"ARM RIGGING FÜR ANIMATOREN" (STRETCHY CHAIN)

RIGGING TUTORIAL
SOFTWARE: SOFTIMAGE XSI
LEVEL: FORTGESCHRITTEN/EXPERTE

von Christoph Schinko, 2006

1. EINFÜHRUNG

Wenn man von einem noch nicht existierenden 3D Charakter träumt, ist es leicht sich auszumalen was er alles können wird und wie leicht es sein wird ihn zu steuern und zu kontrollieren. Während des riggings gehen aber meist viele von diesen ursprünglichen Ideen verloren – sei es wegen zeitlichen Einschränkungen und/oder technischen Schwierigkeiten.

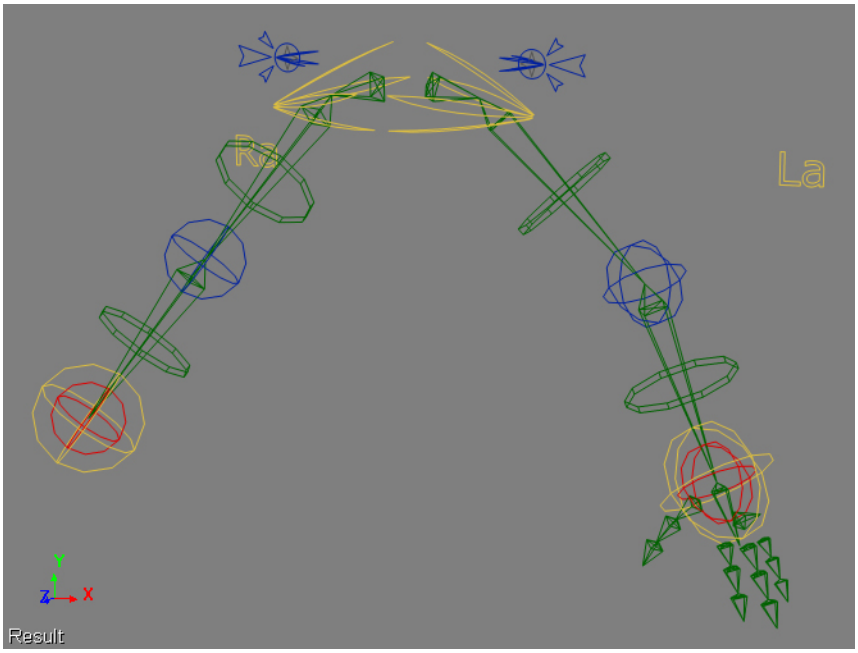


Bild 1: Die finale Setup

Einer dieser Träume von Animators ist das Animieren von Squash & Stretch, einem der Grundstöcke für gute 2D Animation, der aber in 3D Setups meist professionell ignoriert wird. Sicher kann Squash & Stretch übertragen und sinngemäß in 3D Animationen aufgenommer werden, etwa in Bezug auf das Posen und das Verhalten der Gliedmaßen des Charakters, aber schauen wir einmal, ob die untenstehende Wunschliste in eine arm rig integriert werden kann, die schnell und leicht zu steuern ist.

Squash & Stretch
- Die Möglichkeit eine skeleton chain zu überdehnen und dieses Verhalten ein- und auszuschalten
- Die Option eine eventuelle Überdehnung zurück zu setzen, ohne die aktuelle Pose zu verlieren
- Die Möglichkeit, den Ober-, Unterarm oder beide zusammen extra dehnen zu können
- Volumetrisches Skalieren der Armgeometrie und die Möglichkeit das auszuschalten
- Manuelles Animieren der volumetrischen Skalierung

- Die Möglichkeit den Ellbogen in alle Richtungen verschieben zu können
- Umschalten von IK zu FK
- schnelles Fixieren der Hand während des Animierens (zB. Am Tisch anhalten)

Automatische Hilfen zur Gewichtung der Geometrie
- roll divisions für Ober- und Unterarm

2. STRETCH

1. Zeichnen einer 2-bone chain im rechten Viewport und benennen der root „r_arm", der bones „b_upArm" und „b_loArm" und des Effektors „e_arm". Diese Namensgebung hilft beim Folgen dieses Tutorials und beim schnellen Selektieren über XSIs MCP, durch „b_*" etc.

2. Setzen der Länge des Oberarmes (in der ppg des bones) auf „3", und der des Unterarmes auf „4", um mit runden Werten zu arbeiten (und einen klaren Unterschied zu sehen).

Wenn der Effektor bewegt wird, bleibt die Länge der Bones freilich immer die gleiche, bis der Arm ganz durchgestreckt ist und der Effektor nicht weiter von der Root wegbewegt werden kann.
Durch das Ändern der Längen-Werte in den ppgs der bones kann der Arm gedehnt werden – und wieder zur ursprünglichen Länge zurückgesetzt werden.
Diese Veränderungen werden in Abhängigkeit zum Effektor gesetzt, oder besser: der Distanz zwischen Effektor und chain root.

Diese Distanz kann durch eine expression ermittelt, und weiters zur Beeinflußung der Längenparameter verwendet werden. Um einen guten Überblick zu behalten, ist es ratsam ein separates Objekt mit dem Messen der Distanz zu betrauen.

3. Erstellen eines Nulls (mit dem Namen „dist"), um jene Distanz zw. Eff und root zu ermitteln und gleichzeitig anzuzeigen.

4. Setzen einer expression auf die lokale Y Translation des Nulls (dist.kine.local.posy) und „apply" drücken.

ctr_dist(r_arm.kine.global.pos, e_arm.kine.global.pos)

Wenn der Effektor jetzt bewegt wird, wandert das Null entsprechend die Y-Achse auf und ab, zu eben jenem Wert der gemessenen Distanz.

5. Setzen einer Expression auf die Länge des Oberarmes, um ihm mitzuteilen, daß, wenn die Distanz vom eff zur root größer ist als die Summe der ursprünglichen Bonelängen (also, des durchgestreckten Armes), das bone kontinuierlich gedehnt wird (um den eff zu erreichen). Die Summe der ursprünglichen bone Längen (3+4=7) wird hier als durchgestreckte Distanz bezeichnet werden.

Cond(dist.kine.local.posy > 7, 3 + (dist.kine.local.posy – 7) / 2, 3)

Das heißt: Wenn (die Distanz > größer als die durchgestreckte Distanz (3+4) ist, DANN setze die Bonelänge zu ihrem ursprünglichen Wert (3) + die HÄLFTE der Differenz (weil's ja 2 bones sind) zwischen der eigentlichen Distanz (dist) und der durchgestreckten Distanz (dist – (3+4)).

6. Die gleiche Expression wird dem length Parameter des Unterarmes zugewiesen – nur ist dessen Originallänge 4, nicht 3.

*Cond(dist.kine.local.posy > 7, **4** + (dist.kine.local.posy – 7) / 2, **4**)*

Jetzt kann der Eff beliebig weit von der Root gezogen werden, und die bones werden automatisch länger werden – und auch wieder kürzer, bis sie ihre ursprüngliche Länge erreicht haben und sich die chain wieder normal verhält.

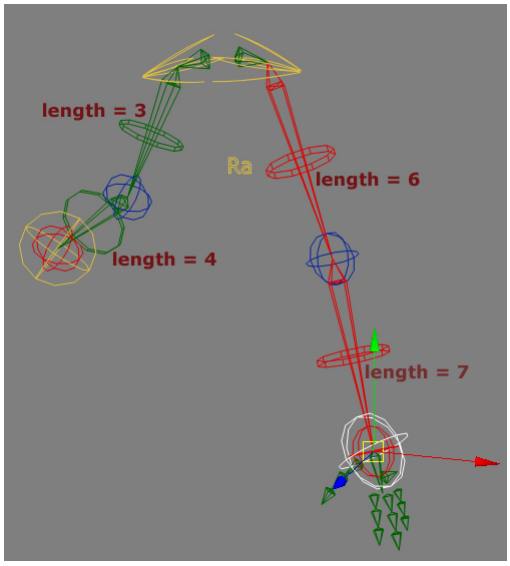


Bild 2: Überdehnen der Arm-Chain

Wenn das getestet wird, vermißt man sofort folgende 3 Dinge:
- die Möglichkeit die chain auf „normal", also undehnbar umzustellen
- die Möglichkeit den Arm auch zu beugen wenn er überdehnt ist
- und die Möglichkeit den Arm zu stauchen, also die bones zu verkürzen

3. RESET & LOCK

Es ist wahrscheinlich meist besser wenn der Arm beim Animieren seine Länge beibehält, und etwaiiges Überdehnen erst nachher per Hand dazu animiert wird. Außerdem wäre es nett einen unabsichtlich überdehnten Arm beim Posen eines Charakters einfach zur ursprünglichen Länge zurücksetzen zu können – ohne die Pose zu verlieren, also entlang der eff/root-Achse.
All das kann durch eine condition in der length expression erzielt werden, indem man eine Veränderung der Längen nur erlaubt wenn der neue Reset&Lock Parameter auf 0 ist. Da beide Bonelängen auf der gemessenen Distanz zwischen eff & root basieren, setzt man am Besten dort an.

7. Erstellen eines neuen Nulls („resetAndLock"), und

8. ändern der Expression am „dist"-Null wie folgt

cond(resetAndLock.kine.local.posy == 0, ctr_dist(r_arm.kine.global.pos, e_arm.kine.global.pos), 7)

Dies wird das dist-Null dazu veranlassen nur dann die beauftragte Distanz zu messen, wenn die Y-Translation des reset&lock Nulls 0 ist. Bei jedem anderen Wert wird die Distanz einfach auf 7 gesetzt, die Gesamtlänge des originalen durchgestreckten Armes – womit man wieder das gleiche Chainverhalten hat wie am Anfang. [Bild 3]

4. STRETCH OFFSET

Neben dem Zurücksetzen und Sperren der Armlänge ware es durchaus hilfreich jades bone einzeln (od. Beide gemeinsam) manuell dehnen zu können – jedoch ohne den Effektor dabei zu verschieben. Solche manuellen offsets können auch dafür verwendet werden, den Arm während einer Überdehnung zu beugen, und man kann damit sogar den Arm zusammenstauchen, also die Längen verkürzen.
Dies wird über 3 Parameter gesteuert, die jeweils zur Dehnung des
a) Oberarmes,
b) beider Bones gleichzeitig, und
c) Unterarmes beisteuern.

9. Erstellen von 3 Objekten für diese Parameter („stretchUp", „stretch" und „stretchLo").

Dieses Mal wird deren Skalierung dazu verwendet, die Bonelängen zu verändern. Die Skalierungswerte werden einfach zu den bereits vorhandenen Längen-Expressions dazu addiert.

10. Ändern der Expression von upArm.bone.length

Vorher:
cond(dist.kine.local.posy > 7, 3 + (dist.kine.local.posy - 7) / 2, 3)

Nachher: *cond(dist.kine.local.posy > 7, 3 + (dist.kine.local.posy - 7) / 2, 3)* ***+ stretch.kine.local.sclx + stretchUp.kine.local.sclx – 2***

„-2" kompensiert die Skalierungswerte der Kontrollobjekte („1" pro Objekt), also eine Skalierung von 1 bedeutet KEINEN Offset (0).

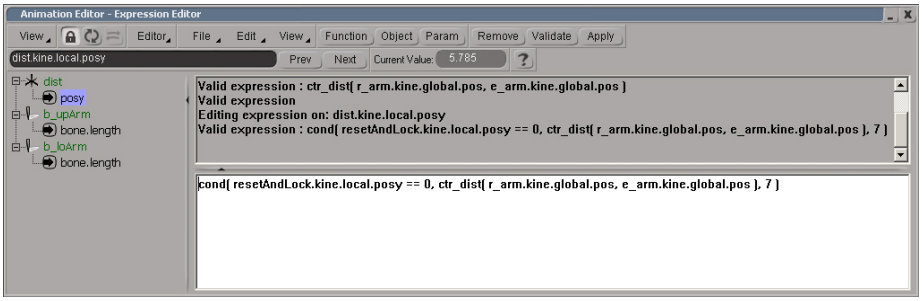
11. Dieselbe Veränderung wird auf den Unterarm übertragen, wieder unter Berücksichtigung, daß dessen ursprüngliche Länge 4 war.

cond(dist.kine.local.posy > 7, 4 + (dist.kine.local.posy - 7) / 2, 4) ***+ stretch.kine.local.sclx + stretchLo.kine.local.sclx - 2***

Jetzt folgt der Effektor natürlich noch den Bones, weil er noch zu keinem Kontrollobjekt constrained wurde.

12. Position constrain den Eff zu einem neuen Kontrollobjekt („c_arm")

Bild 3: Expression Editor



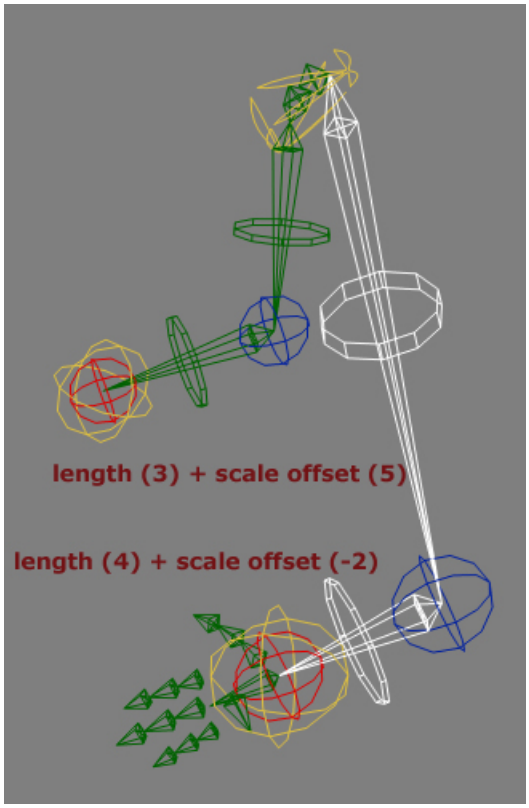


Bild 4: Individuelle Stretch offsets

5. NULL OBJEKTE

Es befinden sich jetzt bereits mehrere Nulls in der Szene, um mit den Expressions zu helfen. Freilich könnten diese Expressions auch verschiendene Parameter ein und desselben Nulls benutzen, oder ein custom parameter set, das die notwendigen Parameter in sich trägt (Vorsicht: XSI mag es nicht, wenn ein customPSet mit gewissen Parametern unter ein Objekt der Chain geparented wird – das könnte einen dependency cycle hervorrufen). Auch könnten alle Expressions über Kontrollobjekte der Charakter Rig gesteuert, und so alle Funktionalität direkt im 3D Viewport zugänglich gemacht werden – was für Animatoren wohl das Wünschenswerteste wäre. Trotzdem ist es empfehlenswert zunächst mit einzelnen Nullobjekten zu arbeiten, da man eine bessere Übersicht behält und leichter die Ursache von Problemen und unerwartetem Verhalten lokalisieren kann.

6. ELLBOGEN OFFSET

Nun jedoch zu etwas ganz anderem. Ein Objekt soll erstellt werden, mit dem sich der Ellbogen des Armes, völlig unabhängig vom Rest der Chain, durch einfaches Translieren verschieben läßt. Die Root und der Effektor bleiben wo sie sind, aber der Ellbogen kann frei an jede Position, in jede Richtung verschoben werden (also etwas ganz anderes als ein upvector constrain). Diese Option ist enorm nützlich zum perfektionieren von Posen, die zB besser aussehen würden wenn der Ellbogen an einer anatomisch inkorrekten Stelle sein würde – sei es auch nur für ein paar frames.
Im legendären Buch von Richard Williams, „The Animator's Survival Kit", spricht er oft darüber ein Gelenk zu „brechen", um eine besser wirkende Animation zu erzielen.
Mit diesem Ellbogen Offset wird das in 3D leicht möglich.

Man verwendet dazu eine zweite Chain, die der Ersten folgt. Eigentlich handelt es sich um zwei 1-bone-chains, nicht um eine 2-bone-chain. Die Funktionsebene dieser simulierten 2-bone chain wird durch das Ellbogen Kontrollobjekt definiert.

13. Erstellen eines Kontrollobjektes für den Ellbogen („c_elbow") und Zuteilen der selben Translation wie b_loArm durch *match translation* unter „Transform" im MCP.

14. Parenting von c_elbow unter b_loArm, damit es immer dem ursprünglichen Ellbogen folgt und von dort aus leicht animiert werden kann.

15. Zeichnen zweier 1-bone-chains, genau entlang der bones der arm chain. Das macht man am Besten mit Snapping (centers). Benennen der chains „offsetUp" (r_offsetUp, b_offsetUp,...) und „offsetLo".

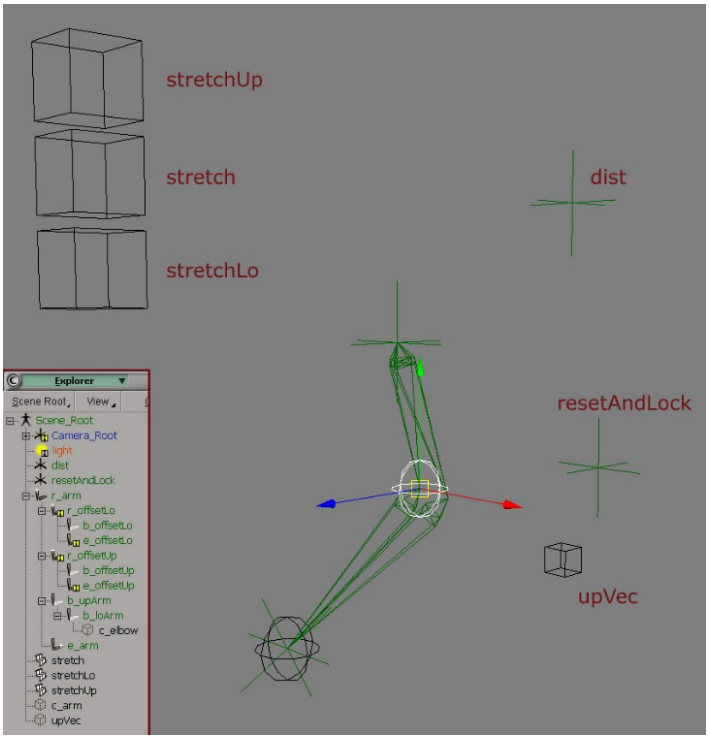
16. Position constrainen der offsetUp chain zur ursprünglichen Root und dem Ellbogen Kontrollobjekt, und der zweiten chain zum Ellbogen Kontrollobjekt und zum ursprünglichen Effektor wie folgt [Bild 5]:

a) r_offsetUp -> r_arm
b) e_offsetUp -> c_elbow
c) r_offsetLo -> c_elbow
d) e_offsetLo -> e_arm

17. Verstecken der neuen roots und Effektoren zur besseren Übersicht.

Und schon funktioniert der Ellbogen Offset wie er soll, und kann mit Ctrl+Shift+R bequem zurückgesetzt werden. Nun müssen die neuen bones nur noch die korrekte Länge zugewiesen bekommen, da sie

Bild 5: Freier Offset für Ellbogen



schließlich später als Deformer der Geometrie dienen werden.
Dazu kann man wie vorher die Distanz zwischen Ellbogen und Root, bzw. Effektor verwenden.

18. Setzen der folgenden Expression

a) b_offsetUp.bone.length:
ctr_dist(r_arm.kine.global.pos, c_elbow.kine.global.pos)

b) b_offsetLo.bone.length:
ctr_dist(e_arm.kine.global.pos, c_elbow.kine.global.pos)

Das ist schon viel besser. Wenn man jedoch jetzt der ursprünglichen arm chain einen upVector zuweist, sieht man das die neuen bones nicht der x-Rotation der alten bones folgen. Das kann man aber für offsetLo leicht ändern.

19. Erstellen eines upVector Objekts („c_upVec"), und zuweisen desselben: b_upArm selektieren, *Create>Skeleton>ChainUpVector* klicken und c_upVec wählen.

Bewegt man nun c_upVec herum, so sieht man das Rotationsproblem der offset chains. Anstatt r_offsetLo zu c_elbow position zu constrainen, nimmt man einfach ein pose constrain.

20. Löschen des vorhandenen position constraints unter r_offsetLo, und Erstellen eines Pose constraints an dessen Stelle (r_offsetLo an c_elbow).

Jetzt stimmen die Rotationen immer überein. Für das offsetUp bone geht das nicht so leicht, aber lassen wir das zunächst beiseite, da sich dieses Problem zu einem späteren Zeitpunkt von alleine lösen wird.
Vor dem nächsten Schritt noch die neuen chains zur alten root parenten, nur der Übersicht wegen.

21. Drag&Drop r_offsetLo und r_offsetUp unter r_arm.

7. VOLUMETRISCHE DEFORMATION

Überdehnt man den Arm jetzt, so würde die Geometrie zwar folgen, aber immer den gleichen Durchmesser beibehalten, also im Volumen zunehmen. Um jedoch das Volumen konstant zu halten, skaliert man y+z der bones indirekt proportional zur Länge.

22. Setzen einer Expression an die Y-Skalierung des Oberarmes (b_offsetUp). Für eine indirekt Proportionale Verbindung zwischen der Länge und der Skalierung, wird die Skalierung 1/ die Länge genommen.

1 / (b_offsetUp.bone.length – 2)

Wenn die chain nicht überdehnt wird, soll die Skalierung des Bones 1 sein, die Formel muß also 1/1 ergeben. Darum wird von der ursprüngliche Bonelänge (3) zwei abgezogen.

23. Die gleiche Expression wird für die Z-Skalierung genommen. Wenn man hier den Y-Skalierungs Parameter einfach auf den Z Parameter zieht, meldet XSI einen dependency cycle. Also wollen wir ihm den Gefallen tun und die Expression eingeben.

Jetzt behält der Oberarm sein Volumen bei Überdehnung. Beim Animieren stellt sich jedoch schnell heraus, daß das nicht immer wünschenswert ist, und es oft besser wäre dieses korrekte Skalieren auszuschalten oder von Hand animieren zu können.

24. Erstellen eines neuen Nulls („volScale"), dessen lokaler posY Parameter dazu dienen wird das volumetrische Skalieren ein und auszuschalten.

25. Ändern der SclY Expression am Oberarm, um vol. Skalierung auszuschalten wenn das volScale null auf y=0 gesetzt wird.

cond(volScale.kine.local.posy == 0, 1, 1 / (b_offsetUp.bone.length - 2))

26. Die X-Translation des selben volScale Nulls wird herangezogen um die volumetrische Skalierung manuell zu verändern.

cond(volScale.kine.local.posy == 0, 1, 1 / (b_offsetUp.bone.length - 2) ***+ volScale.kine.local.posx*** *)*

27. Die selbe Veränderung wird auch auf SclZ angewandt.

28. Und nun die selbe Prozedur für den Unterarm b_offsetLo (SclY + SclZ)

cond(volScale.kine.local.posy == 0, 1, 1 / (b_offsetLo.bone.length - 3) + volScale.kine.local.posx)

8. HANDGELENK SETUP UND KONTROLL-OBJEKTE

Wie würde man nun am Besten die Drehung des Handgelenks in diese Rig integrieren und steuern? Da gibt es 2 unterschiedliche Methoden:
Entweder folgt die Handrotation immer der des Unterarmes, sodaß das Handgelenk immer „gerade" ist, oder die Handrotation ist unabhängig von der Rotation des Unterarmes, und zeigt immer in die gleiche globale Richtung.
Beide Methoden haben ihre Vor- und Nachteile. Beim Posen eines Charakters, oder beim Animieren eines schwingenden Arms, ist es natürlicher wenn die Hand dem Unterarm folgt, und jegliche zusätzliche Animation (follow through) wird von f-curves leicht verständlich dargestellt, da ja die 0,0,0 Rotation immer geradlinig vom Unterarm wegzeigt.
Dieses Verhalten ist jedoch nicht erwünscht, wenn der Arm einmal animiert ist und später zB. Der upVector verändert wird – die gesamte Handanimation würde sich ebenfalls ändern. Noch fataler wäre diese Änderung wenn die Hand mit einem anderen Szenenobjekt interagiert, und sie zB etwas hält.

Es wäre also wünschenswert beide dieser Methoden in einer Rig zu vereinen, und den Animator selbst entscheiden und zur gewünschten Methode umschalten zu lassen.

29. Zeichnen einer 1-bone-chain im rechten viewport („r_wrist", „b_wrist" und „e_wrist").

30. Angleichen der Rotation (match rotation) von r_wrist zum ursprünglichen Arm Effektor e_arm. Die Rotationen des HandBONES auf 0 setzen, sodaß die Z-Rotation der Hand bei normal hängendem Arm die Hand zum und vom Körper weg dreht.

31. Kopieren des c_arm Kontrollobjekts („c_wrist") und Kleinerskalieren desselben. Es wird die Rotation der Hand steuern.

32. Erstellen eines Nulls („wristBuffer"), Angleichen der Translation zu „c_wrist", und Angleichen der Rotation zu „e_arm".

33. Parenten von „c_wrist" unter „wristBuffer", und Rotationen von „c_wrist" auf 0 setzen.

34. Orientation constrain das Handbone „b_wrist" zum Handkontrollobjekt „c_wrist".

35. Position constrain „r_wrist" zu „e_arm"

36. Position constrain „wristBuffer" ebenfalls zu „e_arm"

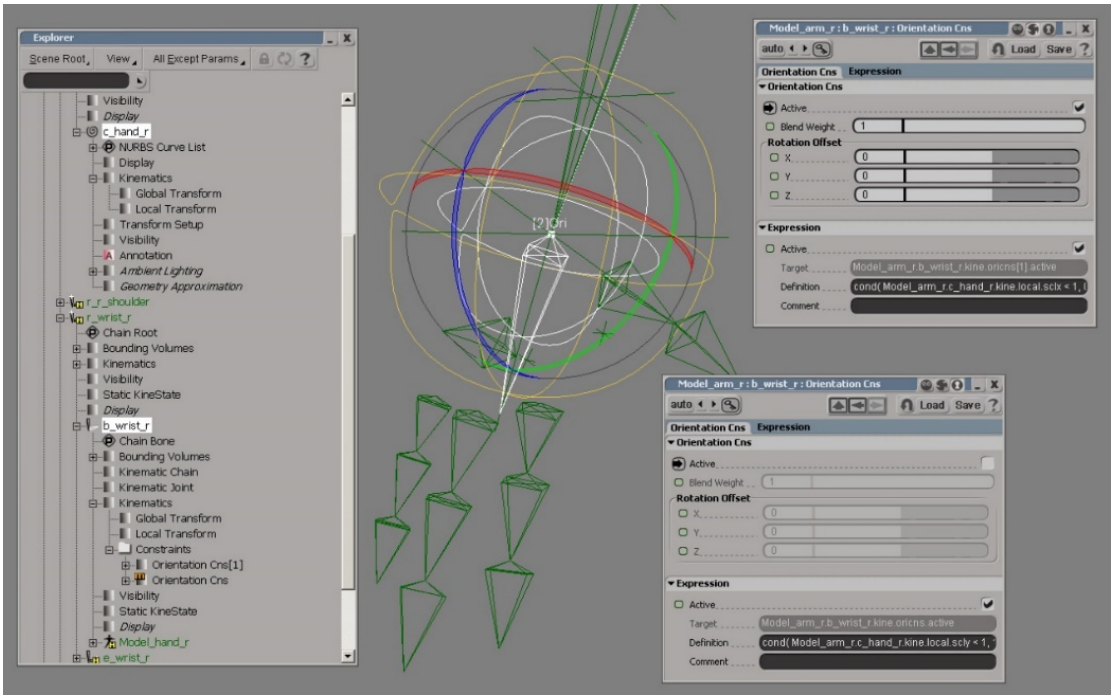


Bild 6: Handgelenk Setup

37. Orientation constrain „wristBuffer“ zu „e_arm“ und locken der ppg des constraints.

Der „active“ Parameter dieses constraints wird dazu benutzt werden zwischen den zwei oben besprochenen Methoden hin- und herzuschalten.

38. Verlinken des active-Parameters mit dem SclX Parameter von „c_wrist“, um direkt im Viewport umschalten zu können.

cond(c_wrist.kine.local.sclx < 1, 0, 1)

Nun ist die Setup dazu bereit getestet zu warden. Man kann den Charakter nun sehr bequem posen und das Folgen der Handrotation leicht umschalten und animieren. Die f-curves nehmen eine gerade ausgestreckte Hand als 0,0,0 wahr, und sind somit klar verständlich und einfach zu editieren.

Bemerkung: Während des Animationsvorganges kann das Handgelenk gimbal lock hervorrufen. Dagegen hilft es bis zu einem gewissen Grad, die Rotation Order des Handgelenk bones umzustellen (YZX scheint ganz gut zu funktionieren), oder man folgt online tutorials die der Umgehung von gimbal lock dienen – wie zum Beispiel die altbewährte Methode die Rotationsachsen auf verschiedene Objekte aufzuteilen.

9. ARM & HAND SPERREN

Was für's Animieren auch immer wichtig ist, ist die Möglichkeit die Hand schnell in einer bestimmten Position zu fixieren, wenn sie sich zum Beispiel an einem Tisch festhält. Das macht man generell indem man den Arm position constrained, und die Hand orientation constrained.

39. Erstellen eines Hilfs-Objekts, zu dem der Arm und die Hand später constrained wird („handSnap“). Das Objekt sollte so geformt sein, daß man sieht in welche Richtung die Hand zeigen wird.

40. Speichern eines Tranlations-keys auf „c_arm“

41. Position constrain „c_arm“ zu „handSnap“

42. Setzen einer Expression auf den „active“ Parameter dieses position constrains

cond(c_arm.kine.local.scly < 1, 1, 0)

Damit schaltet man wieder über die Skalierung des Kontrollobjektes zwischen constrained oder nicht constrained um. Wenn „c_arm“ in Y runterskaliert wird, springt der Arm zu „handSnap“ und bleibt dort fixiert. Für die Rotation der Hand macht man es ganz ähnlich.

43. Orientation constrain „c_wrist“ zu „handSnap“

44. Setzen der gleichen expression auf den „active“ Parameter des Orientation constrains

cond(c_wrist.kine.local.scly < 1, 1, 0)

So kann man die beiden constraints unabhängig von einander aktivieren und animieren, je nachdem wie man's gerade braucht. Da die „blending“ Schieberegler beider constraints noch frei verfügbar sind, können sie natürlich mit animiert werden, und so beide constraints langsam ein- oder ausgeblendet werden.

Nun sollte noch ein praktischer Aufbewahrungsort für das „handSnap“ Objekt gefunden werden – einer wo es beim Animieren nicht im Weg ist, aber im viewport trotzdem schnell auffindbar ist. Dazu bietet sich zum Beispiel die Arm root an, welcher dieses und andere Kontroll Objekte folgen könnten.

10. VISUELLES FEEDBACK

Die Rig beinhaltet bereits viel Funktionalität und verschiedene Optionen, die vielleicht für den Animator verwirrend oder schwer auffindbar sein können. Es ist also durchaus angebracht sich ein bißchen damit zu beschäftigen, detailiere Beschreibungen und wo auch immer möglich visuelles Feedback zur Verfügung zu stellen.

Zunächst könnte man an alle Kontroll-Objekte annotation-properties hängen, die jegliche

Funktionalität des selektierten Objektes beschreiben.

45. Selektieren von „c_arm", dann *Get>Property>Annotation* wählen

46. Hier wäre ein Hilfetext wie „SclY < 1 aktiviert POSITION constraint zum HANDSNAP Objekt" angebracht. Dasselbe nun für alle anderen Kontrollobjekte, die verwirrend oder unklar werden könnten. Die annotation property kann dann jederzeit ganz leicht geöffnet werden, indem man mit dem gewünschten Objekt selektiert „Selection" im MCP klickt, oder im Explorer im Selection-Modus unter dem Objekt nachschaut – man findet gleich das rötliche Annotation icon.

Ein anderer Weg eine Rig leichter Verständlich zu gestalten, wäre das Ändern von Farben der Kontrollobjekte oder der bones, wenn diese zum Beispiel überdehnt sind. Dazu legt man eine Expression auf die „display colour" property eines Objektes, um es dazu zu bringen auf einmal knallrot zu sein, wenn es größer skaliert wird.

47. Selektieren und isolieren von „b_upArm" im Explorer. „Display" unter dem Objekt ist jetzt noch kursiv geschrieben, dh. diese Eigenschaft wird mit dem Rest der Szene geteilt – es muß also zuerst lokal, nur für diese Objekt geltend, gemacht werden.

48. Klicken auf das „Display" icon und beantworten der folgenden Dialog-Box mit „Yes" (make local). Jetzt ist „Display" nicht mehr kursiv.

49. Expandieren der display options im Explorer und setzen einer Expression auf die „color" Property

cond(b_upArm.bone.length > 3, 15, 0)

"15" ist ein alamierendes Rot, indem das bone immer dann angezeigt wird, wenn es länger als seine ursprüngliche Länge (3) ist. Man sieht also direkt im viewport, ob und was überdehnt oder skaliert ist. Das wäre zum Beispiel auch für die stretch offset Kontrollobjekte nützlich, um gleich erkennen zu können ob sie verändert wurden. Besser wäre es sogar diese in rot darzustellen, wenn sie größer ODER kleiner als 1 skaliert wurden („!=" ist das Expression-Kommando für „ungleich" – siehe „*Function>Controls*" im Expression Editor). Vielleicht nimmt man dafür gleich zwei conditions: wechsle zu rot wenn größer als 1.1 oder kleiner als 0.9, dann hat man etwas Platz dazwischen und kann leichter zu 1 zurückfinden, wenn man im viewport und nicht numerisch arbeitet.

50. Für die Ellbogen Offset bones wäre es am Besten nur dann sichtbar zu sein, wenn der Ellbogen vom normalen Arm verschoben wurde. Als legt man eine Expression auf deren view visibility,

cond(c_offset_l_arm.kine.local.posx == 0 && c_offset_l_arm.kine.local.posy == 0 && c_offset_l_arm.kine.local.posz == 0, 0, 1)

und vielleicht färbt man sie gelb („126") ein, damit man sie immer unterscheiden kann (genaue Farbwerte kann man der history des Script-Editors entnehmen).

Schlußendlich, wenn die Rig so funktioniert wie man das gerne hätte, geht man daran die vielen Helfer-Nulls mit visuell sinnvollen Objekten zu ersetzen, und so die Rig endgültig userfreundlich zu gestalten. Es ist zwar eine Herumfitzelei, aber sobald man zum Animieren anfängt erkennt man: es zahlt sich aus!

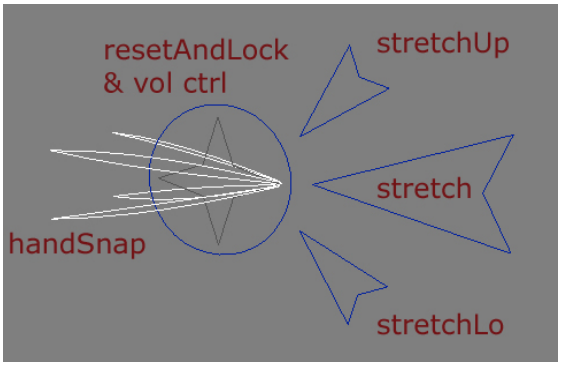


Bild 7: Icons für Chain Optionen

11. FK/IK (forward kinematics / inverse kinematics)

Eine weitere wichtige Entscheidung speziell beim Animieren von Armen ist die Wahl von FK oder IK. Diese Rig zielt offensichtlich auf einen IK Arm ab, aber mit Softimage XSIs exzellenter Implementation von FK/IK blending, wäre es zu schade sich das nicht zunutze zu machen.

XSIs FK/IK Kontrollen findet man in den „kinematic chain properties" unter dem ERSTEN bone jeder chain. Der „FK/IK blend" Schieberegler erlaubt effektives überblenden zwischen den zwei Animationsmethoden.

Das einzige was unsere Rig daran hindert in FK animiert zu werden ist, daß der Effektor zu c_arm position constrained ist. Wenn man aber nun die Aktivität dieses constraints mit einer Expression ansteuert, und zum Beispiel dann de-aktiviert, wenn das c_arm Kontrollobjekt runterskaliert wird, ist man frei, in FK zu animieren, und könnte mit dem gleichen Prozeß der Skalierung auch XSI mitteilen, daß es sich jetzt um eine FK chain handelt.

51. Setzen einer Expression auf den blending Parameter des position constraints unter „e_arm"

c_arm.kine.local.sclx

52. Die gleiche Expression wird nun auf die „FK/IK blend" property in der kinematic chain ppg gelegt.

Skaliert man nun das c_arm Kontrollobjekt in X kleiner, so blendet XSI von IK zu FK, und gleichzeitig wird das position constrain des Effektors zum Kontrollobjekt schwächer, bis – bei einer Skalierung von x=0 – die chain eine echte FK chain ist.

Durch animieren dieser Skalierung kann man also FK/IK blending animieren!

Das Skalieren des Kontrollobjektes gegen 0 visualisiert gleichzeitig die schwindende Influenz des Objektes. Zusätzlich könnte man im FK Modus der chain die shadow icons der bones aktivieren (ebenfalls mit expressions), um sie im viewport leicht selektieren zu können.

12. WEIGHTING HILFSOBJEKTE

Zum Schluß noch einige Überlegungen zum späteren Gewichten der Geometrie. Normalerweise ergibt sich durch das Rollen der Hand in X im Unterarmbereich häßliche Deformationen, da der enveloping Algorithmus rotationen innerhalb eines Deformers nicht interpoliert. Das kann man aber mit Hilfe weiter Deformer Objekte in den Griff bekommen. Freilich gibt es zahlreiche verschiedene Methoden dazu: ein zweiter Unterarmknochen,

gleich oder gegengleich dem Ersten, genau durchdachte, anatomisch korrekte Lösungen für Elle und Speiche, etc etc.
Hier werden wir einfach einige Null Objekte entlang des Unterarmes positionieren, und die Handrotationübertragung zum Ellbogen hin laufend abschwächen.

Das gleiche Problem hat man natürlich auch mit dem Oberarm und dessen Verhalten wenn er in X rotiert wird. Der obere Teil nahe der Schulter bleibt quasi stehen, während sich der untere Teil beim Ellbogen komplett mit dem Oberarmknochen mitrotiert. Auch hier kann der Trick mit den Null Objekten sehr hilfreich sein.

Diese Null Objekte nennt man normalerweise „roll divisions", und sie werden in den in XSI implementierten rigs auf Wunsch automatisch eingefügt, und funktionieren hervorragend über scripted operators. Generell sind die XSI rigs gut durchdacht und sehr funktionell, sie sind also unbedingt eine Überlegung wert.
Hier werden wir diese setup aber selbst aufbauen, sei es nur um unser Verständnis zu vertiefen und dem Rigging-Nirvana einen Schritt näher zu kommen.

DER UNTERARM

53. Erstellen eines Nulls („h_loArmRoll") und parenten desselben unter „b_offsetLo", jenem offset bone, daß später geenveloped wird.

54. 2-point-constrainen des Nulls zu „c_elbow" und „e_arm". Die ppg des constraints offen lassen.

Der „distance percentage" Parameter wird die Verteilung der Nulls entlang des Unterarms bestimmen.

55. AUSSCHALTEN des active flags auf der „upVector" tab der ppg (sonst funktioniert die kommende Expression nicht)

56. Setzen der folgenden Expression auf die X-Rotation des Helfernulls

b_wrist.kine.local.rotx / 2

Von jetzt an wird die X-Rotation von „h_loArmRoll" die Hälfte der X-Rotation der Hand sein.

57. „h_loArmRoll" dreimal kopieren

58. Verteilen der kopierten Nulls mit Hilfe des „distance percentage" Parameters des 2-point-constraints. 25%, 50%, 75% und 100% (um eines direkt am Handgelenk zu haben).

Recht schnell geht das, wenn man die ppg des constraints auf „update similar" setzt (das 2-Augen icon neben dem lock icon in der Menüleiste), die Prozentzahl einstellt und dann das nächste Null selektiert – die ppg wird automatisch das Constraint des neuen Nulls reflektieren.

59. Selektieren von „h_loArmRoll" und öffnen des Expression Editors (shortcut „0"). Die Expression jedes Nulls wird so eingestellt, daß diese immer weniger der Handgelenksrotation vererbt bekommen. Das Null am Handgelenk zum Beispiel „/2" (wenn der Charakter ein Hemd trägt, daß sich nicht 100%ig mit der Hand mitdrehen muß), das nächste Null in Richtung Ellbogen „/4", dann „/8" und „/16" der Handgelenksrotation.

b_wrist.kine.local.rotx / 16

Rotiert man nun die Hand in X, so zeigen die Nullobjekte bereits wie gleichmäßig diese Rotation später auf die Geometrie übertragen wird.

60. Um bei einem „Undo" oder dem zurücksetzen (Reset rotation) der Handrotation unerwartete Rotationen der Nullobjekte vorzubeugen, setzt man am Besten die Y und Z Rotation eines jeden per Expression auf „0".

61. Da die Geometrie ja später zu diesen Hilfsobjekten geenveloped wird, müssen diese auch das vorher überlegte volumetrische Skalieren korrekt weitergeben. Hier reicht es allerdings, die richtige Expression auf die Y und Z-Skalierung EINES Nulls zu legen, und die anderen dann einfach diesen Wert übernehmen zu lassen (per drag&drop im Explorer).

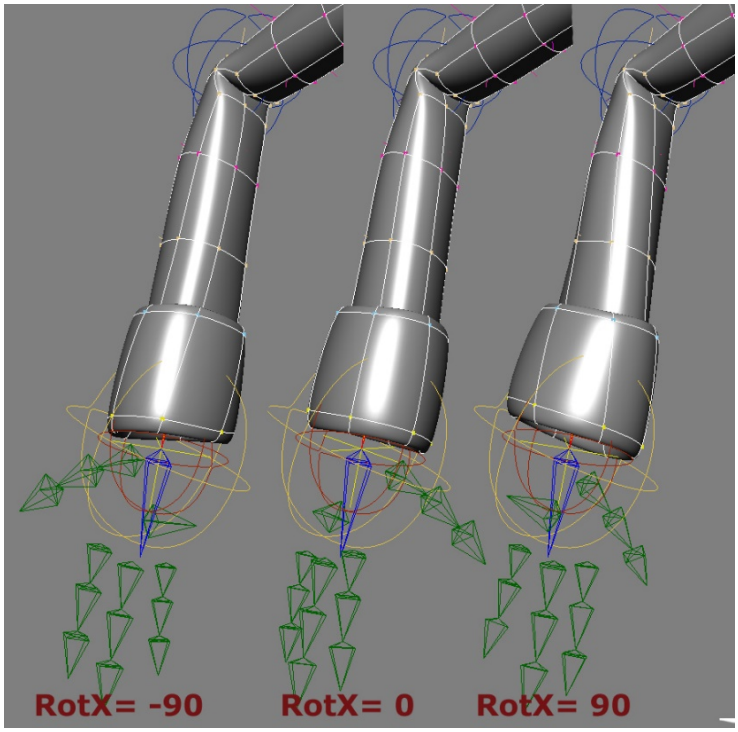


Bild 8: Unterteilung der Unterarm Rotationen

DER OBERARM

Die Hilfsobjekte für den Oberarm baut man ganz ähnlich auf.

62. Erstellen eines Nulls und 2-point-constraints zu „r_arm" und „c_elbow"

63. AUSSCHALTEN von „tangent" UND „upVector" in den entsprechenden ppg tabs des Constraints.

64. Ein direction constraint in Richtung „c_elbow" setzen.

Wenn man bei Schritt 54. zuerst „c_elbow" selektiert hat, so muß das direction constraint auf „r_arm" gehen. Bei Unsicherheit „relation" im Auge-icon des viewports einschalten – die Constraints werden dann angezeigt.

65. Jetzt setzt man die X-Rotation des Nulls per Expression auf die Hälfte der des ursprünglichen Oberarmknochens („b_upArm", nicht das Ellbogen Offset bone).

66. Die übrigen Rotationsachsen werden wie vorher per Expression auf 0 gesetzt.

67. Kopieren des Hilfsnulls 4 weitere Male, verteilen dieser neuen Nulls über die distance percentage des 2-point-constraints, und einstellen der RotX-Expressions wie vorher für den Unterarm.

Diese „h_upArmRoll“ Objekte müssen nicht unter „b_upArmOffset“ geparented werden, und im Grunde hat „b_upArmOffset“ ganz und gar keine Funktion mehr in dieser Setup, weil jegliche Gewichtung sich an die Hilfsnull halten wird. Es kann also getrost gelöscht werden, oder für visuelles feedback auch behalten werden.
Jetzt wird es auch verständlich warum es in Schritt 20 nicht notwenig war, das erwähnte Rotationsproblem mit dem upVector zu lösen.

ELLBOGEN HILFSOBJEKTE

Nun könnte man noch ein weiters Null and den Ellbogen hängen, und ihm die halbe Z-Rotation des Unterarmes zuweisen. Damit wird das Gewichten des Ellbogen leichter, und er bleibt eher, wo er sein sollte. Natürlich kann es auch mit der Rotation des Unterarmes gelinkt werden, und eventuell gewünschte Translation über relative values gesteuert werden, oder man bedient sich für das Perfektionieren der Ellbogen-Geometrie der Shape deform keys...

13. SPIEGELN DER ARM RIG

Beim Spiegeln der Arm rig auf die andere Seite des Charakters darf „freeze negative scaling“ in der *Skeleton>DuplicateSymmetry* Dialog Box NICHT eingeschaltet sein. Die Constraints sollen schon dupliziert werden, jedoch kann das eine oder andere Kontrollobjekt in irgendeine merkwürdige Position springen. Das kann jedoch durch zurücksetzen der Translationswerte auf 0 korrigiert werden.
Das setzt natürlich voraus, daß die Kontrolobjekte zum Beispiel durch *Create>Skeleton>SetNeutralPose* in deren default Pose auf 0 gesetzt wurden.
Nach dem spiegeln sollte man auch die expressions für die roll divisions überprüfen und gegebenenfalls auf „ - wrist rotation / 2“ etc umstellen.

14. SCRIPTING

Es ist natürlich ein ziemlicher Aufwand eine chain in diese rig zu verwandeln. Wenn man das für zB. beide Arme und beide Beine eines Charakters macht, wird es recht zeitintensiv – vor allem, wenn später das Skeleton noch verschoben oder sonst wie angepaßt wird, dann müssen alle Expressions nach-justiert werden.
Um das zu vermeiden, sollte man diese Prozedur erst dann in Angriff nehmen, wenn man sicher ist das alles am rechten Fleck sitzt, oder den rigging Prozeß, zumindest teilweise, in Form eines Skriptes automatisieren.

15. ZUSAMMENFASSUNG

Wir haben jetzt eine recht fortgeschrittene Arm rig vor uns, die speziell für Animatoren entwickelt wurde und die meisten ihrer Wünsche erfüllen sollte. Freilich bevorzugen verschiedene Animatoren verschiedene Rigs und Methoden, aber hoffentlich hat Ihnen dieses Tutorial genug Information und Ideen gegeben, um Ihren eigenen speziellen Weg einen Arm zu Bewegen ins Leben rufen können.

Written by Christoph Schinko in 2006
office@christoph-schinko.com
www.christoph-schinko.com