziehungsweise rechts von dessen Mittelpunkt platziert, um den Abstand zwischen den beiden Augäpfeln auszugleichen und um durch Drehen des Steuerungsobjektes ein „Verdrehen“ der Augen erzielen zu können (Maraffi2001).

5.6 Zusammenfassung

In diesem Kapitel, das sicher einen der Schwerpunkte der Arbeit beschreibt, wurden die verschiedensten Vorgehensweisen zur Konstruktion eines Character-Control-Rigs in Theorie und Praxis erläutert. Einige der angewandten Methoden sind komplette Neukonstruktionen, andere teilweise abgewandelte, bewährte Praktiken. Das Rig in seiner Gesamtheit stellt ein technisch weit fortgeschrittenes Beispiel eines Control-Rigs dar, das zur Animation in größeren Produktionen eingesetzt werden könnte. Es wurde weiterhin das Binden von Geometrie an das Rig erläutert und verschiedene Methoden zur Simulation von Knochen und Muskelbewegungen erläutert. Sicherlich konnte nicht jedes technische Detail des Rigs besprochen werden, das würde sicher auch den Rahmen dieser Arbeit sprengen, aber die prinzipielle Vorgehensweise und viele einzelne Feinheiten konnten erläutert werden.

Mit dem Charakter, wie er sich zu diesem Zeitpunkt präsentiert, ist das Animieren bereits möglich. Im folgenden Kapitel 6 wird jetzt eine Benutzeroberfläche entstehen, die Verbesserungen im Arbeitsablauf des Animators ermöglicht und so das Animieren weiter erleichtert.

6 Entwicklung einer grafischen Benutzeroberfläche zur Animationssteuerung

6.1 Einleitung

Nach dem vorigen Kapitel 5, „Aufbau des Control-Rigs“, steht ein voll funktionstüchtiges Rig zur Verfügung. Diverse Parameter, wie etwa die, welche die Automationen steuern, sind allerdings in Parametersets in der Hierarchie des Rigs verstreut. Es könnte einem Animator, der häufiger mit diesem Rig arbeitet, nicht zugemutet werden, diese Parametersets während der Arbeit ständig erneut zu lokalisieren.

Außerdem hat neben der Steuerung der Steuerungsobjekte über das dreidimensionale Benutzer-Interface von Softimage|XSI auch die Steuerung über zweidimensionale Steuerungselemente ihre Daseinsberechtigung. Nicht zuletzt können in der Synoptic View durch die Arbeit mit Skripts zusätzliche Hilfen für den Animator zur Verfügung gestellt werden.

Aus diesen Gründen wird in diesem Kapitel die Entwicklung einer grafischen Benutzeroberfläche zur Unterstützung der Steuerungsobjekte besprochen. Zunächst wird in Kapitel 6.2, „Die Bedeutung einer guten Charaktersteuerung“ die Notwendigkeit zusätzlicher Steuerungsmethoden erläutert.

Anschließend wird in Kapitel 6.3, „Realisierung der zwei möglichen Steuerungsmethoden“ die praktische Umsetzung der Steuerungsmethoden bei diesem Rig erläutert und das entwickelte Konzept zum parallelen Betrieb von drei- und zweidimensionalen Steuerungskonzepten wiederholt, bevor in Kapitel 6.4, „Die Synoptic View von Softimage|XSI“, das Konzept der Synoptic View in Softimage|XSI allgemein und in Kapitel 6.5, „Die erstellte Synoptic View“, die Implementierung der speziell im Rahmen dieser Arbeit entwickelten Synoptic View besprochen wird.

6.2 Die Bedeutung einer guten Charaktersteuerung

Der Sinn einer Charaktersteuerung ist es, dem Animator alle animierbaren Parameter eines Rigs zugänglich zu machen und damit die Low-Level-Animation des Charakters zu ermöglichen. Der Benutzer darf dabei nie die Übersicht über die Steuerung verlieren; Alle Parameter müssen mit wenigen Mausklicks und ohne unnötig langes Suchen erreichbar sein. Um dieses Ziel zu erreichen, ist sowohl technischer Sachverstand, als auch Grundverständnis für die Erfordernisse einer zweckmäßigen Mensch-Maschine-Interaktion erforderlich, auch unter dem Stichwort „Usability“ bekannt.

Wie entscheidend eine gute Usability für die tägliche Arbeit des Animierens sein kann, zeigen Frank Thomas und Ollie Johnston in ihrem Buch über die Disney-Trickfilme auf. Sie nennen als eines der vielen Erfolgskonzepte Disneys in der ersten Hälfte des 20. Jahhundert die Einführung neuer Zeichentische. Bis dahin war es allgemein üblich, die Blöcke von Blättern, auf denen die Animatoren ihre Zeichnungen anfertigten, am oberen Ende des Tisches zu befestigen und damit zu fixieren. Das zunächst von den Animatoren für unpraktischer gehaltene Konzept, diese Blätter am unteren Ende des Tisches zu befestigen, erlaubte ihnen jedoch, mehr als fünf Blätter schnell durchzublättern, um den Bewegungsablauf zu überprüfen (Abb. 6.1). Diese unscheinbare Entwicklung wird heute als eine von vielen Entdeckungen genannt, die den Erfolg dieses Konzerns ausmachten (Thomas1981).

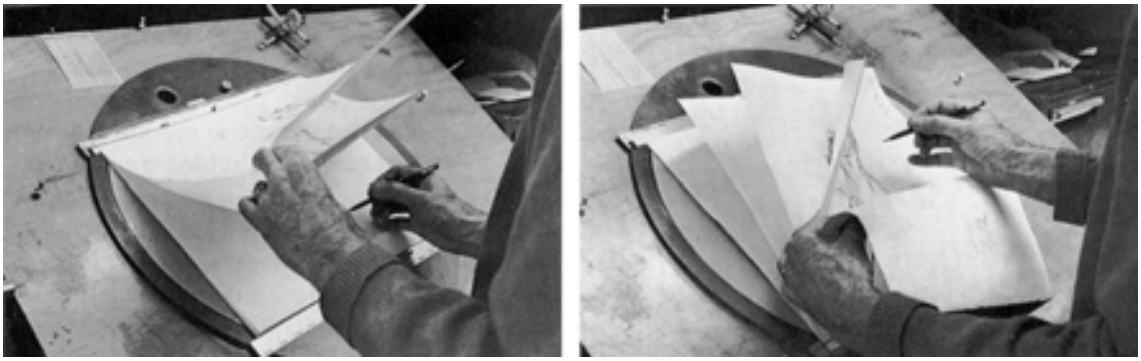

Abb. 6.1: Zeichenblätter am oberen Rand befestigt (links) und Zeichenblätter am unteren Rand befestigt (rechts). Dies erlaubte das schnelle Blättern von mehr als fünf Zeichnungen (Quelle: Thomas1981, S. 31; Fotografien: Dave Spencer)

Vor allem drei Hauptgründe für die besondere Bedeutung einer guten Charaktersteuerung seien im Folgenden herausgehoben:

1. Der Faktor „Effizienz“: Den am leichtesten einzusehenden Faktor stellt sicher der Effizienzvorteil einer guten Charaktersteuerung dar. Animatoren, die im wahrsten Sinne des Wortes wertvolle Arbeitszeit damit verbringen müssen, animierbare Parameter in einer unübersichtlichen Oberfläche erst lokalisieren zu müssen oder für ständig wiederkehrende Aufgaben unnötig viele Kommandos ausführen müssen, können kaum effizient arbeiten.

2. Der Faktor „Qualität“: Eine gute Charaktersteuerung kann einen großen Einfluss auf die Qualität der resultierenden Animationen haben. Selbst die sinnvollste Automation oder der am trickreichsten angelegte Parameter erfüllt ihren Zweck nicht, wenn sie vom Benutzer nicht gefunden werden können. Nicht zuletzt ist zu bemerken, dass durch eine schlechte Charaktersteuerung geplagte Animatoren auch Motivationsprobleme bekommen könnten.

3. Der Faktor „Unabhängigkeit“: Für die Produktion von Charakteranimationen in einem größeren professionellen Umfeld ist auch zu bedenken, dass eine gute Charaktersteuerung einem Unternehmen auch den Vorteil der Unabhängigkeit von einem speziellen Animator bringen kann. (Quelle: Vortrag „Rudi und Ralf –

Animierte Charaktere als Sympathieträger am Beispiel des Jägermeister Spots“ von Peter Span, Spans und Partner, im Rahmen der fmx/03, 02.05.2003). Zwar wird häufig versucht, einen Charakter von ein und demselben Animator animieren zu lassen, um den persönlichen Stil des Charakters in seinen Bewegungen leichter reproduzieren zu können. Da dies jedoch nicht in jedem Fall möglich ist, sollte ein, gegebenenfalls für diesen Charakter neuer, Animator nicht schon mit der Bedienung der Charaktersteuerung zu kämpfen haben.

6.3 Realisierung der zwei möglichen Steuerungsmethoden

Bereits in Kapitel 5.3.1 wurde der Aufbau einer unteren Parameter-Ebene als Basis aller Steuerungsmöglichkeiten erläutert. Auf diese untere Ebene soll nicht direkt durch den Benutzer zugegriffen werden. Sie bildet stattdessen die gemeinsame Schnittmenge der beiden im Folgenden dargestellten Steuerungsmöglichkeiten.

Die Steuerungsmethode, die zu diesem Zeitpunkt bereits funktionsfähig ist, besteht aus der Steuerung des Charakters über Steuerungsobjekte. Mit Ausnahme von wenigen Parametern, etwa derjenigen die sich auf Automationen beziehen, lassen sich alle Parameter über Steuerungsobjekte steuern.

Dabei muss zwischen Steuerungsobjekten unterschieden werden, die über ihre Transformation, beziehungsweise die der ihnen zugeordneten Null-Objekte, direkt die Transformationen von anderen Elementen des Rigs wie etwa Effektoren steuern, und solchen Steuerungsobjekten, die, beispielsweise über Expressions, andere Parameter, meistens Custom Parameter, steuern. Zu der letztgenannten Gruppe können als Beispiel diejenigen Steuerungsobjekte genannt werden, die durch ihre Transformationen die Gewichtung von Shape Keys zur Simulation der Gesichtsmuskulatur steuern.

Diese letztgenannte Gruppe von Steuerungsobjekten könnten, wie gelegentlich auch zu sehen, auch als separate Konsole, also räumlich getrennt vom Rest des Rigs, arrangiert werden (Abb. 6.2). Dies ist jedoch nur eine Frage der Platzierung dieser Steuerungsobjekte. In diesem Fall wurde entschieden, ihre Position an den Rest des Rigs zu binden, die Steuerungsobjekte für die Gesichtsmuskulatur von „Willy“ also an dem Steuerungsobjekt für seinen Kopf auszurichten.

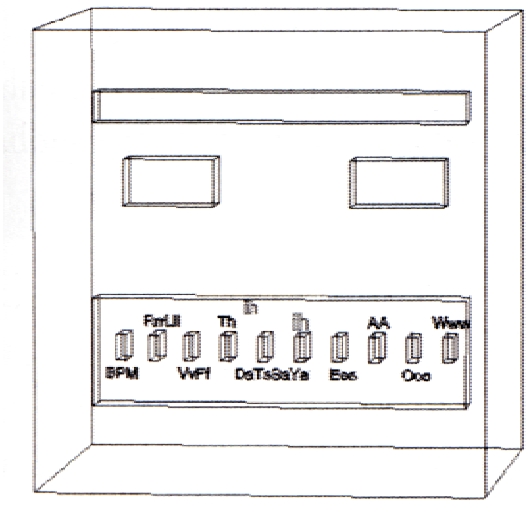

Abb. 6.2: Steuerungsobjekte zur Steuerung von Gesichtsanimationen, als Konsole arrangiert (Quelle: Rossano2002, S. 317)

In Kapitel 5.3.2 wurde bereits auf Maßnahmen eingegangen, welche die leichtere Bedienung des Rigs über die Steuerungsobjekte zum Ziel haben: Die Steuerungsobjekte wurden beispielsweise mit Transform Setups

versehen um nach der Selektion eine sofortige Transformation zu vereinfachen. Ebenfalls dem Ziel die Transformation zu vereinfachen, diente die Einrichtung von Position- und Rotation-Limits. Die farbige Unterscheidung zwischen drei Gruppen von Steuerungsobjekten wurde durch Modifizieren der Objektfarbe in der Drahtgitter-Darstellung erreicht, was die Auffindbarkeit eines speziellen Steuerungsobjektes verbessern soll.

Die oben getroffene Unterscheidung zwischen zwei Gruppen von Steuerungsobjekten kann in ähnlicher Weise auch bei zweidimensionalen Steuerungselementen getroffen werden. Diese Unterscheidung ist hier für die technische Umsetzung dieser Steuerungsmethode hilfreich.

So wären einmal diejenigen zweidimensionalen Steuerungselemente, die Parameter von Steuerungsobjekten repräsentieren, also auch über Steuerungsobjekte steuerbar sind. Da Steuerungsobjekte mit genau den Parametern, die auch zweidimensional über die Synoptic View zugänglich gemacht werden sollen, hier bereits existieren, können diese Parameter auch zu diesem Zweck verwendet werden. Dazu genügt es, die Steuerungsobjekte mit sogenannten „Marking Sets“ zu versehen.

Dadurch legt XSI für das entsprechende Objekt ein Parameterset mit dem Namen „MarkingSet“ an, in welchem vorher anzugebende Parameter des Objektes als Proxy Parameter angelegt und verknüpft werden. Als anzulegende Parameter werden dabei natürlich die für das jeweilige Steuerungsobjekt sinnvollen Transformationen gewählt. Dies sind natürlich auch diejenigen, welchen trotz Rotation- und Position-Limits noch Bewegungsspielraum gelassen wurde.

Der eigentliche Sinn dieser Funktion ist es, das Setzen von Keys zu erleichtern, da XSI, falls einem Objekt ein Marking Set zugewiesen wurde, beim Ausführen des Kommandos „SaveKey“ für dieses Objekt nur diejenigen Parameter, die markiert wurden, mit einem Key versieht.

Dies wurde beim Programmieren des Skripts zum Setzen von Keys für die Synoptic View genutzt. Der Hauptgrund für die Verwendung vom Marking Sets war aber, dass im Rahmen der erstellten Steuerung die erstellten Marking Sets, beziehungsweise deren Proxy Parameter, als Slider für die Transformationen der Steuerungsobjekte verwendet werden können.

Bei der Erstellung der Synoptic View konnte sich weiterhin die Tatsache zunutze gemacht werden, dass Marking Sets auch beim Selektieren eines Steuerungsobjektes in einem offenen Property-Editor angezeigt werden, sobald unmittelbar zuvor der Property-Editor einmal zur Anzeige eines Custom Parametersets genutzt wurde (Kapitel 6.5.1, „Allgemeine Hinweise zur erstellten Synoptic View“).

Da die Marking Sets quasi Proxy Parameter der lokalen Translationen der Steuerungsobjekte enthalten, ist eine parallele Steuerung über Slider und über Steuerungsobjekte möglich. Änderungen an den Slider wirken sich unmittelbar auch auf die Steuerungsobjekte aus und umgekehrt.

Zu der zweiten Gruppe unter den zweidimensionalen Steuerungselementen gehören diejenigen Parameter, die es nur als Parametersets gibt, die also kein Gegenstück unter den Steuerungsobjekten haben. Zu ihnen gehören beispielsweise alle Parameter aus dem Bereich der Automationen.

Für diese Parameter mussten manuell Parametersets erstellt und Proxy Parameter angelegt werden. Bei dem Anlegen der Proxy Parameter konnte nun auf den Suffix, der den Typ des Parameters angibt, verzichtet werden, ebenso konnte der Name des Parameters derart gewählt werden, dass er eine für den Anwender leichter zu verstehende Erklärung der Funktion des Parameters bietet.

Zusammenfassend lässt sich sagen, dass sowohl die Steuerung durch Steuerungsobjekte als auch durch zweidimensionale Steuerungselemente ihre Daseinsberechtigung hat. Nicht alle Parameter lassen sich über Steuerungsobjekte steuern. Betrachtet man allerdings den Bedienkomfort, dürfte die Steuerung über Steuerungsobjekte den meisten Animatoren eher entgegenkommen.

Steuerungsobjekte können dem Benutzer ein besseres visuelles Feedback geben, ihre Funktion ist meistens nahezu selbsterklärend. Durch die fehlende Beschränkung auf zwei Dimensionen können mehrere Parameter mir einem einzigen Steuerungsobjekt gesteuert werden und dadurch auch die Anzahl der zu steuernden Elemente reduziert werden (Rossano2002).

Es kann allerdings auch Anwendungen geben, in denen ein präzises Steuern einer Bewegung über Slider oder gar die direkte Eingabe von Zahlenwerten sinnvoller ist. Die Anzeige von Marking Sets während der Arbeit erlaubt es außerdem auf einfache Weise, Keys nur auf ganz bestimmte Parameter eines Steuerungsobjektes zu setzen, nicht auf allen im Marking Set enthaltenen Parameter.

6.4 Die Synoptic View von Softimage|XSI

Die Synoptic View ist eine sehr mächtige Funktion in Softimage|XSI , die es ermöglicht, einen schnellen Zugang zu spezifischen Daten oder Kommandos eines Charakters oder einer Szene zur Verfügung zu stellen. Eine Synoptic View besteht aus einer einfachen Textdatei in der durch eine Untermenge von HTML-Befehlen eine Image-Map geladen wird. Sowohl die Textdatei, im Folgenden trotz einiger Einschränkungen „HTML-Datei“ genannt, als auch die eingebundene Bilddatei werden dabei außerhalb jeder XSI-Szenendatei gespeichert (XsiDoc2002, Animation).

Die HTML-Seite ist über ein „Synoptic View-Property“ mit einem oder mehreren Objekten in einer XSI-Szene verbunden und kann bei Selektion dieses Objektes innerhalb von XSI geöffnet werden. Durch Klick auf einen von der Image-Map definierten Hotspot kann entweder über einen HTML-Link eine weitere Synoptic View geladen oder ein Skript ausgeführt werden.

Dieses Skript wird dabei entweder direkt in den Head-Bereich der HTML-Seite geschrieben oder als externe Datei durch die HTML-Seite aufgerufen. Es kann nach den in XSI üblichen Vorgaben für Skripts erstellt werden (Kapitel 4.4.3). Einzige Ausnahme dabei ist, dass für die Synoptic View erstellte Skripts nicht in allen sonst möglichen Skriptsprachen, sondern nur entweder in VBScript oder in JScript verfasst sein dürfen.

Auch in der „Net View“ von Softimage|XSI lassen sich Skripts über HTML-Seiten aufrufen. Der große Vorteil der Synoptic View ist demgegenüber aber die Multi-Plattformfähigkeit. Während die Net View auf dem Internet Explorer von Microsoft basiert und deshalb auch nur in Versionen von Softimage|XSI für Windows enthalten ist, basiert die Synoptic View auf internem Code von Softimage (XsiDoc2002, Fundamentals).

6.5 Die erstellte Synoptic View

6.5.1 Allgemeine Hinweise zur erstellten Synoptic View

Die im Rahmen dieser Arbeit entstandene Synoptic View ist in sechs Untermenüs aufgeteilt, die in den folgenden Unterkapiteln im Detail besprochen werden. Das Layout orientiert sich dabei sowohl an der Gestaltung der grafischen Benutzeroberfläche von XSI, als auch an den Synoptic Views der beiden, im Funktionsumfang von XSI enthaltenen, Beispiel-Rigs.

Die Struktur der Synoptic View wurde derart entworfen, dass die beiden Untermenüs „Selection“ und „Reset“, die beide in den angesprochenen Beispiel-Synoptic Views vorkommen, in ähnlicher Weise auch in der entworfenen Synoptic View auftreten (Abb. 6.3).

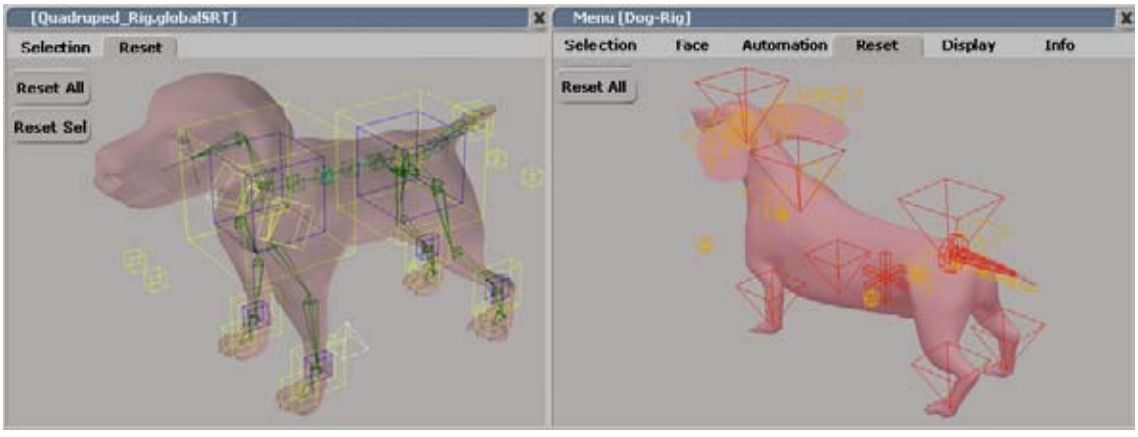

Abb. 6.3: Das Design der erstellten Synoptic View (rechts) orientiert sich an dem der im Lieferumfang von XSI enthaltenen Synoptic Views (links)

Das „Synoptic View-Property“, das demjenigen Objekt zugewiesen wird, welches aktiviert sein muss um bei Aufruf der Synoptic Views (Standard-Tastaturbelegung: F3) die Synoptic View zu öffnen, kann theoretisch jedem beliebigen Objekt zugewiesen werden. Damit jedes der Steuerungsobjekte die Synoptic View von „Willy“ öffnen kann, wurde die Synoptic View-Property dem kompletten Modell zugewiesen.

Das Objekt, dem die Synoptic View-Property zugewiesen wird, wird automatisch von XSI in der Variable „in_obj“ gespeichert. Diese Variable wurde in der vorliegenden Synoptic View, genau wie in den bei XSI mitgelieferten Synoptic Views, dazu genutzt, das Modell zu finden, auf welche die Funktionen der Synoptic View angewendet werden. Dadurch ist es möglich, mehrere der Rigs von „Willy“ in eine Szene zu laden und jedes durch seine eigene Synoptic View anzusprechen.

Als Skriptsprache wurde VBScript verwendet. Auf die Verwendung von externen Skripts wurde aus Gründen der Übersichtlichkeit verzichtet. Alle Skripts wurden direkt in die HTML-Datei eingebunden. Aufgrund der Mängel des Synoptic Editors bezüglich des Editierens der Skripts (beispielsweise fehlende Scrollbalken) wurde ein einfacher Texteditor für diesen Zweck verwendet. Auch die Bedienung des Synoptic Editors, was das Definieren der Hotspots angeht, ist nicht ideal. Aus diesem Grund wurde zur Erstellung der Image Maps eine externe Software, Adobe ImageReady, verwendet. Zur Erstellung von MouseOver-Effekten genügt es in XSI, eine zweite Version des Bildes, das die Grundlage für die Image-Map bildet, mit allen Buttons im aktiven Zustand zur Verfügung zu stellen.

6.5.2 Untermenü ‚Selection’

Jedes Steuerungsobjekt, mit Ausnahme derjenigen, welche die Parameter des Gesichts steuern, lässt sich über das erste Untermenü „Selection“ selektieren beziehungsweise deselektieren. Die in XSI üblichen Steuertasten werden unterstützt, das heißt bei gehaltener „Shift“-Taste wird die Auswahl erweitert, bei gehaltener „Strg“-Taste wird der Zustand der Auswahl umgeschaltet und bei gehaltener „Shift“- und „Strg“-Taste wird von der Auswahl abgezogen. Die den Steuerungsobjekten zugewiesenen „Transform Setups“ erleichtern die Bedienung zusätzlich, seit der Version 3.5 von Softimage|XSI werden sie nämlich automatisch bei Benutzen des Kommandos „SelectObj" aktiviert.

Der Button „Sel All“ dient zur Selektion aller Steuerungsobjekte, der Button „Key Sel“ zum Setzen eines Keys für das aktive Steuerungsobjekt und aller durch das Marking Set angegebenen Parameter. Über das Parameterset, welches bei einem Klick auf den Button „Options“ aktiviert wird, lässt sich einstellen, ob die Steuerungsobjekte für die Pfoten beim Selektieren so behandelt werden, als ob sie Child-Objekte der Hüft- und Brust-Steuerungsobjekte wären oder nicht (Abb. 6.4).

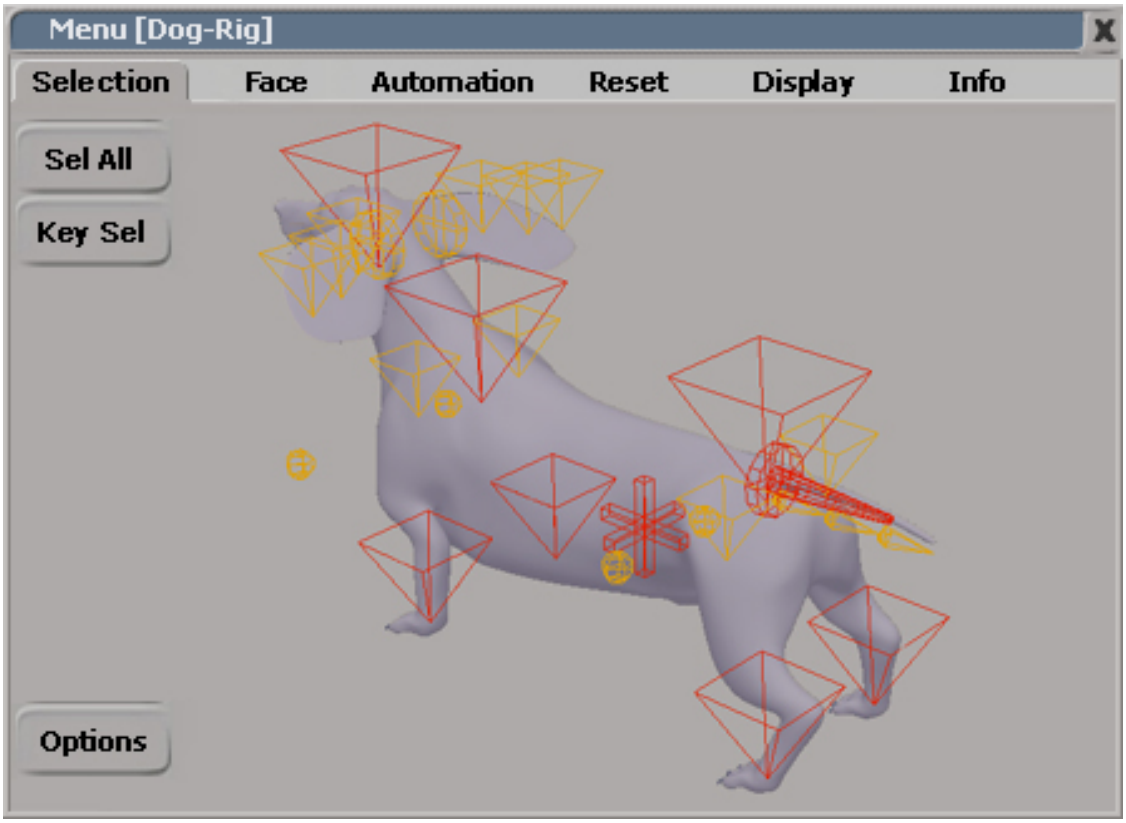

Abb. 6.4: Die Oberfläche des Untermenüs „Selection“

Im Folgenden werden alle Unterprogramme und Funktionen dieses Untermenüs als Struktogramme besprochen (Abb. 6.5, Abb. 6.6, Abb. 6.7 und Abb. 6.8). Auszüge des Quellcode stammen aus den im Lieferumfang von XSI enthaltenen Synoptic Views.

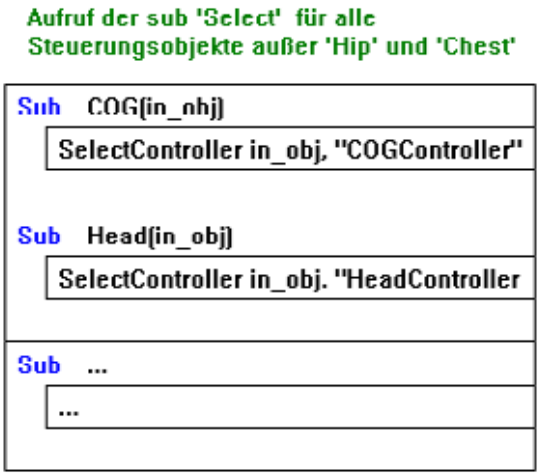

Abb. 6.5: Die Unterprogramme, die beim Klick auf den entsprechenden Button aufgerufen werden, führen ihrerseits die Funktion „SelectController“ aus

Tool-Funktionen des Untermenüs 'Selection'

```
Function   SelectController(in_obj. controllerName)
    set oRoot = Application.ActiveProject.ActiveScene.Root
    set myModel = oRoot.FindChild(in_obj)
    controllerPfad = myModel & "." & controllerName
    GetKeyboardState , key
    Select Case  CByte(key)
        Case  0 ' "No key pressed"
            SelectObj myModel & "." & controllerName
        Case  1 ' "Shift pressed"
            If  checkCollision(myModel.FindChild(controllerName)) = False
            Then                                                  Else
            AddToSelection myModel & "." & controllerName
        Case  2 ' "Ctrl pressed"
            ToggleSelection myModel & "." & controllerName
        Case  3 ' "Shift/Ctrl pressed"
            RemoveFromSelection myModel & "." & controllerName
        Case Else

Function   checkCollision(inObj)        'returns True if inObj is already selected
    dim sel, elem
    set sel = GetValue["SelectionList"]
    For  Each elem in Selection
                                  If  elem = inObj
        Then                                        Else
        checkCollision = True
        SelectObj sel
        Exit Function
    checkCollision = False
    SelectObj sel
```

Abb. 6.6: Die Funktion „SelectController“ wird aufgerufen, um die Selektion eines Steuerungsobjektes, je nach dazu gehaltener Taste, zu verändern. Die Funktion „checkCollision“ kann dabei aufgerufen werden, um zu testen ob ein Objekt schon Teil der Auswahl ist

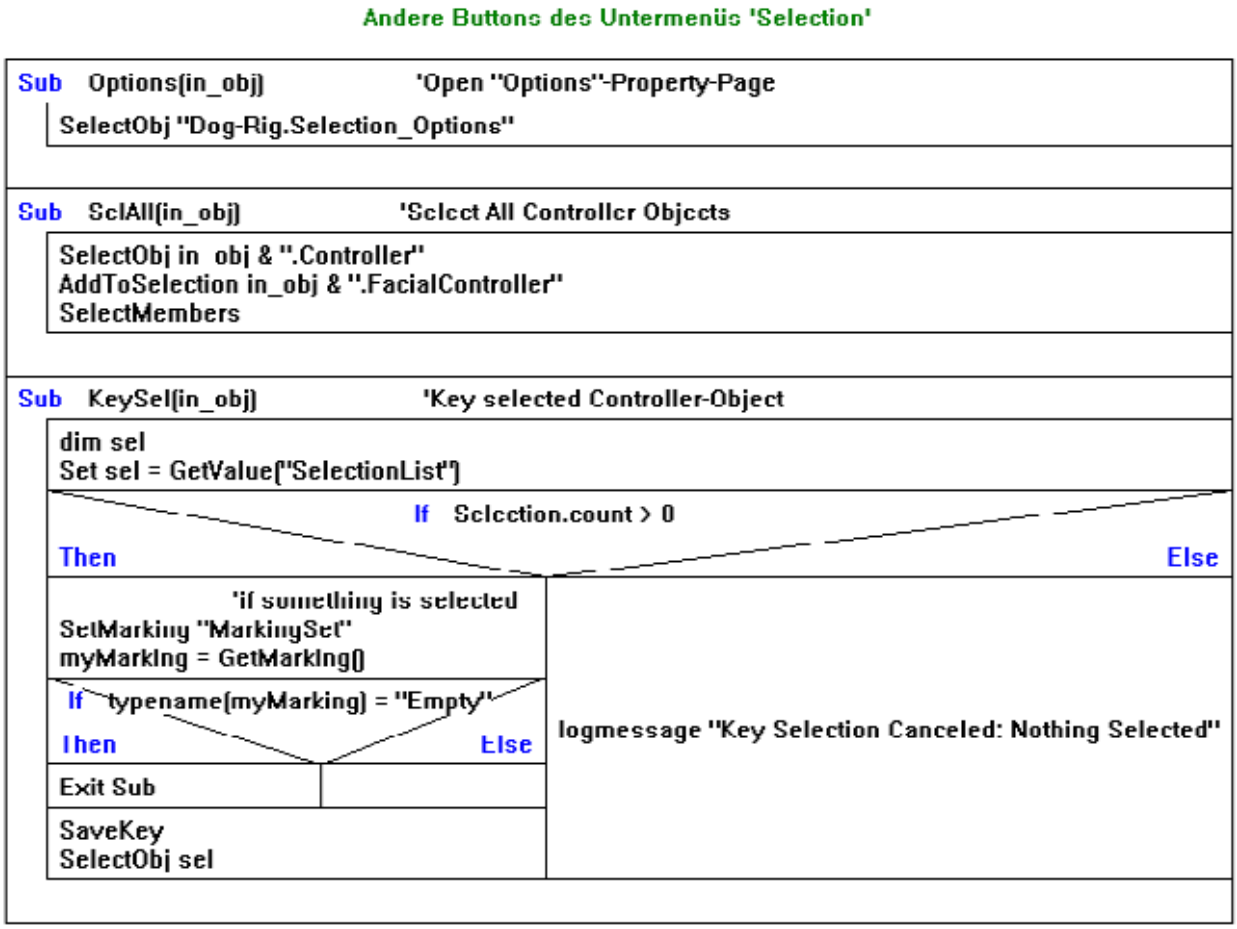

Abb. 6.7: Unterprogramme die von den Buttons „Options“, „Sel All“ und „Key Sel“ aufgerufen werden

Abb. 6.8: Struktogramm des Unterprogramms, das für die Selektion des Brust-Steuerungsobjektes und gegebenenfalls der Vorderpfoten zuständig ist. Das Unterprogramm für die Hüfte wurde analog dazu entwickelt.

6.5.3 Untermenü ‚Face'

Die Buttons für die Steuerungsobjekte des Gesichts und der Button „Key Sel" dieses Untermenüs funktionieren in der gleichen Weise wie die entsprechenden Buttons im Untermenü „Selection" (Abb. 6.9).

Aufgrund von in Kapitel 5.4 erläuterten Einschränkungen von referenzierten Modellen kann die Länge der Shape-Clips, welche die Basis für die Gesichtsanimationen bilden, nicht „per Hand", also über den Animationsmixer, angepasst werden. Dies ist jedoch einem Animator, der unter Umständen über den internen Aufbau des Rigs nicht informiert ist, auch nicht zuzumuten. Daher wurde in der Synoptic View mit dem Button „Adjust Shape-Clip-Length to Scene" eine Funktion implementiert, welche bei Ihrem Aufruf die Shape-Clips auf die volle Länge der aktuellen Szene skaliert.

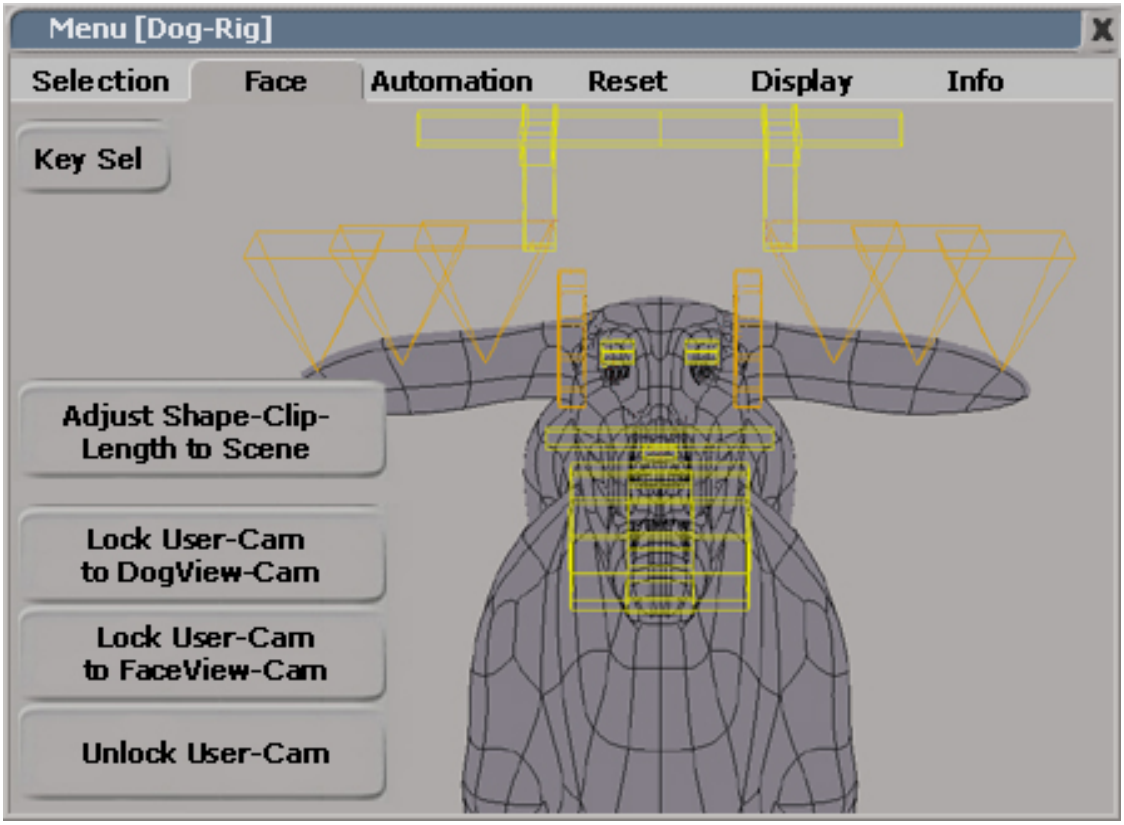

Abb. 6.9: Die Oberfläche des Untermenüs „Face"

Die zusätzlichen Funktionen dieses Untermenüs beziehen sich auf die Verbesserung der Übersichtlichkeit für die Animation des Gesichts und des Kopfes des Rigs. Dazu wurden zwei Kamera-Objekte in das Rig eingebaut, eine auf das Gesicht des Charakters ausgerichtet um seine Gesichtsmimik unkompliziert überprüfen zu können und eine in einer Position zwischen den Augen platziert, so dass ihr Sichtkegel etwa dem Sichtfeld des Charakters entspricht. Mithilfe dieser Kamera kann leicht die Blickrichtung des Rigs kontrolliert werden.

Da es per Skripts nicht möglich ist, die Anzeige dieser Kameras in einem Viewport zu aktivieren, wurden Funktionen geschrieben, welche alle relevanten Parameter der User-Camera, die standardmäßig in einem der Viewports angezeigt wird, an die entsprechenden Parameter der entsprechenden Kamera des Rigs bindet. Mithilfe des Buttons „Unlock User-Cam“ kann die User-Camera wieder frei vom Benutzer gesteuert werden. Die Tatsache, dass die beiden Kameras nicht direkt im Viewport angezeigt werden, schützt aber auch vor versehentlichem Verschieben oder Drehen der Kameras.

Die Selektion der Steuerungsobjekte und die Funktion zum Setzen von Keys wurden in der gleichen Weise implementiert wie die entsprechenden Funktionen im Untermenü „Selection“. An dieser Stelle sei nur das Struktogramm für die übrigen Funktionen gezeigt (Abb. 6.10).

Zusätzliche Funktionen des Untermenüs 'Face'

```
Sub   AdjustClips(in_obj)        'Adjust Clip-Length to Scene
      SceneMax = GetValue ("PlayControl.Out")
      SceneMin = GetValue ("PlayControl.in")

            'Setzen der Anfangs- und Endwerte für die ShapeClips
            'Muster:
            'SetValue "CLIP_PFAD.actionclip.timectrl.clipout", SceneMax
            'SetValue "CLIP_PFAD.actionclip.timectrl.clipin", SceneMin

Sub   FaceCam(in_obj)     'Lock User-Cam in all Views to FaceCam
      Dim ViewString(3)
      ViewString(0) = "Views.ViewA.UserCamera"
      ViewString(1) = "Views.ViewB.UserCamera"
      ViewString(2) = "Views.ViewC.UserCamera"
      ViewString(3) = "Views.ViewD.UserCamera"
      For   i = 0 To 3

            'Setzen einer Expression auf die User-Kamera-Parameter
            'Muster:
            'CopyPaste in_obj & ".Face-Cam.PAR_NAME",, ViewString(i) & ".PAR_NAME", Fc

Sub   DogViewCam(in_obj)  'Lock User-Cam in all Views to DogViewCam
      Dim ViewString(3)
      ViewString(0) = "Views.ViewA.UserCamera"
      ViewString(1) = "Views.ViewB.UserCamera"
      ViewString(2) = "Views.ViewC.UserCamera"
      ViewString(3) = "Views.ViewD.UserCamera"
      For   i = 0 To 3

            'Setzen einer Expression auf die User-Kamera-Parameter
            'Muster:
            'CopyPaste in_obj & ".DogView-Cam.PAR_NAME",, ViewString(i) & ".PAR_NAME", Fc

Sub   UnlockCam(in_obj)   'Unlock User-Cam in all Views
      Dim ViewString(3)
      ViewString(0) = "Views.ViewA.UserCamera"
      ViewString(1) = "Views.ViewB.UserCamera"
      ViewString(2) = "Views.ViewC.UserCamera"
      ViewString(3) = "Views.ViewD.UserCamera"
      For   i = 0 To 3

            'Löschen der Expression aud die User-Kamera-Parameter
            'Muster:
            'RemoveAnimation ViewString(i) & ".PAR_NAME", Fc
```

Abb. 6.10: Die Unterprogramme des Untermenüs „Face", zusätzlich zu denen, die ähnlich denen des Untermenüs „Selection" sind

6.5.4 Untermenü ‚Automation'

In diesem Untermenü kann durch Klick auf einen der Buttons eines von sechs Parametersets aktiviert werden (Abb. 6.11). Sofern ein Property-Editor geöffnet ist und bereits ein Custom Parameterset anzeigt wird, wird das aktive Parameterset automatisch in diesem Property-Editor dargestellt.

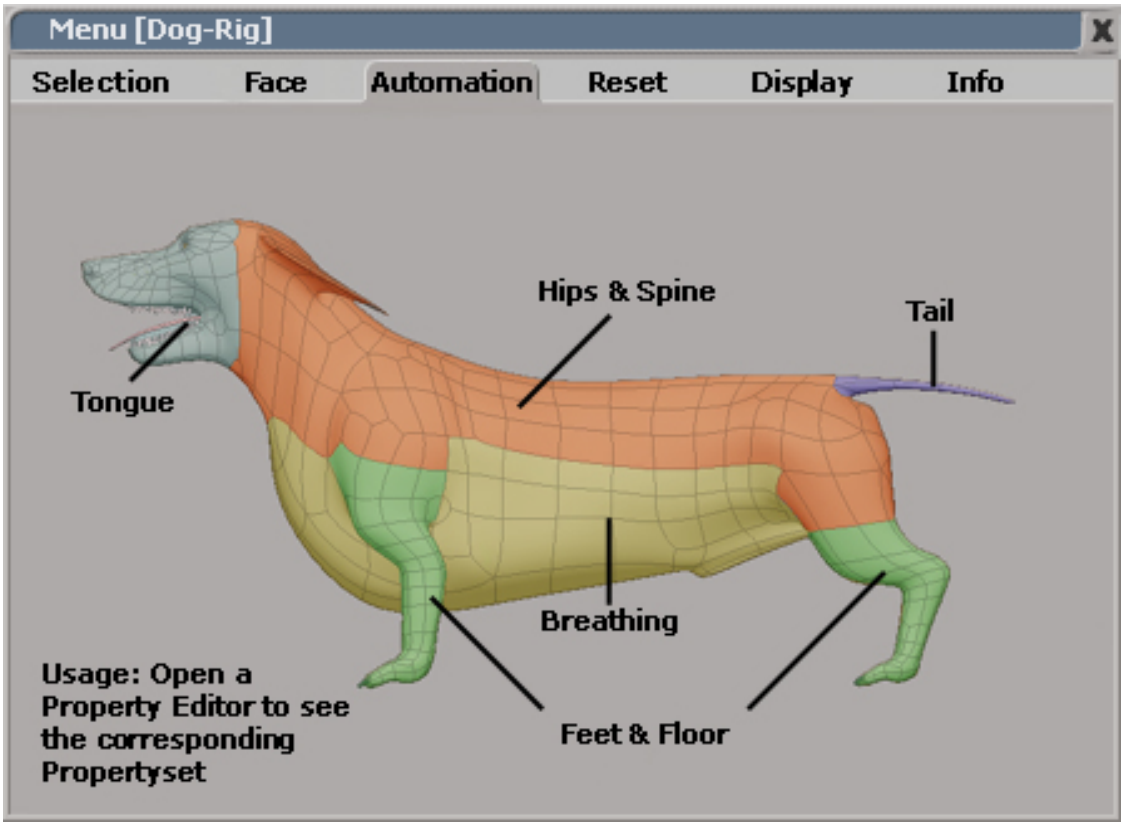

Abb. 6.11: Die Oberfläche des Untermenüs „Automation"

Das Aktivieren der Parametersets erfolgt mithilfe des Befehls „SelectObj". Auf ergänzende Struktogramme zu diesem Untermenü wird hier aufgrund der Trivialität verzichtet.

Die in den Parametersets enthaltenen Parameter sind zum größten Teil Proxy Parameter, die mit Parametern des Typs „Variablen" verknüpft sind. Es sind die in Kapitel 5.5 für Automationen verwendeten Variablen, die für die Verwendung als Front-End mit aussagekräftigeren Namen versehen wurden.

6.5.5 Untermenü ‚Reset'

In diesem Untermenü können durch Klick auf einen, einem Steuerungsobjekt zugeordneten, Button alle animierbaren, lokalen Transformationen des entsprechenden Steuerungsobjektes auf den Wert (0|0|0) gesetzt werden, was der Standardposition entspricht. Der Button „Reset All" bewirkt das gleiche für alle Steuerungsobjekte (Abb. 6.12).

Dazu wurde eine Funktion geschrieben, „resetObj", die von dem Untermenü des gedrückten Buttons, neben dem obligatorischen „in_obj", einen String mit dem Namen des zurückzusetzenden Steuerungsobjektes und sechs Boolesche Variablen übergeben bekommt. Die Booleschen Variablen legen fest, welche Parameter bei dem speziellen Steuerungsobjekt auf null zurückgesetzt werden müssen (Abb. 6.13, Abb. 6.14).

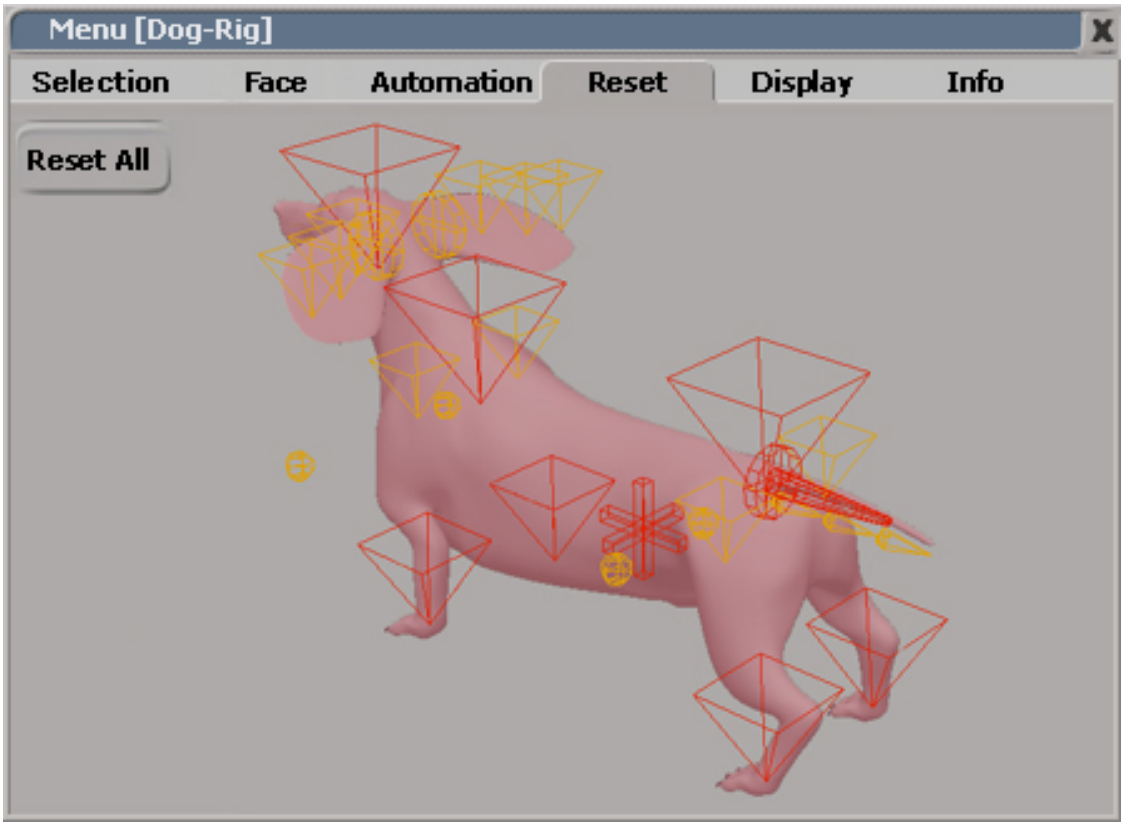

Abb. 6.12: Die Oberfläche des Untermenüs „Reset"

Funktion zum Zurücksetzen einer Kontrollobjektes

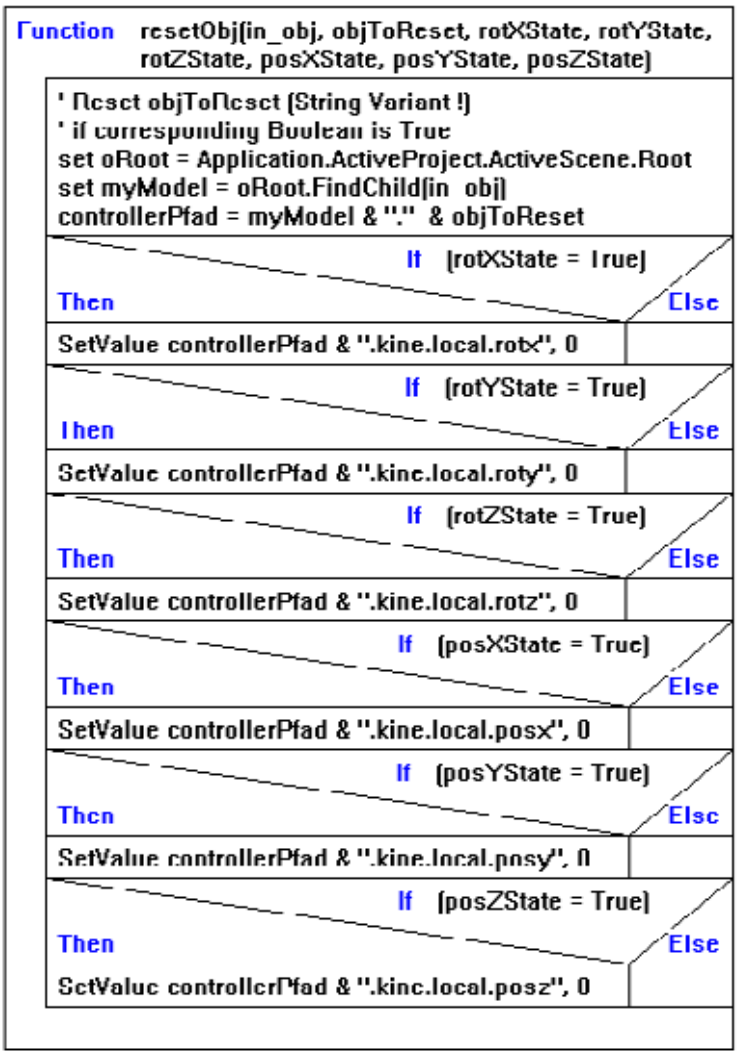

Abb. 6.13: Funktion zum Zurücksetzen der Steuerungsobjekte

Zurücksetzen der Kontrollobjekte, Aufruf der Funktion 'resetObj'

```
Sub   Chest(in_obj)
    resetObj in_obj, "ChestController", True, True, True, True, True, True

Sub   Tail(in_obj)
    resetObj in_obj, "TailController", True, False, True, False, False, False

Sub   ...
    ...
```

Zurücksetzen des COG-Kontrollobjektes
(Ausnahme: ohne Aufruf der Funktion 'resetObj')

```
Sub   COG(in_obj)  'COG is returned to Default ( Not(0|0|0) )
    set oRoot = Application.ActiveProject.ActiveScene.Root
    set myModel = oRoot.FindChild(in_obj)
    controllerPfad = myModel & ".COGController"
    SetValue controllerPfad & ".kine.local.rotx", 0
    SetValue controllerPfad & ".kine.local.roty", 0
    SetValue controllerPfad & ".kine.local.rotz", 0
    SetValue controllerPfad & ".kine.local.posx", 0
    SetValue controllerPfad & ".kine.local.posy", 3.5
    SetValue controllerPfad & ".kine.local.posz", -1.5
```

Abb. 6.14: Aufruf der Funktion zum Zurücksetzen der Steuerungsobjekte (oben). Zum Zurücksetzen des COG-Steuerungsobjektes (unten) wird die Funktion nicht aufgerufen, da er standardmäßig nicht im Koordinatenursprung liegt.

6.5.6 Untermenü ‚Display'

Über das Untermenü „Display" kann die Sichtbarkeit von Gruppen umgestellt werden, die verschiedenen Elemente des Rigs enthalten. Im Einzelnen sind dies: Alle Steuerungsobjekte, die Steuerungsobjekte zur Steuerung des Gesichts, alle Bones, alle Muscle-Deformer, alle Kurven zur Simulation der Gesichtsmuskeln, die Envelope- und die Haar-Objekte des Modells (Abb. 6.15). Dabei ist es empfehlenswert, die Anzeige der Haar-Objekte in den Viewports zu unterdrücken, da dies die Darstellung stark verlangsamt.

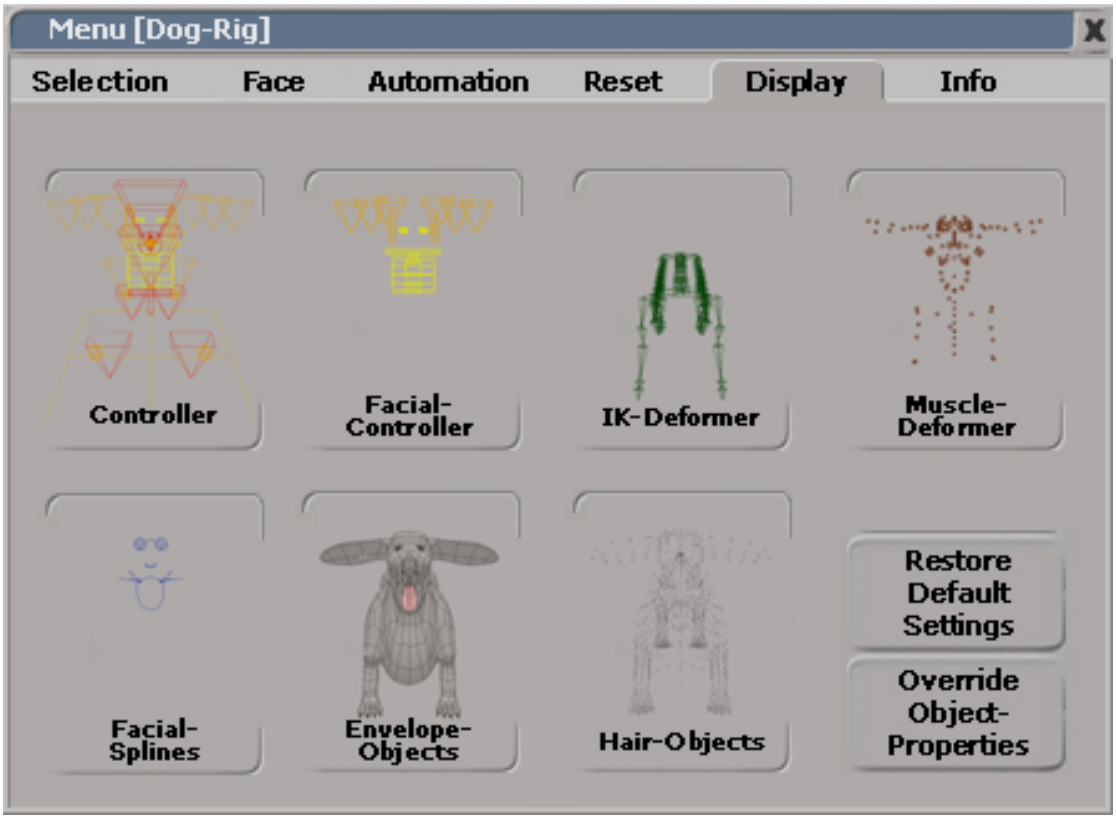

Abb. 6.15: Die Oberfläche des Untermenüs „Display"

Des weiteren wurde der Button „Restore Default Settings" implementiert, der die Sichtbarkeit aller Gruppen auf den Standardwert zurückstellt, sowie der Button „Override Object-Properties". Dieser stellt den Wert der „Override Object-Properties"-Variable um, welche die Anzeige der Objekt-spezifischen Anzeige-Einstellungen, wie etwa die Drahtgitter-Darstellung der Steuerungsobjekte unterdrückt, beziehungsweise nicht unterdrückt (Abb. 6.16).

```
Unterprogramme für das Verändern der
Anzeige der verschiedenen Gruppen

Sub   ToggleControllerDisplay(in_obj)
    Group1 = in_obj & ".Controller"
    Group2 = in_obj & ".FacialController"
    CurrentMode = GetValue ( Group1 & ".viewvis" )
    If-Elself
        If   CurrentMode = 1
            SetValue Group1 & ".viewvis", 0
            SetValue Group2 & ".viewvis", 0
        Elself   CurrentMode = 0
            SetValue Group1 & ".viewvis", 1
            SetValue Group2 & ".viewvis", 1
        Else
            
Sub   ...
    ...

Unterprogramm zum Wiederherstellen der Standardwerte

Sub   RestoreDefaults(in_obj)
    SetValue in_obj &  ".Controller"& ".viewvis", 1
    SetValue in_obj &  ".FacialController"& ".viewvis", 1
    SetValue in_obj &  ".IK Deformer"& ".viewvis", 0
    SetValue in_obj &  ".Muscle-Deformer"& ".viewvis", 0
    SetValue in_obj &  ".MuscleCurves"& ".viewvis", 0
    SetValue in_obj &  ".ModelObjectsGroup"& ".viewvis", 1
    SetValue in_obj &  ".HairGroup"& ".viewvis", 0

Unterprogramm zum Umschalten der
"Override-Objects-Properties"-Variable

Sub   OverrideProperties(in_obj)
    'Toggle Property in all Views:
    ToggleValue "mixviewmode", "Views.*.*.camdisp"

    'Toggle Property in all Cameras:
    ToggleValue "mixviewmode", "*.camdisp"
```

Abb. 6.16: Die Funktionen und Unterprogramme des Untermenüs „Display“

6.5.7 Untermenü ‚Info'

Das Untermenü „Info" enthält ausschließlich nicht-interaktiven Inhalt. Dieser beinhaltet unter Anderem Informationen zu dem Autor der Arbeit und dem Rahmen in dem diese Arbeit erstellt wurde (Abb. 6.17).

Abb. 6.17: Die Oberfläche des Untermenüs „Info"

6.5.8 Zusammenfassung

Die im Rahmen dieses Kapitels vorgestellte Synoptic View eignet sich für die Gestaltung grafischer Benutzeroberflächen vor allem aufgrund der Plattformenabhängigkeit und der guten Integration in XSI. Sie lässt dem Entwickler aber auch alle nötigen Freiheiten zur Entwicklung und Präsentation umfangreicher und innovativer Funktionen zur Steuerung eines Charakters. Dabei wird dem Entwickler durch die Verwendung bekannter Konzepte wie etwa der HTML-Imagemap oder der Unterstützung verschiedener Skriptsprachen die Arbeit erleichtert.

Im Rahmen dieses Kapitels wurde die Implementierung verschiedener sinnvoller und teilweise innovativer Funktionen mithilfe von Struktogrammen dokumentiert.

7 Die Anwendung in der Praxis

7.1 Einleitung

Im Rahmen dieses Kapitels sollen noch einmal die verschiedenen Hinweise zur Animation mit dem im Rahmen dieser Arbeit entstandenen Rig, die zum größten Teil schon im Rahmen eines anderen Kapitels besprochen wurden, zusammengetragen werden. Außerdem sollen Einzelbilder einiger erster Ergebnisse der Arbeit mit „Willy“ die Fähigkeiten des Rigs demonstrieren.

7.2 Animation mit dem Rig

Bei der Arbeit mit dem vorliegendem Rig muss, genau wie bei jedem anderen computergenerierten Charakter, beachtet werden, dass der Bewegungsspielraum eines Charakters immer seine Grenzen hat. Dies ist nicht zu vermeiden und nur natürlich, schließlich hat auch ein lebender Organismus eng gesetzte, physikalisch begrenzte Bewegungsmöglichkeiten. Auf eine strikte Begrenzung der Bewegungsmöglichkeiten durch etwa Rotation Limits wurde bei der Erstellung dieses Rigs bewusst verzichtet (vergleiche Kapitel 5.3.1), aber auch bei Verwendung dieser Beschränkungen sind immer gewisse Bereiche vorhanden, in denen die Bewegung eines Charakters nicht mehr realistisch ist. Die beste Möglichkeit, um aus der Überschreitung dieser Grenzen resultierende Fehler zu vermeiden, ist die Kenntnis der Grenzen. Dies lässt sich in der Praxis nur durch praktische Übung mit dem Rig erreichen.

> „You need to know the Limits of your Character, you never know what the Animator does” (Quelle: Vortrag „The Hulk“ von Gerald Gutschmidt, Computer Graphics Supervisor, Industrial Light & Magic im Rahmen der eDIT|VES 2003, 28.09.2003).

Um eine effiziente Low-Level-Animation mit der implementierten Charaktersteuerung zu erreichen und um die dazu entwickelten Techniken gut ausnutzen zu können, besteht nach Meinung des Autors die ideale Anordnung der Elemente der grafischen Benutzeroberfläche für die Animation sowohl aus einer Synoptic View und einem Property Editor zum Anzeigen der zweidimensionalen Steuerungselemente. Weiterhin sollte in einem Viewport eine Ansicht auf die Szene aus der Sicht der User-Camera dargestellt werden, um die entwickelten Funktionen des Untermenüs „Face“ ausnutzen zu können (Abb. 7.1).

Für eine derart eingerichtete Benutzeroberfläche wäre die Erstellung eines benutzerdefinierten Layouts nicht nötig, da das standardmäßige Layout von Softimage|XSI zu diesem Zweck modifiziert werden kann. Um in dem geöffneten Property-Editor die animierbaren Transformationen der Steuerungsobjekte und die Parametersets des Untermenüs „Automation“ anzeigen zu können, muss in diesem zunächst einmalig, durch Selektion beispielsweise eines der Marking Sets, manuell ein Custom Parameterset angezeigt werden. Anschließend wird bei Selektion eines Steuerungsobjektes das entsprechende Marking Set automatisch in diesem Property-Editor angezeigt.

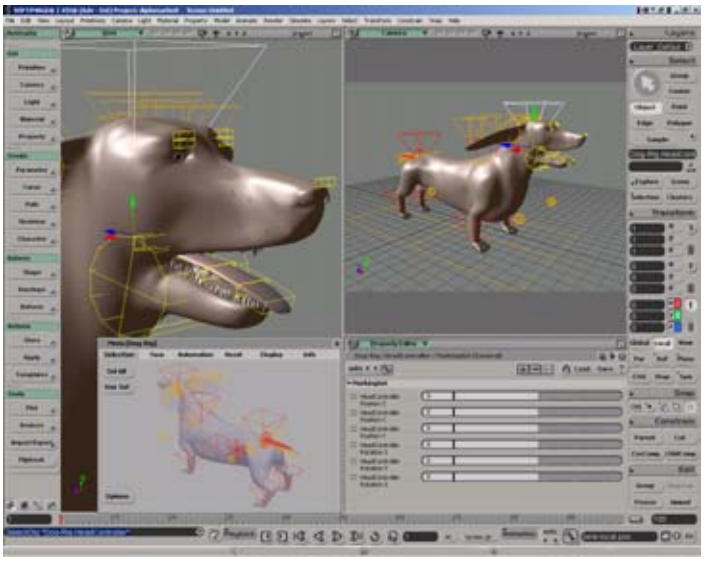

Abb. 7.1: Eine, für die Arbeit mit dem Rig sinnvolle, Anordnung: User-Camera, Standard-Kamera, Property-Editor und Synoptic View

Es ist naturgemäß sehr schwierig, bewegte Bilder in einer schriftlichen Arbeit darzustellen (Abb. 7.2, Abb. 7.3). In diesem Kapitel werden daher nur einige ausgewählte Einzelposen präsentiert, für weitere Beispiele sei hier auf die dieser Arbeit angelegte Daten-CD-Rom verwiesen („Anhang E: Inhalt der CD-Rom“).

Abb. 7.2: Verschiedene Ansichten von „Willy"

7.3 Zusammenfassung

Im Rahmen dieses Kapitels wurde die technische Bedienung des entwickelten Charakters besprochen. Voraussetzung für das Erstellen überzeugender Animationen ist allerdings weiterhin Übung im Umgang mit diesem Rig und seiner Steuerungsmethode. Die Erfahrungen und das Fachwissen über das Animieren von Lebensformen im Allgemeinen ist selbstverständlich durch kein noch so gutes Control-Rig zu ersetzen.

Abb. 7.3: Weitere Szenenfotos von „Willy"

8 Fazit

Die vorliegende Arbeit hat den Prozess der Charakteranimation in Theorie und Praxis zum Thema. Es wurde ein Ansatz für die wissenschaftliche Bearbeitung eines technisch schwierigen und künstlerisch fordernden Vorganges gesucht und gefunden.

Im Rahmen dieser Arbeit wurde die Entstehung eines kompletten Character-Control-Rigs beobachtet, von der Vorbereitungsphase bis zur Anwendung. Dabei wurden verschiedenste Techniken kombiniert und der aktuelle Stand der Technik aufgezeigt. Es wurde ein vorzeigbarer Charakter entwickelt, um die Praxisnähe des Projektes zu demonstrieren. Am Ende dieser Arbeit steht ein praxistaugliches Endergebnis, sowie eine theoretische Gesamtübersicht über den technischen Hintergrund der Animation computergenerierter Charaktere.

Für zukünftige Projekte wurde beim Entwurf des im Rahmen dieser Arbeit erstellten Rigs besonders auf dessen Skalierbarkeit geachtet. So ist es durch einfache, dokumentierte Mittel möglich, es in kürzester Zeit und mit geringem Aufwand an andere Hunde, und, im Rahmen, auch generell an andere Vierbeiner anzupassen. Des weiteren sind Elemente des Rigs getrennt von der Gesamtkonstruktion für weitere Projekte nutzbar. Als Beispiel soll hier die Schwanzkonstruktion genannt sein. Das im Rahmen dieser Arbeit zusammengetragene und entwickelte Know-how kann, unabhängig von einer expliziten Verwendung konkreter Elemente des Rigs, für Projekte aus dem ganzen Bereich der Charakteranimation und darüber hinaus eingesetzt werden.

Auch das erstellte Konzept zur Steuerung des Charakters „Willy“ kann dennoch keinesfalls ein in jeder Situation perfektes Rig darstellen. Es ist eher der Extremfall eines Rigs, das für eine regelmäßige Anwendung in einem großen Rahmen erdacht wurde. Für die meisten kommerziellen Produktionen lässt das zeitliche und finanzielle Budget ein derart weit entwickeltes Rig nicht zu.

Es ist auch zu bedenken, dass es kaum das perfekte Rig geben wird, dass für jede Situation die beste Lösung darstellt. Das entwickelte Rig versucht aber, am ehesten diesen Zielen nahe zu kommen und dabei auch Anregungen zur Entwicklung zukünftiger Methoden zu geben.

Es bleibt zu hoffen, dass die wirtschaftliche und künstlerische Bedeutung der Charakteranimation in Deutschland in Zukunft weiter an Zuwachs gewinnt, womit eine verstärkte Nutzung dieser Technologien verbunden wäre.

Anhang A: Literaturverzeichnis

Arima2001 **Arima, Shinsaku; Rossano, Anthony:** *XSI Illuminated: Foundation.* Mesner Inc. 2001. ISBN: 0-9707530-0-7

Birn2001 **Birn, Jeremy:** *Lighting und Rendering*. München: Markt + Technik Verlag, 2001 ISBN: 3-8272-5969-X

Efstathiou2003 **Efstathiou, Sofronis:** *Creating Eyeballs in Softimage XSI.* [ONLINE]. Stand: 24.05.2003; URL: http://www.onionboy.co.uk/tutorials_eye.html

Faigin1990 **Faigin, Gary:** *The Artist's Complete Guide to Facial Expressions.* New York, USA: Watson-Guptill Publications, 1990. ISBN: 0-8230-1628-5

Hayes2003 **Hayes, Andy**: *Lecture Notes on Scripting*. [ONLINE]. Stand: 24.05.2003; URL: http://andymator.co.uk/scripting.htm

Helzle2003 **Helzle, Thomas; Desse, Christophe:** *Wie Sie lebendige Character entwerfen: Tipps aus der Praxis.* In: Digital Production, 03/2003, S. 116 – 121

Hogarth1990 **Hogarth, Burne:** *Dynamic Anatomy.* New York, USA: Watson-Guptill Publications, 1990. ISBN: 0-8230-1551-3

Koenigsmarck2000 **Koenigsmarck, Arndt von:** *3D Character Design.* Bonn: Galileo Press GmbH, 2000. ISBN: 3-934358-13-6

Maraffi2001 **Maraffi, Chris:** *Softimage|XSI Character Animation: f/x & Design.* Scotsdale, Arizona USA: The Coriolis Group, 2001. ISBN: 1-57610-642-X

McCann2003 **McCann, Jaimy:** *Photoshop Illuminated: For Animators*. Mesner Inc., 2003. ISBN: 0-9707530-3-9

Microsoft2001 **Microsoft Corporation:** *Windows Script Technologies.*
[ONLINE] Copyright: Microsoft Corporation, 2001, Topic Version 5.6.9309.1546;
URL: http://msdn.microsoft.com/scripting

Misner2001 **Isner, Michael:** *XSI Scripting: Commands VS Object Model*.
[ONLINE] Copyright: Michael Isner, 2001, Stand: 24.05.2003;
URL: http://www.isner.com/tutorials /commands_vs_object_model2.htm

Misner2003a **Isner, Michael:** *Poly Texturing Primer*.
[ONLINE] Copyright: Michael Isner, 2003, Stand: 09.07.2003;
URL: http://www.edharriss.com/tutorials /tutorial_xsi_isner_texturemaping/ texture_tutorial.htm

Misner2003b **Isner, Michael:** *Introducing the Isner Spine*.
[ONLINE] Copyright: Michael Isner, 2003, Stand: 24.05.2003;
URL: http://www.isner.com/isnerspine/spine_introduction.htm

Murtak2003 **Murtak, Josh:** *Scripted Operator Shoulder.*
[ONLINE]. Stand: 24.05.2003;
URL: http://www.softimage.com/education/Xsi/ SelfPacedLearning/Tutorials/webTutorials/XSI_3_0/scriptedop/scriptedop.pdf

Muybridge1957 **Muybridge, Eadweard:** *Animals in Motion*.
New York, USA: Dover Publications, Inc., 1957.
ISBN: 0-486-20203-8

Muybridge1979 **Muybridge, Eadweard:** *Muybridge's Complete Human and Animal Locomotion*.
New York, USA: Dover Publications, Inc., 1957.
ISBN: 0-486-23793-1

Rabiler2001 **Rabiler, Guy:** *Script - Commands vs ObjectModel: Enlightments on the two XSI Scripting approach.* [ONLINE]
Copyright: Guy Rabiler, 2001, Stand 24.05.2003;
URL: http://grabiller.3dvf.net/site/content/ tut_script_commandsvsobjects_eng.html

Rahman2003 **Abdel Rahman, Nora:** *Terminator 3: Von Zeitschleifen und Flüssig-Simulationen.*
In: Digital Production, 04/2003, S. 32 – 36

Rosa2003 **Rosa, Roger:** *Modeling a Dog using Subdivision-Surfaces*.
[ONLINE]. Stand: 24.05.2003;
URL: http://roger.rosa.free.fr/resources/html/tutorials/xsi /model/labrador/labrador_01.htm

Rossano2002 **Rossano, Anthony:** *XSI Illuminated: Character*.
Mesner Inc., 2002.
ISBN: 0-9707530-4-7

Rossano2003 **Rossano, Anthony:** *XSI Illuminated: Expressions*. [ONLINE].
Copyright: Mesmer Animation Labs, Stand: 24.05.2003;
URL: http://enemy.mesmer.com/moebius/FMPro?-db=article&-format=free_tutorial_read.html&-op=eq&ProductName=XSI%20Expressions&-max=1&-find=

Thomas1981 **Thomas, Frank; Johnston, Ollie:** *The Illusion of Life:Disney Animation.*
New York, USA: Disney Editions, 1981.
ISBN: 0-7868-6070-7

Watt2002 **Watt, Alan**: *3D-Computergrafik. 3. Auflage.*
München: Pearson Education Deutschland GmbH, 2002.
ISBN: 3-8273-7014-0

Williams2001 **Williams, Richard:** *The Animator's Survival Kit: A Manual of Methods, Principles and Formulars for Classical, Computer, Games, Stop Motion and Internet Animators.*
London, UK: Faber and Faber Limited, 2001.
ISBN: 0-571-20228-4

XsiDoc2002 **Softimage|XSI Documentation Team:**

- *Softimage|XSI Version 3.0 Documentation CD: Fundamentals.* [CD-ROM].
  Avid Technology, Inc. 2002, Part No. 0130-05536-01
- *Softimage|XSI Version 3.0 Documentation CD: Setup & Licensing.* [CD-ROM].
  Avid Technology, Inc. 2002, Part No. 0130-05542-01
- *Sofftimage|XSI Version 3.0 Documentation CD: Modeling & Deformations.* [CD-ROM].
  Avid Technology, Inc. 2002, Part No. 0130-05539-01
- *Softimage|XSI Version 3.0 Documentation CD: Animation*. [CD-ROM].
  Avid Technology, Inc. 2002, Part No. 0130-05534-01
- *Softimage|XSI Version 3.0 Documentation CD: Simulation.* [CD-ROM].
  Avid Technology, Inc. 2002, Part No. 0130-05543-01
- *Softimage|XSI Version 3.0 Documentation CD: Shaders, Lights & Cameras*. [CD-ROM].
  Avid Technology, Inc. 2002, Part No. 0130-05537-01
- *Softimage|XSI Version 3.0 Documentation CD: Rendering & Compositing.* [CD-ROM].
  Avid Technology, Inc. 2002, Part No. 0130-05540-01
- *Softimage|XSI Version 3.0 Documentation CD: Scripting*. [CD-ROM].
  Avid Technology, Inc. 2002, Part No. 0130-05538-01
- *Softimage|XSI Version 3.0 Documentation CD: Tutorials.* [CD-ROM].
  Avid Technology, Inc. 2002, Part No. 0130-05544-01

Anhang B: Ausgewählte Expertenmeinungen

Im Rahmen dieser Arbeit wurde ein Fragebogen erstellt, der die Aufgabe hatte, die unterschiedlichen Methoden zur Steuerung eines Charakters auf ihre Praxistauglichkeit hin zu untersuchen. Zweck des Fragebogens war auch, die Verbreitung dieser Techniken in deutschen Unternehmen zu ermitteln. Dabei ist natürlich zu beachten, dass ein komplexeres Charaktersetup, wie es hier besprochen wird, im professionellen Umfeld nur rentabel ist, wenn ein Charakter in einem großen Stil und regelmäßig eingesetzt wird. Wie auch die Auswertung der Fragebogen zeigte, ist dies in Deutschland noch relativ selten vorzufinden.

Es konnten dennoch drei Experten gefunden werden, die bereit waren, einige Fragen bezüglich der Steuerung von Charakteren zu beantworten.

Im einzelnen waren dies:

- Sebastian Faber, LIGA_01 Computerfilm
- Thomas Helzle, ScreenDream
- Jörg Liebold, Freelancer

An dieser Stelle soll ausdrücklich allen Interviewten gedankt werden, die einen Teil ihrer Zeit für diesen Fragebogen opferten.

Es sei noch erwähnt, dass die vorliegenden Expertenmeinungen keinesfalls repräsentativ sind, die Antworten spiegeln nur die Antworten einzelner ausgesuchter Personen dar. Auch sollen die ausgesuchten Zitate, die den ausgefüllten Fragebogen entnommen wurden, nicht für die Meinung anderer stehen oder eine Gesamtmeinung wiederspiegeln.

1. Frage zu pers. Angaben, hier nicht aufgezeigt.

2. Haben Sie/Ihr Unternehmen bereits Erfahrungen im Bereich Charakteranimation gesammelt?	Sebastian Faber	Thomas Helzle	Jörg Liebold
Ja, viel Erfahrung	X		X
Ja, etwas Erfahrung		X	
Nein			

3. Welche Software verwenden Sie/Ihr Unternehmen hauptsächlich im Bereich Charakteranimation?	Sebastian Faber	Thomas Helzle	Jörg Liebold
	Softimage|XSI, Maya	Lightwave, Messiah:Studio, Kaydara Motionbuilder	Softimage|XSI

4. Ist in Ihrem Unternehmen jeweils ein bestimmter Animator für einen speziellen Charakter zuständig oder wechseln die Animatoren (wenn ja, wie häufig etwa)?	Sebastian Faber	Thomas Helzle	Jörg Liebold
Ja	X	X	X
Nein			

- Sebastian Faber: „Jobabhängig. Generell versuchen wir einen Character immer vom selben Animator animieren zu lassen, dies ist je nach Umfang oder aus Produktionstechnischen Gründen nicht immer möglich."
- Thomas Helzle: „Bei kleinen Projekten und wenn vermeidbar, wechselt der Animator nicht – aber nicht immer ist das möglich..."

5. Verwenden Sie Slider zum Steuern von Charakteren? (In XSI etwa in Kombination mit der Synoptic View).	Sebastian Faber	Thomas Helzle	Jörg Liebold
Ja		X	X
Nein			

- Sebastian Faber: „Eher Selten. Bei Lipsync-Animation häufiger ansonsten so gut wie nie. Das von Ihnen dargestellte Beispiel findet in der Praxis bei erfahrenen Animators kaum Anklang. Gute Animator arbeiten am liebsten direkt in den Function-Curves, selbst ausgereifte Hilfsmittel, wie der Animation-Mixer von XSI werden von unseren Animators nur bedingt akzeptiert."
- Thomas Helzle: „Messiah bietet seit 1999 OpenGL-Slider direkt im Viewport, die mit Expressions mit beliebigen Funktionen belegt werden können. Für Morphing, Handstellungen und andere „vorgefertigte" Zustände ist dies sehr effizient. Auch Gesichtsanimation über Bone-Rigs lassen sich durch Zuordnung kompletter Posen flexibel abrufen und mixen. Daneben sind Slider unverzichtbar für Dynamic Parenting, IK-FK-Blending und viele weitere Hilfsfunktionen."

6. Verwenden Sie eigens programmierte Skripts bzw. Buttons o.ä. zum Vereinfachen der Arbeit mit einem Charakter (beispielsweise Skripts zum Selektieren von Effektoren)?	Sebastian Faber	Thomas Helzle	Jörg Liebold
Ja	X	X	X
Nein			

- Thomas Helzle: „Ja, soweit möglich schon – hier (also Buttons zu Selektion etc.) ist Messiah allerdings nicht ganz optimal ausgestattet, was sich erst in einer nächsten Version ändern soll. Aber grundsätzlich verwende ich natürlich eine Menge selbstgeschriebener Scripte..."

7. Verwenden sie Dummy-Objekte als Anfasser am Charakter selbst zum Steuern von Charakteren?	Sebastian Faber	Thomas Helzle	Jörg Liebold
Ja	X	X	X
Nein			

8. Verwenden sie Dummy-Objekt als Konsole arrangiert zum Steuern von Charakteren?	Sebastian Faber	Thomas Helzle	Jörg Liebold
Ja			
Nein	X	X	X

- Thomas Helzle: „Es sei denn, man bezeichnet die in den Viewport integrierten Slider als Konsole... ;-)"

9. Automatisieren Sie verschiedene Bewegungen Ihrer Charaktere durch Skripts (Expressions)? Wenn ja, wie stark sind Ihre Charaktere automatisiert?	Sebastian Faber	Thomas Helzle	Jörg Liebold
Ja	X	X	X
Nein			

- Sebastian Faber: „Der Grad der Automatisierung hängt stark von der Dauer der Animation ab. Bei Charakteren für einen Spielfilm oder eine Serie lohnt eine stärkere Automatisierung, da sich der Aufwand über die Distanz rechnet. Im Commercial-Bereich sind nur die Grundfunktionen der Character automatisiert (Abrollen der Füße, Wirbelsäule etc.)."

- Thomas Helzle: „Eigentlich gehört hier ein „Jein“ hin. Automatisierung ist gefährlich, aber oft unumgänglich...

  Wichtig ist, dass jede Automatisierung auch ausgeblendet werden kann. Die Zentrierung der Hüfte zwischen den Beinen ist toll für Demonstrationen und wirkt „Cool“, aber wenn die Figur einen Salto macht, sich hinsetzt etc. nervt sie sehr schnell. Der Schwerpunkt bei einem Menschen ist nahezu nie ganz da wo ihn die Expression hinsetzt. Beim Gehen „fällt“ man ja immer ein wenig nach vorn, die Hüfte wandert zwar teilweise, aber nicht komplett über das momentan auf dem Boden befindliche Bein... Ich habe einige Riggs anderer Animatoren gesehen, die zusätzlich zur Hüft-Automation noch eine Vielzahl an Reglern benötigten, um die Hüfte mehr links, rechts, vorn oder hinten zu platzieren... Das ist dann häufig leichter von Hand animiert... ;-)

  Für mich heißt die Regel bei Automation: So wenig wie möglich, so viel wie nötig. Außerdem versuche ich hauptsächlich Dinge zu automatisieren, die auch bei Lebewesen automatisch gehen, also fixe Abhängigkeiten die durch Sehnen, Knochen oder Notwendigkeit verursacht sind...“

10. Falls Sie Softimage|XSI verwenden: Sind Ihre Rigs bezüglich Topologie und Benennung der Bones kompatibel zum XSI-Standardmodell, bzw. richten Sie sich nach einem anderen Standard?	Sebastian Faber	Thomas Helzle	Jörg Liebold
Ja	X		X
Nein			

- Sebastian Faber: „Wir verwenden unsere eigenen Standards und eigene Tools zur Benennung der einzelnen Bones etc.“

11. Falls Sie Softimage|XSI verwenden: Welche der folgenden Technologien/Tools verwenden Sie zur Steuerung Ihrer Charaktere?	Sebastian Faber	Thomas Helzle	Jörg Liebold
Expressions	X		
Scripted Operator	X		
Custom Views/Toolbars	X		X
Synoptic View	X		
Net View			

12. Welche der folgenden Vorzüge einer guten Animationssteuerung würden Sie wie stark bewerten? (1=sehr wichtig, 5=vollkommen unwichtig)	Sebastian Faber	Thomas Helzle	Jörg Liebold
Zeitersparnis	5	3	5
Mehr Flexibilität im Produktionsprozess, z. B. wegen besserer Möglichkeiten für einen Wechsel des Animators	5	3	3
Bessere Qualität der Animation	5	3	4
Sonstige: (Bitte angeben)			

- Sebastian Faber: „Abgesehen von Automatisierungen – via Expressions und Scripted Operator, sowie im Bereich Lipsync ist mir keine Automatisierung bekannt, die das arbeiten mit Function-Curves ersetzen kann. Alle bisherigen Arbeiten diesbezüglich sei es per Midi-Interface, komplexe Slider-tables etc. haben sich als ineffektiv herausgestellt und sich von daher auch nicht durchgesetzt. Eine weitere Abstraktionsebene verschlechtert eher das Ergebnis der Characteranimation – und macht den Animator unflexibel und langsam."
- Thomas Helzle: „Dieser ganze Bereich ist extreme „Geschmackssache". Der eine Animator arbeitet ausschließlich mit FK, der andere mit IK, der dritte mit kompletter Automation... [...]„

13. Welche der folgenden Methoden der Animationssteuerung halten Sie für den Produktionsbereich am sinnvollsten?	Sebastian Faber	Thomas Helzle	Jörg Liebold
Verwenden von Slidern zum Steuern des Charakters			X
Verwenden von Dummy-Objekten an dem Charakter selbst	X		
Verwenden von Dummy-Objekten als Konsole arrangiert			
Sonstiges bzw. Kombination daraus (Bitte angeben)	X		

- Sebastian Faber: „Expressions zur Steuerung von Sekundäranimation (Muskelsysteme, Wirbelsäule etc.)"
- Thomas Helzle: „Auch hier gilt: „Whatever makes you happy" ;-) Je nach verwendetem Programm und Animationsparadigma unterscheiden sich diese Dinge stark. Ich halte alle obigen Methoden für sinnvoll – je nach Anwendung. Daneben gibt es aber noch eine Reihe weiterer, die auch ihren Sinn haben...[...]"

Anhang C: Parametertabellen

Tabelle C.1: Die verwendeten Variablen

Name des Parametersets	Name des Parameters	Beschreibung	Controller	Proxy Parameter
var_FootRollSet	var_LeftBackRoll	Fußrollen/-Drehen	LeftBackFoot-Controller	MarkingSet des Controllers
var_FootRollSet	var_LeftFrontRoll	Fußrollen/-Drehen	LeftBackFoot-Controller	MarkingSet des Controllers
var_FootRollSet	var_RightBackRoll	Fußrollen/-Drehen	RightBackFoot-Controller	MarkingSet des Controllers
var_FootRollSet	var_RightFrontRoll	Fußrollen/-Drehen	RightFrontFoot-Controller	MarkingSet des Controllers
var_LeftBack_FKIKSwitchSet	var_LeftBackSwitch	FK-IK-Blending	/	FeetAndFloor_ Automation
var_LeftFront_FKIKSwitchSet	var_LeftFrontSwitch	FK-IK-Blending	/	FeetAndFloor_ Automation
var_RightBack_FKIKSwitchSet	var_RightBackSwitch	FK-IK-Blending	/	FeetAndFloor_ Automation
var_RightFront_FKIKSwitchSet	var_RightFrontSwitch	FK-IK-Blending	/	FeetAndFloor_ Automation
var_RipBreathSet	var_ManuBreath	Manuelles Weiten des Brustkorbs	/	Breathing_ Automation
var_RipBreathSet	var_AutoBreath Amount	Amplitude des automatischen Atmens	/	Breathing_ Automation
var_RipBreathSet	var_AutoBreath Frequency	Frequenz des automatischen Atmens	/	Breathing_ Automation
var_TailSet	var_TailExprManu Switch	Switch zwischen Expression-Driven und Manueller-FK	/	Tail_ Automation
TailController.	kine.local.rotz	Horizontale Drehung des Schwanzes	TailController	MarkingSet des Controllers
TailController.	kine.local.rotx	Vertikale Bewegung des Schwanzes	TailController	MarkingSet des Controllers
var_EarsRollSet	var_EarLeftRoll	Variable für "Ohr-Rollen"	EarLeftRoll_ Controller	EarRighRoll_ Controller
var_EarsRollSet	var_EarRightRoll	Variable für "Ohr-Rollen"	EarRightRoll_ Controller	EarRighRoll_ Controller
var_TongueSet	var_TonguePosition	Position der Zunge	/	Tongue_Automation
var_TongueSet	var_TongueBend(1-3)	Biegung der Zunge	/	Tongue_Automation
var_TongueAutomationSet	var_TongueAutomation Amount	Mitbewegung der Zunge mit dem Unterkiefer	/	Tongue_Automation

Name des Parametersets	Name des Parameters	Beschreibung	Controller	Proxy Parameter
var_AutomationsamountSet	var_HipRotationAmount	Automatische Drehung der Hüfte	/	HipsAndSpine_ Automation
var_AutomationsamountSet	var_ChestRotation Amount	Automatische Drehung der Hüfte	/	HipsAndSpine_ Automation
var_AutomationsamountSet	var_HipPosition Amount	Automatische Verschie-bung der Hüfte	/	HipsAndSpine_ Automation
var_AutomationsamountSet	var_ChestPosition Amount	Automatische Verschie-bung der Hüfte	/	HipsAndSpine_ Automation
var_FloorDetectionSet	var_EnableFloor Detection	Zustand der Kollisions-kontrolle	/	FeetAndFloor_ Automation
var_ShoulderOffsetAmountSet	var_ShoulderOffset AmountSide	Automatische Drehung der Schulter (Schulter vorne links)	/	HipsAndSpine_ Automation
var_ShoulderOffsetAmountSet	var_ShoulderOffset AmountUp	Automatische Drehung der Schulter (Schulter vorne links); per Expres-sion gleichgesetzt mit "var_ShoulderOffset AmountSide"	/	/
var_ShoulderOffsetAmountSet	var_ShoulderOffset AmountForward	Automatische Drehung der Schulter (Schulter vorne links); per Expres-sion gleichgesetzt mit "var_ShoulderOffset AmountSide"	/	/
var_ShoulderOffsetAmountSet	var_ShoulderOffset AmountSide	Automatische Drehung der Schulter (Schulter vorne rechts)	/	HipsAndSpine_ Automation
var_ShoulderOffsetAmountSet	var_ShoulderOffset AmountUp	Automatische Drehung der Schulter (Schulter vorne rechts); per Ex-pression gleichgesetzt mit "var_ShoulderOffset AmountSide"	/	/
var_ShoulderOffsetAmountSet	var_ShoulderOffset AmountForward	Automatische Drehung der Schulter (Schulter vorne rechts); per Ex-pression gleichgesetzt mit "var_ShoulderOffset AmountSide"	/	/
var_ShoulderOffsetAmountSet	var_ShoulderOffset AmountSide	Automatische Drehung der Schulter (Schulter hinten links)	/	HipsAndSpine_ Automation
var_ShoulderOffsetAmountSet	var_ShoulderOffset AmountUp	Automatische Drehung der Schulter (Schulter hinten links); per Expres-sion gleichgesetzt mit "var_ShoulderOffset AmountSide"	/	/

Name des Parametersets	Name des Parameters	Beschreibung	Controller	Proxy Parameter
var_ShoulderOffsetAmountSet	var_ShoulderOffset AmountForward	Automatische Drehung der Schulter (Schulter hinten links); per Expression gleichgesetzt mit "var_ShoulderOffset AmountSide"	/	/
var_ShoulderOffsetAmountSet	var_ShoulderOffset AmountSide	Automatische Drehung der Schulter (Schulter hinten rechts)	/	HipsAndSpine_ Automation
var_ShoulderOffsetAmountSet	var_ShoulderOffset AmountUp	Automatische Drehung der Schulter (Schulter hinten rechts); per Expression gleichgesetzt mit "var_ShoulderOffset AmountSide"	/	/
var_ShoulderOffsetAmountSet	var_ShoulderOffset AmountForward	Automatische Drehung der Schulter (Schulter hinten rechts); per Expression gleichgesetzt mit "var_ShoulderOffset AmountSide"	/	/
EyeRight_MuscleCurve_ ClosedShape_ShapeKey_Track.	EyeRight_MuscleCurve_ ClosedShape_ ShapeKey_Clip. actionclip.weight	Öffnung des Augenlids rechts	RightEyelid Controller	MarkingSet des Controllers
EyeLeft_MuscleCurve_ ClosedShape_ShapeKey_Track.	EyeLeft_MuscleCurve_C losedShape_ ShapeKey_Clip. actionclip.weight	Öffnung des Augenlids links	LeftEyelid Controller	MarkingSet des Controllers
EyeRight_MuscleCurveOutline_ BrowsUpShape_ShapeKey_Track.	EyeRight_MuscleCurveO utline_BrowsUpShape_ ShapeKey_Clip. actionclip.weight	Heben der Augenbrauen rechts	RightEyebrow Controller	MarkingSet des Controllers
EyeLeft_MuscleCurveOutline_ BrowsUpShape_ShapeKey_Track.	EyeLeft_MuscleCurveOu tline_BrowsUpShape_ ShapeKey_Clip. actionclip.weight	Heben der Augenbrauen links	LeftEyebrow Controller	MarkingSet des Controllers
Nose_MuscleCurve_ WideShape_ShapeKey_Track.	Nose_MuscleCurve_Wid eShape_ ShapeKey_Clip. actionclip.weight	"Röcheln"	NoseTipController	MarkingSet des Controllers
JawTop_MuscleCurve_ UpShape_ShapeKey_Track.	JawTop_MuscleCurve_U pShape_ ShapeKey_Clip. actionclip.weight	"Höhe" der Backen	JawController	MarkingSet des Controllers
JawTop_MuscleCurve_ WidenShape_ShapeKey_Track.	JawTop_MuscleCurve_ WidenShape_ ShapeKey_Clip. actionclip.weight	"Breite" der Backen	JawController	MarkingSet des Controllers
sowie die lokalen Rotations- und Translationswerte aller Controller die IK-Chains steuern		Diverse	Diverse	MarkingSet des Controllers

Tabelle C.2: Die verwendeten Konstanten

Name des Parametersets	Name des Parameters	Beschreibung	Pfad
const_SkullRadius	const_SkullRadius	Schädelgröße	HelperHierarchy.Head_Anchor. Skull_Dummy
const_RipSet	const_BoneLenght_Rip(1-9)	Rippenlänge insgesamt	HelperHierarchy.RipHierarchy. spineCurveBody
const_BodySpineSet	const_BodySpineScale	Körper-Spine-Länge	HelperHierarchy.RipHierarchy. spineCurveBody
const_NeckSpineSet	const_NeckSpineScale	Hals-Spine-Länge	HelperHierarchy.NeckHierarchy. spineCurveNeck
const_RipBreathRangesSet	const_Rip(1-9)_RotRange	Rotationsmöglichkeit bei der Atmung	HelperHierarchy.RipHierarchy
const_RipBreathRangesSet	const_Throat_SclWidth	Skalierrbarkeit des Halses (Breite) bei der Atmung	HelperHierarchy.RipHierarchy
const_RipBreathRangesSet	const_Throat_SclWidth	Skalierrbarkeit des Halses (Höhe) bei der Atmung	HelperHierarchy.RipHierarchy
const_TailBendSet	const_TailBendBone(1-8)	Default-Biegung des Schwanzes	COGController.HipController_Holder. HipController.HipController_Null. Hip_Anchor.TailHierarchy
const_TailFlexibilitySet	const_TailFlexibility Bone(1-8)	Gewichtung der Einzelnen Bones bezüglich Drehung	COGController.HipController_Holder. HipController. HipController_Null.Hip_Anchor. TailHierarchy
const_TailLenghtSet	const_TailLenght	Länge des Schwanzes	COGController.HipController_Holder. HipController.HipController_Null. Hip_Anchor.TailHierarchy
const_TailRollFlexibilitySet	const_TailRollFlexibility Bone(1-8)	Gewichtung der Einzelnen Bones bezüglich des Rollens	COGController.HipController_Holder. HipController.HipController_Null. Hip_Anchor.TailHierarchy
const_TailTimeOffset	const_TailTimeOffset Bone(1-8)	Zeitliche Verzögerung pro Bone in Sekunden	COGController.HipController_Holder. HipController.HipController_Null. Hip_Anchor.TailHierarchy
const_EarsLenghtSet	const_EarLenght	Länge der Ohren	HelperHierarchy.EarsHierarchy
const_EarsRollflexibilitySet	const_EarRollFlexibility Deformer(1-9)	Gewichtung der Einzelnen Ohrdeformer bezüglich Drehung	HelperHierarchy.EarsHierarchy
const_TongueDefault LenghtSet	const_TongueLenght	Default-Länge der Zunge	HelperHierarchy.TongueHierarchy
const_ChestPositionLimitSet	const_ChestPositionLimit	Maximale Verschiebung der Hüfte	COGController.ChestController_Holder. ChestController.ChestController_Null. Chest_Anchor
const_HipPositionLimitSet	const_HipPositionLimit	Maximale Verschiebung der Hüfte	COGController.HipController_Holder. HipController.HipController_Null. Hip_Anchor
const_ForeLegBoneLenghtSet	const_ForeLegShoulder	Knochenlänge Vorderbeine	COGController.ChestController_Holder. ChestController.ChestController_Null. Chest_Anchor.Chest_Anchor_Layer2
const_ForeLegBoneLenghtSet	const_ForeLegBone1	Knochenlänge Vorderbeine	COGController.ChestController_Holder. ChestController.ChestController_Null. Chest_Anchor.Chest_Anchor_Layer2

Name des Parametersets	Name des Parameters	Beschreibung	Pfad
const_ForeLegBoneLenghtSet	const_ForeLegBone2	Knochenlänge Vorderbeine	COGController.ChestController_Holder. ChestController.ChestController_Null. Chest_Anchor.Chest_Anchor_Layer2
const_ForeLegBoneLenghtSet	const_ForeLegBone3	Knochenlänge Vorderbeine	COGController.ChestController_Holder. ChestController.ChestController_Null. Chest_Anchor.Chest_Anchor_Layer2
const_ForeLegBoneLenghtSet	const_ForeLegPawBone1	Knochenlänge Vorderbeine	COGController.ChestController_Holder. ChestController.ChestController_Null. Chest_Anchor.Chest_Anchor_Layer2
const_ForeLegBoneLenghtSet	const_ForeLegPawBone2	Knochenlänge Vorderbeine	COGController.ChestController_Holder. ChestController.ChestController_Null. Chest_Anchor.Chest_Anchor_Layer2
const_HindLegBoneLenghtSet	const_HindLegShoulder	Knochenlänge Hinterbeine	COGController.HipController_Holder. HipController.HipController_Null. Hip_Anchor
const_HindLegBoneLenghtSet	const_HindLegBone1	Knochenlänge Hinterbeine	COGController.HipController_Holder. HipController.HipController_Null. Hip_Anchor
const_HindLegBoneLenghtSet	const_HindLegBone2	Knochenlänge Hinterbeine	COGController.HipController_Holder. HipController.HipController_Null. Hip_Anchor
const_HindLegBoneLenghtSet	const_HindLegBone3	Knochenlänge Hinterbeine	COGController.HipController_Holder. HipController.HipController_Null. Hip_Anchor
const_HindLegBoneLenghtSet	const_HindLegPawBone1	Knochenlänge Hinterbeine	COGController.HipController_Holder. HipController.HipController_Null. Hip_Anchor
const_HindLegBoneLenghtSet	const_HindLegPawBone2	Knochenlänge Hinterbeine	COGController.HipController_Holder. HipController.HipController_Null. Hip_Anchor
const_ShoulderOffsetSet	const_ShoulderOffset CurveSide	Kurve, die die Fussposition mit der Seitbewegung der Schulter verlinkt (Schulter vorne links)	COGController.ChestController_Holder. ChestController.ChestShoulderLeft Controller_Holder.ChestShoulderLeft Controller.ChestShoulderLeft_Null. ChestShoulderLeft_PointAt
const_ShoulderOffsetSet	const_ShoulderOffset CurveUp	Kurve, die die Fussposition mit der Aufwärtsbewegung der Schulter verlinkt (Schulter vorne links)	COGController.ChestController_Holder. ChestController.ChestShoulderLeft Controller_Holder.ChestShoulderLeft Controller.ChestShoulderLeft_Null. ChestShoulderLeft_PointAt
const_ShoulderOffsetSet	const_ShoulderOffset CurveForward	Kurve, die die Fussposition mit der Forwärtsbewegung der Schulter verlinkt (Schulter vorne links)	COGController.ChestController_Holder. ChestController.ChestShoulderLeft Controller_Holder.ChestShoulder LeftController.ChestShoulderLeft_Null. ChestShoulderLeft_PointAt
const_ShoulderOffsetSet	const_ShoulderOffset CurveSide	Kurve, die die Fussposition mit der Seitbewegung der Schulter verlinkt (Schulter vorne rechts)	COGController.ChestController_Holder. ChestController.ChestShoulderRight Controller_Holder.ChestShoulderRight Controller.ChestShoulderRight_Null. ChestShoulderRight_PointAt
const_ShoulderOffsetSet	const_ShoulderOffset CurveUp	Kurve, die die Fussposition mit der Aufwärtsbewegung der Schulter verlinkt (Schulter vorne rechts)	COGController.ChestController_Holder. ChestController.ChestShoulderRight Controller_Holder.ChestShoulderRight Controller.ChestShoulderRight_Null. ChestShoulderRight_PointAt

Name des Parametersets	Name des Parameters	Beschreibung	Pfad
const_ShoulderOffsetSet	const_ShoulderOffset CurveForward	Kurve, die die Fussposition mit der Forwärtsbewegung der Schulter verlinkt (Schulter vorne rechts)	COGController.ChestController_Holder. ChestController.ChestShoulderRight Controller_Holder.ChestShoulderRight Controller.ChestShoulderRight_Null. ChestShoulderRight_PointAt
const_ShoulderOffsetSet	const_ShoulderOffset CurveSide	Kurve, die die Fussposition mit der Seitbewegung der Schulter verlinkt (Schulter hinten links)	COGController.HipController_Holder. HipController.HipShoulderLeft Controller_Holder.HipShoulderLeft Controller.HipShoulderLeftController_ Null.HipShoulderLeftController_PointAt
const_ShoulderOffsetSet	const_ShoulderOffset CurveUp	Kurve, die die Fussposition mit der Aufwärtsbewegung der Schulter verlinkt (Schulter hinten links)	COGController.HipController_Holder. HipController.HipShoulderLeft Controller_Holder.HipShoulderLeft Controller.HipShoulderLeftController_ Null.HipShoulderLeftController_PointAt
const_ShoulderOffsetSet	const_ShoulderOffset CurveForward	Kurve, die die Fussposition mit der Forwärtsbewegung der Schulter verlinkt (Schulter hinten links)	COGController.HipController_Holder. HipController.HipShoulderLeft Controller_Holder.HipShoulderLeft Controller.HipShoulderLeftController_ Null.HipShoulderLeftController_PointAt
const_ShoulderOffsetSet	const_ShoulderOffset CurveSide	Kurve, die die Fussposition mit der Seitbewegung der Schulter verlinkt (Schulter hinten rechts)	COGController.HipController_Holder. HipController.HipShoulderRight Controller_Holder.HipShoulderRight Controller.HipShoulderRightController_ Null.HipShoulderRightController_PointAt
const_ShoulderOffsetSet	const_ShoulderOffset CurveUp	Kurve, die die Fussposition mit der Aufwärtsbewegung der Schulter verlinkt (Schulter hinten rechts)	COGController.HipController_Holder. HipController.HipShoulderRight Controller_Holder.HipShoulderRight Controller.HipShoulderRight Controller_Null.HipShoulderRight Controller_PointAt
const_ShoulderOffsetSet	const_ShoulderOffset CurveForward	Kurve, die die Fussposition mit der Forwärtsbewegung der Schulter verlinkt (Schulter hinten rechts)	COGController.HipController_Holder.Hip Controller .HipShoulderRightController_Holder.Righ tShoulderLeftController. HipShoulderRightController_Null.HipSho ulderRightController_PointAt

Tabelle C.3: Die verwendeten internen Variablen

Name des Parametersets	Name des Parameters	Beschreibung	Pfad
int_MouthSet	Int_MouthOpening	Mundöffnung, gesteuert durch JawController	HelperHierarchy.Head_Anchor. Root_LowerJaw
internal_RipBreathSet	internal_RipBreath	effektiver Brustkorbzustand	HelperHierarchy.RipHierarchy
var_TailSet	Int_TailBend	Natürliche Drehung des Schwanzes	COGController.HipController_Holder. HipController.HipController_Null. Hip_Anchor.TailHierarchy
var_TailSet	Int_TailRoll	Schwanzdrehung in sich selbst	COGController.HipController_Holder. HipController.HipController_Null. Hip_Anchor.TailHierarchy
internal_ChestPositionSet	internal_ChestSideOffset	Zwischenspeicher für Bewegung der Hüfte	COGController.ChestController_Holder. ChestController.ChestController_Null. Chest_Anchor
internal_ChestPositionSet	internal_ChestUpOffset	Zwischenspeicher für Bewegung der Hüfte	COGController.ChestController_Holder. ChestController.ChestController_Null. Chest_Anchor
internal_ChestPositionSet	internal_ChestForward Offset	Zwischenspeicher für Bewegung der Hüfte	COGController.ChestController_Holder. ChestController.ChestController_Null. Chest_Anchor
internal_HipPositionSet	internal_HipSideOffset	Zwischenspeicher für Bewegung der Hüfte	COGController.HipController_Holder.Hip Controller.HipController_Null.Hip_Anchor
internal_HipPositionSet	internal_HipUpOffset	Zwischenspeicher für Bewegung der Hüfte	COGController.HipController_Holder.Hip Controller.HipController_Null.Hip_Anchor
internal_HipPositionSet	internal_HipForwardOffset	Zwischenspeicher für Bewegung der Hüfte	COGController.HipController_Holder.Hip Controller.HipController_Null. Hip_Anchor
internal_Shoulder AutomatationsSet	internal_LegDistanceSide	Zwischenspeicher für Bewegung des Fußes (vorne links)	COGController.ChestController_Holder. ChestController.ChestShoulderLeft Controller_Holder.ChestShoulderLeft Controller.ChestShoulderLeft_Null. ChestShoulderLeft_PointAt
internal_Shoulder AutomatationsSet	internal_LegDistanceUp	Zwischenspeicher für Bewegung des Fußes (vorne links)	COGController.ChestController_Holder. ChestController.ChestShoulderLeft Controller_Holder.ChestShoulderLeft Controller.ChestShoulderLeft_Null. ChestShoulderLeft_PointAt
internal_Shoulder AutomatationsSet	internal_LegDistance Forward	Zwischenspeicher für Bewegung des Fußes (vorne links)	COGController.ChestController_Holder. ChestController.ChestShoulderLeft Controller_Holder.ChestShoulderLeft Controller.ChestShoulderLeft_Null. ChestShoulderLeft_PointAt
internal_Shoulder AutomatationsSet	internal_LegDistanceSide	Zwischenspeicher für Bewegung des Fußes (vorne rechts)	COGController.ChestController_Holder. ChestController.ChestShoulderRight Controller_Holder.ChestShoulderRight Controller.ChestShoulderRight_Null. ChestShoulderRight_PointAt
internal_Shoulder AutomatationsSet	internal_LegDistanceUp	Zwischenspeicher für Bewegung des Fußes (vorne rechts)	COGController.ChestController_Holder. ChestController.ChestShoulderRight Controller_Holder.ChestShoulderRight Controller.ChestShoulderRight_Null. ChestShoulderRight_PointAt
internal_Shoulder AutomatationsSet	internal_LegDistance Forward	Zwischenspeicher für Bewegung des Fußes (vorne rechts)	COGController.ChestController_Holder. ChestController.ChestShoulderRight Controller_Holder.ChestShoulderRight

Name des Parametersets	Name des Parameters	Beschreibung	Pfad
		(vorne rechts)	Controller_Holder.ChestShoulderRight Controller.ChestShoulderRight_Null. ChestShoulderRight_PointAt
internal_Shoulder AutomatationsSet	internal_LegDistanceSide	Zwischenspeicher für Bewegung des Fußes (hinten links)	COGController.HipController_Holder. HipController.HipShoulderLeft Controller_Holder.HipShoulderLeft Controller.HipShoulderLeft Controller_Null.HipShoulderLeft Controller_PointAt
internal_Shoulder AutomatationsSet	internal_LegDistanceUp	Zwischenspeicher für Bewegung des Fußes (hinten links)	COGController.HipController_Holder. HipController.HipShoulderLeft Controller_Holder.HipShoulderLeft Controller.HipShoulderLeftController_ Null.HipShoulderLeftController_PointAt
internal_Shoulder AutomatationsSet	internal_LegDistance Forward	Zwischenspeicher für Bewegung des Fußes (hinten links)	COGController.HipController_Holder. HipController.HipShoulderLeft Controller_Holder.HipShoulderLeft Controller.HipShoulderLeftController_ Null.HipShoulderLeftController_PointAt
internal_Shoulder AutomatationsSet	internal_LegDistanceSide	Zwischenspeicher für Bewegung des Fußes (hinten rechts)	COGController.HipController_Holder. HipController.HipShoulderRight Controller_Holder.HipShoulderRight Controller.HipShoulderRightController_ Null.HipShoulderRightController_PointAt
internal_Shoulder AutomatationsSet	internal_LegDistanceUp	Zwischenspeicher für Bewegung des Fußes (hinten rechts)	COGController.HipController_Holder. HipController.HipShoulderRight Controller_Holder.HipShoulderRight Controller.HipShoulderRightController_ Null.HipShoulderRightController_PointAt
internal_Shoulder AutomatationsSet	internal_LegDistance Forward	Zwischenspeicher für Bewegung des Fußes (hinten rechts)	COGController.HipController_Holder. HipController.HipShoulderRight Controller_Holder.HipShoulderRight Controller.HipShoulderRightController_ Null.HipShoulderRightController_PointAt
internal_BackFeetMeasureSet	internal_FeetAverage Forward	Misst die Vorwärtsposition der Füße relativ zur Hüfte	HelperHierarchy.FeetHierarchy. FeetMeasureHierarchy. BackFeetMeasure_Anchor
internal_BackFeetMeasureSet	internal_FeetAverageUp	Misst die Höhe der Füße relativ zueinander	HelperHierarchy.FeetHierarchy. FeetMeasureHierarchy. BackFeetMeasure_Anchor
internal_BackFeetMeasureSet	internal_FeetAverageSide	Misst die Seitwärtsposition der Füße relativ zur Hüfte	HelperHierarchy.FeetHierarchy. FeetMeasureHierarchy. BackFeetMeasure_Anchor
internal_BackFeetMeasureSet	internal_FeetAverageUp2	Misst die Höhe der Füße relativ zur Hüfte und zur Standardpostion	HelperHierarchy.FeetHierarchy. FeetMeasureHierarchy. BackFeetMeasure_Anchor
internal_FrontFeetMeasureSet	internal_FeetAverage Forward	Misst die Vorwärtsposition der Füße relativ zur Hüfte	HelperHierarchy.FeetHierarchy. FeetMeasureHierarchy. FrontFeetMeasure_Anchor
internal_FrontFeetMeasureSet	internal_FeetAverageUp	Misst die Höhe der Füße relativ zueinander	HelperHierarchy.FeetHierarchy. FeetMeasureHierarchy. FrontFeetMeasure_Anchor
internal_FrontFeetMeasureSet	internal_FeetAverageSide	Misst die Seitwärtsposition der Füße relativ zur Hüfte	HelperHierarchy.FeetHierarchy. FeetMeasureHierarchy. FrontFeetMeasure_Anchor
internal_FrontFeetMeasureSet	internal_FeetAverageUp2	Misst die Höhe der Füße relativ zur Hüfte und zur Standardpostion	HelperHierarchy.FeetHierarchy. FeetMeasureHierarchy. FrontFeetMeasure_Anchor

Anhang D: Die Schematic View

Die am besten geeignete Möglichkeit, die teilweise sehr umfangreichen und unübersichtlichen Konstruktionen zwischen Elementen, bestehend aus Parent-Child-Beziehungen, Constraints oder etwa Expressions, zu visualisieren ist in Softimage|XSI die sogenannte Schematic View. Diese ist eine hierarchische Darstellung aller oder einer Teilmenge von Objekten einer Szene und ihrer Beziehungen zueinander. Für die Darstellung der Objekt-Beziehungen im Rahmen dieser Arbeit wurde eine leicht unterschiedliche Art der Darstellung gewählt, die nur einen Teil der Elemente der Schematic View von XSI verwendet.

Objekte werden in der Schematic View durch Vierecke dargestellt, im Normalfall Rechtecke. Geklonte Objekte werden durch trapezförmige Vierecke dargestellt. In den Vierecken ist der Name des jeweiligen Objektes angegeben, der Typ, beispielsweise Polygon-Objekt oder Null-Objekt, ist als Icon links von dem Kasten dargestellt (Tabelle D.1). Links über dem Kasten gibt gegebenenfalls ein Kürzel weitere Informationen über das Objekt, etwa ob es durch ein Constraint beeinflusst wird oder ob es mit einer Expression versehen ist (Tabelle D.2).

Icon	Bedeutung
	Null-Objekt
	Kamera-Objekt
	Polygon-Objekt
	NURBS-Surface-Objekt
	Kurve, Spline
	Punkt-Lichtquelle
	Ursprung oder Effektor einer kinematischen Kette
	Bone einer kinematischen Kette

Tabelle D.1: Auswahl von Icons die den Typ des Elements angeben

Kürzel	Bedeutung
C	Das Element wird durch ein Constraint von einem anderen Element beeinflusst
E	Das Element ist mit einer Expression versehen
L	Das Element besitzt verlinkte Parameter
Cl	Das Element ist Klon eines anderen Elementes
I	Das Objekt ist Instanz eines anderen
D	Dem Objekt ist eine Deformation zugewiesen
V	Das Element ist Teil eines Envelopes
S	Dem Objekt ist eine Shape-Animation zugewiesen

Tabelle D.2: Auswahl von Kürzeln, die Informationen über das entsprechende Element geben

Beziehungen von Elementen zueinander werden durch Linien kenntlich gemacht. Auch hier wurde aus Gründen der Übersichtlichkeit in dieser Arbeit eine leicht unterschiedliche Darstellungsweise verwendet, als sie innerhalb von Softimage|XSI üblich ist. Eine Parent-Child-Beziehung wird etwa durch eine einfache schwarze Linie dargestellt, die vom unteren Ende des Parent-Objektes bis zum oberen Ende des Child-Objektes geht. Constraints werden durch eine grüne Linie mit Pfeil an ihrem Ende verdeutlicht, die vom unteren Ende des Objektes, das beeinflusst wird, bis zum linken Rand des Objektes, das Einfluss hat, geht. Alle Operatoren, die Geometrie erstellen oder verändern, werden durch eine lila Linie mit Pfeil an ihrem Ende dargestellt, die vom unteren Ende des erstellten oder veränderten Objektes bis zum linken Rand des Input-Objektes geht (Abb. D.1, Abb. D.2).

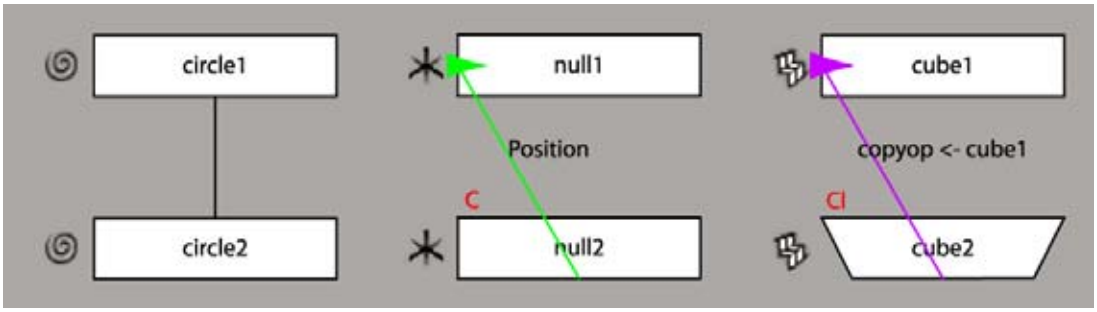

Abb. D.1: Beispiele für Beziehungen zwischen Objekten in der Schematic View. „circle2" ist Child von „circle1", „null2" ist durch ein Positions-Constraint gebunden an „null1" und „cube2" geht aus „cube1" durch eine Kopier-Operation hervor, da es ein Klon ist, ist das umgebende Viereck trapezförmig

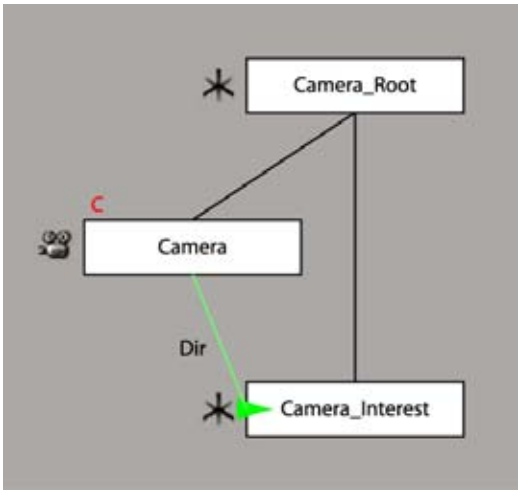

Abb. D.2: Beispiel für eine Schematic View: Der Standardmäßige Aufbau einer Kamera-Hierachie

Anhang E: Inhalt der CD-Rom

Auf der dieser Arbeit beiliegenden CD-Rom finden sich die letzten Versionen der erarbeiteten Ergebnisse, ebenso wie weiteres Dokumentationsmaterial und eine digitalisierte Version der Ausarbeitung.

Das Copyright aller auf dem Datenträger befindlichen Daten liegt, sofern nicht ausdrücklich anders angegeben, bei dem Autor dieser Arbeit.

Auszug aus dem Verzeichnisbaum der CD-Rom:

- **diplomarbeit.pdf:** digitale Version der vorliegenden Arbeit
- **<diplomarbeit>:** XSI-Projektordner
  - **<models>**

    Willy.emdl: in einem XSI-Modell zusammengefasste Elemente des Charakters „Willy“ (Rig und Envelope-Objekte)

    Willy-Envelope.emdl: in einem XSI-Modell zusammengefasste Elemente des entwickelten Rigs

    Dog-Rig.emdl: in einem XSI-Modell zusammengefasste Envelope-Objekte des Charakters „Willy“
  - **<render_pictures>**
    - **<animationen>:** verschiedene bereits mit “Willy” entstandenen Animationen
    - **<stills>:** verschiedene Einzelbilder
  - **<synoptic>:** enthält die erstellten Synoptic Views
  - **<scenes>**

    DogRig.scn: XSI-Szenendatei, die das erstellte Rig enthält

    Willy.scn: XSI-Szenendatei, die die erstellte Geometrie enthält
- **<studien>**
  - **<videos>:** für Animationsstudien aufgenommene Videos
  - **<fotos>:** als Referenzen verwendetet Fotos