

Einführung in die Spieleprogrammierung

Animation



- Animation \neq Bewegung
- Anima (lat.): Atem, Seele, Leben
- \rightarrow „Bewegen“
- Animation:
Erzeugung
synthetischer
Bewegtbilder

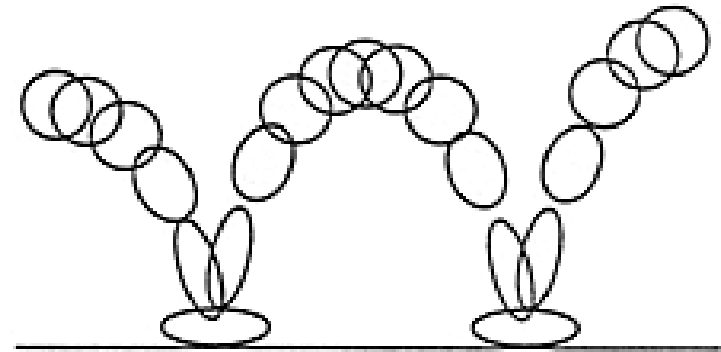


Toy Story (1995): Erster vollständig
am Computer erstellter Kinofilm

- Animation bezeichnet nicht nur Bewegungen, sondern alle visuellen Änderungen:
 - Position
 - Form
 - Farbe/Textur/Transparenz/Struktur
 - Beleuchtung
 - Kameraeinstellung
 - Post-Processing-Effekte

- **Squash and Stretch**

Verformung bei Beschleunigung und Kollision je nach Masse und Steifheit (Volumen sollte gleich bleiben), aber z. B. auch bei Gesichtsanimationen

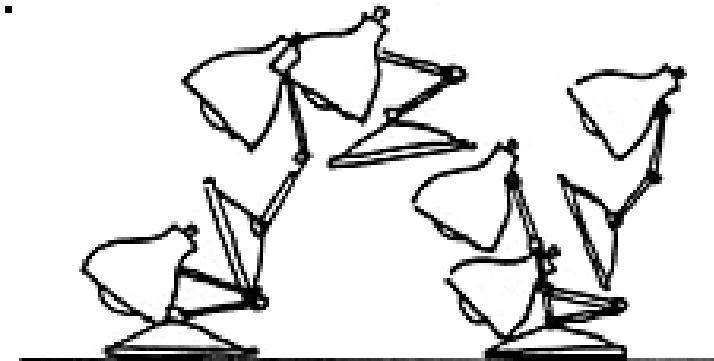


- **Anticipation**

Jede wichtige Bewegung sollte für den Zuschauer vorhersehbar sein, also entsprechend eingeleitet werden.

- **Staging**

Art der Präsentation einer Idee so, dass sie vollständig verstanden bzw. erkannt wird. Es sollte nur eine Hauptidee gleichzeitig dargestellt werden. Hervorhebung durch klare Unterscheidung von der Umgebung.



- **Straight Ahead Action and Pose to Pose**
Gegensätzliche Prinzipien zur Bewegungserstellung: Bild für Bild vs. Pose für Pose (= Zunächst Start und Ende vorgeben und dann erst die Zwischenzustände einfügen)
- **Follow Through and Overlapping Action**
„Ausklingen“ einer Aktion und Herstellung eines Übergangs zur nächsten Aktion.
- **Slow In and Slow Out**
Unterschiedlicher Abstand der Zwischenbilder → Unterschiedliche Geschwindigkeiten.
Geschwindigkeitsänderung sollten nicht abrupt, sondern über Beschleunigungen erfolgen.
- **Arcs („Bogen“)**
Gebogene Pfade für natürliche Bewegungen.

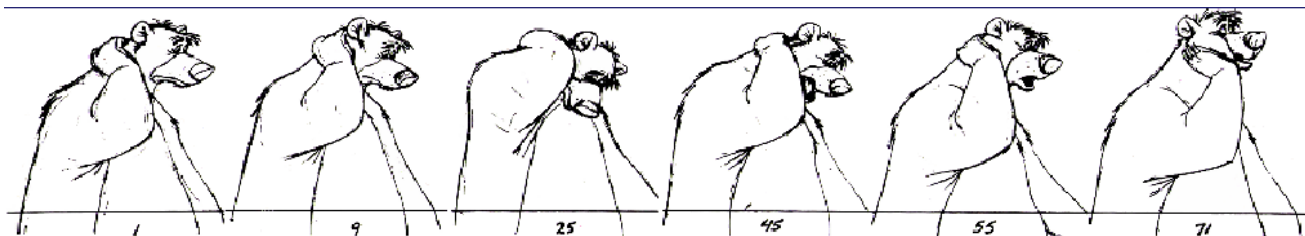
- **Exaggeration**
Betonung durch Übertreibung.
- **Secondary Action**
Aktion die direkt aus einer anderen folgt, aber diese niemals dominiert.
- **Timing**
Geschwindigkeit stellt Größe und Gewicht eines Objekts dar oder auch eine Stimmung.
Geschwindigkeit so einstellen, dass der Beobachter folgen kann und sich nicht langweilt.
- **Solid Drawing***
Animator muss die Grundsätze des Zeichnens verstehen (z.B. Anatomie, räumliche Darstellung, Beleuchtung...), aber auch die mögliche zeitliche Veränderung einer Szene.
- **Appeal**
Design oder Aktion soll dem Betrachter gefallen

*Von Lasseter weg gelassen, gehört aber zu den traditionellen Animationsprinzipien

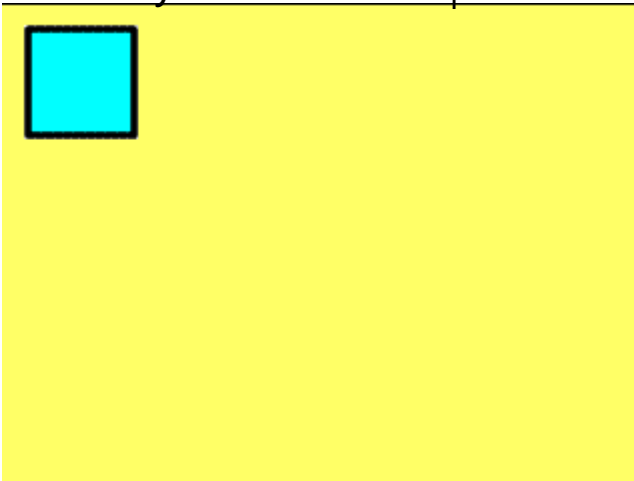
Quelle: J. Lasseter: Principles of Traditional Animation Applied to 3D Computer Graphics, SIGGRAPH 87
F. Thomas and O. Johnston: The Illusion of Life – Disney Animation, Walt Disney, 1981

- Für Trickfilme können alle Animationen vorab berechnet werden.
- In Computerspielen müssen Animationen zur Laufzeit ausgeführt werden.
 - Echtzeitanforderungen
 - Anpassung an aktuelle Spielsituation
- Finden eines guten Kompromisses zwischen flexiblen Animationen und vorausberechneten Animationen.
- Animation sollte mit mindestens 12 Bildern pro Sekunde (FPS) abspielen
- Bei Spielen oft höhere Anforderung (je nach Genre), für Shooter oft:
<20 = unspielbar; 20-35 bedingt spielbar; >35 flüssig spielbar
- Mehr als ~70 FPS kann das Auge nicht mehr wahrnehmen

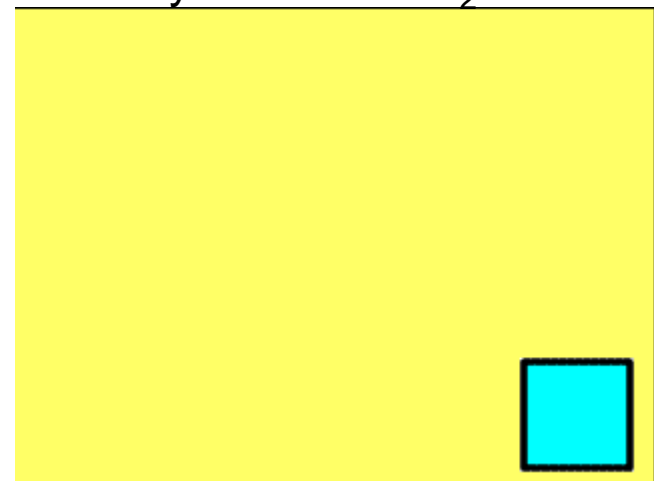
- Technik aus dem Trickfilmbereich
- Prinzip:
 - Gezeichnet werden nur die wichtigsten Bilder der Animation, die sog. Keyframes
 - Zwischenframes werden mittels Interpolation berechnet
- Vorteile:
 - Weniger Aufwand bei der Animationserstellung, da weniger Bilder gezeichnet werden müssen
- Nachteile:
 - Interpolation nötig

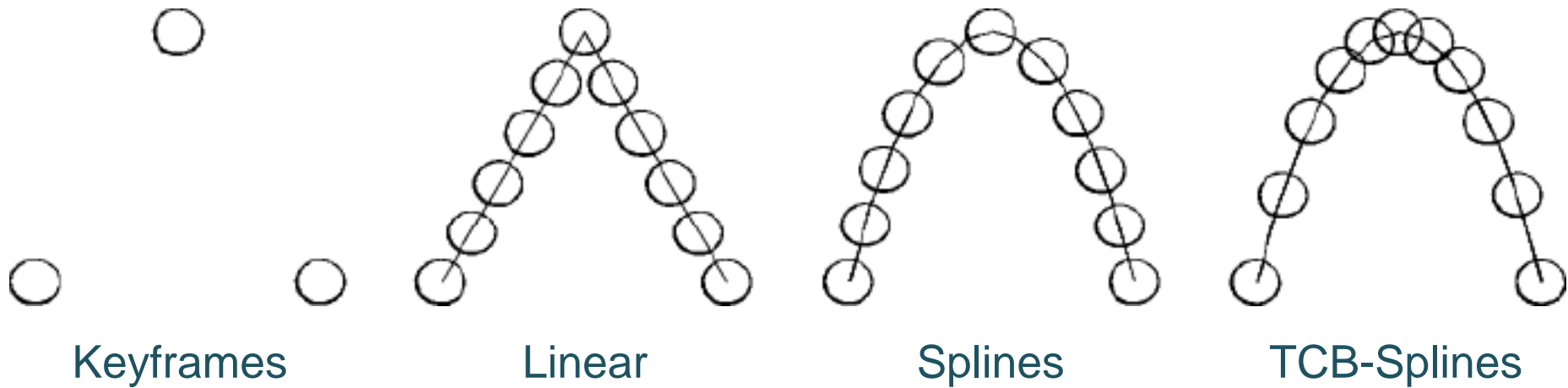


Keyframe 1 bei $t_1=0s$



Keyframe 2 bei $t_2=2s$





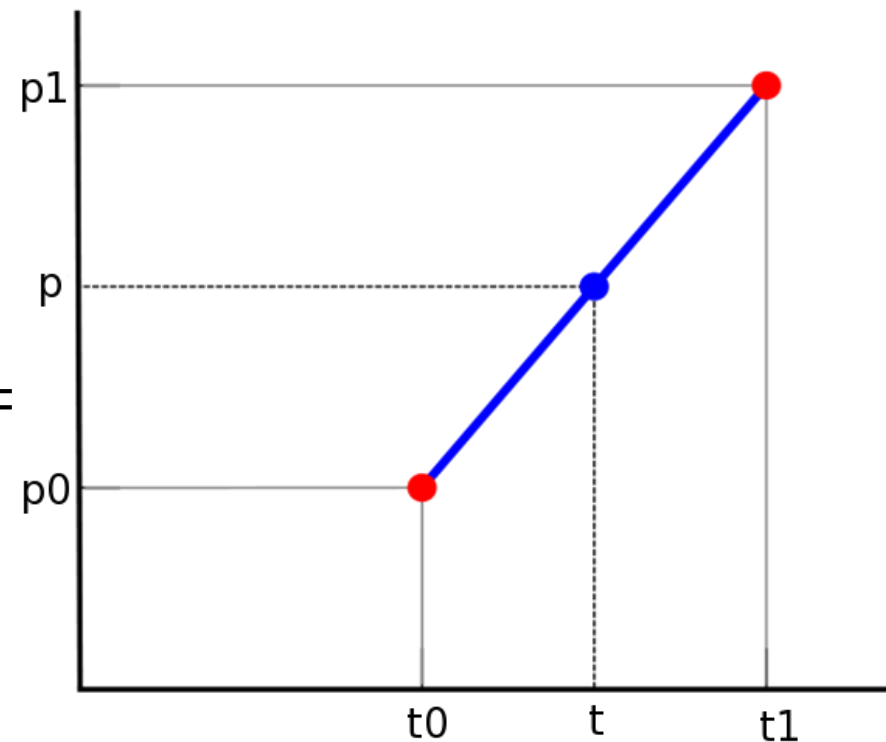
Anpassungen bei Interpolation sowohl über die Position als auch über die Zeit!

- Lineare Interpolation (lerp) ergibt eine gerade Linie zwischen zwei Punkten

$$m = \frac{p_1 - p_0}{t_1 - t_0} = \frac{p - p_0}{t - t_0}$$

$$p = p_0 + \frac{p_1 - p_0}{t_1 - t_0} \cdot (t - t_0) =$$

$$= p_0 + m \cdot \Delta t$$



- Setzen wir den Anfangs- und Endzeitpunkt jeweils auf 0 und 1 erhalten wir:

$$p = p_0 + (p_1 - p_0)(t - 0) = (1 - t)p_0 + tp_1$$

- Wir bezeichnen diese Funktion mit

$$B_{p_0 p_1}(t) := (1 - t)p_0 + tp_1$$

- Quadratische Bézier Interpolation:

„Interpolation zwischen 2 linearen Interpolationen“

$$B_{p_0, p_1, p_2}(t) = (1 - t)B_{p_0, p_1}(t) + tB_{p_1, p_2}(t)$$

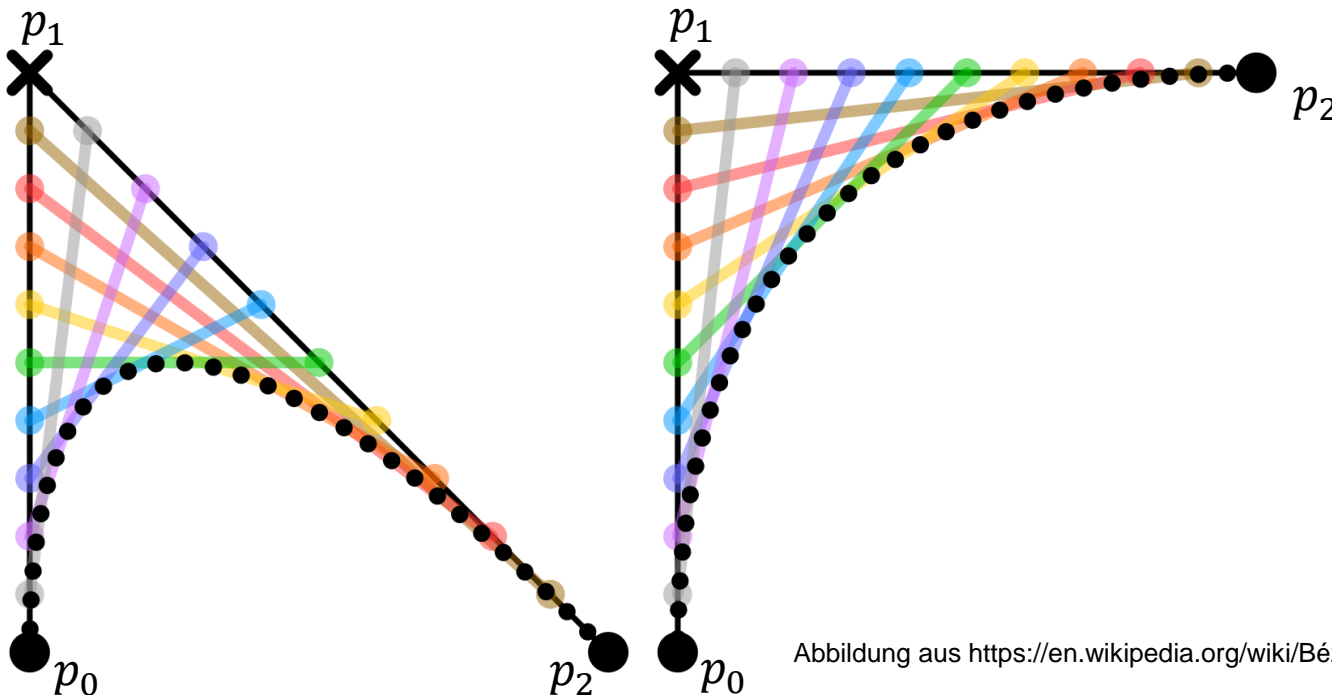
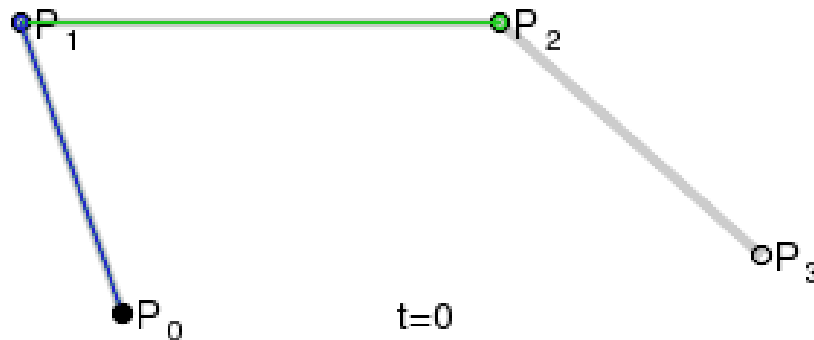


Abbildung aus https://en.wikipedia.org/wiki/Bézier_curve

- Kubische Bezier Interpolation

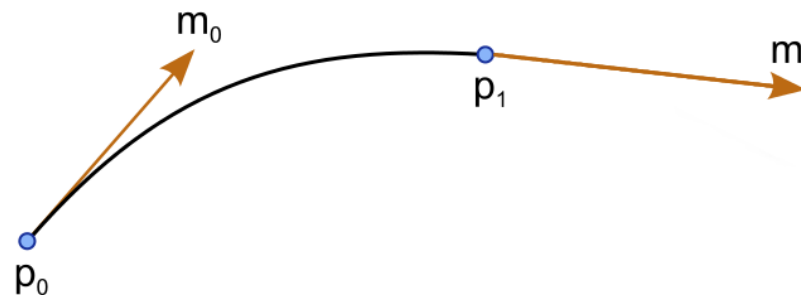
Bestimmung von Steigung im Anfangs- und Endpunkt

$$B_{p_0,p_1,p_2,p_3}(t) = (1-t)B_{p_0,p_1,p_2}(t) + tB_{p_1,p_2,p_3}(t)$$



Grafik aus <https://de.wikipedia.org/wiki/Bézierkurve>

- Ausmultipliziert ist $B_{p_0 p_1 p_2 p_3}(t)$ ein Polynom vom Grad 3 (also mit t^3)
- Dadurch ist die 2-te Ableitung überall stetig
- Die Beschleunigung der Bewegung macht also keine Sprünge



- Splines berechnen die Interpolation zwischen den Punkten über ein höhergradiges Polynom → Kurven möglich
- Interpolation in einem Segment $[p_0, p_1]$ über die ein- und ausgehenden Tangenten (m_0 bzw. m_1)
- Ausgehende Tangente eines Segments = Eingehende Tangente des nächsten Segments → Nur eine Tangente je Punkt, keine Knicke
- Interpolation für Punkt p von p_0 nach p_1 abhängig von Zeit t im Einheitsintervall $[0, 1]$ mit den hermetischen Basisfunktionen $h_{00} \dots h_{11}$:

$$p(t) = \underbrace{(2t^3 - 3t^2 + 1)}_{h_{00}} p_0 + \underbrace{(-2t^3 + 3t^2)}_{h_{10}} p_1 + \underbrace{(t^3 - 2t^2 + t)}_{h_{01}} m_0 + \underbrace{(t^3 - t^2)}_{h_{11}} m_1$$

- Ähnlich ist die Berechnung von Bézier-Kurven; *hier aber*. Stützpunkte statt Tangenten; Umrechnung ist jedoch möglich!
- *Aber*. Wie werden die Tangenten bei automatischer Interpolation festgelegt?

- TCB-Splines benutzen 3 Parameter (T, C, B) zur Berechnung der Tangenten
- Es kann doch wieder unterschiedliche ein- und ausgehende Tangente an einem Punkt geben → Knicke möglich
- Berechnung der eingehenden Tangente mIN_i und ausgehenden Tangente $mOUT_i$ am Punkt p_i über die Parameter T, C, B, sowie den vorangehenden und nachfolgenden Punkt p_{i-1} bzw. p_{i+1} über:

$$mIN_i = \frac{(1-T) \cdot (1-C) \cdot (1+B)}{2} \cdot (p_i - p_{i-1}) + \frac{(1-T) \cdot (1+C) \cdot (1-B)}{2} \cdot (p_{i+1} - p_i)$$

$$mOUT_i = \frac{(1-T) \cdot (1+C) \cdot (1+B)}{2} \cdot (p_i - p_{i-1}) + \frac{(1-T) \cdot (1-C) \cdot (1-B)}{2} \cdot (p_{i+1} - p_i)$$

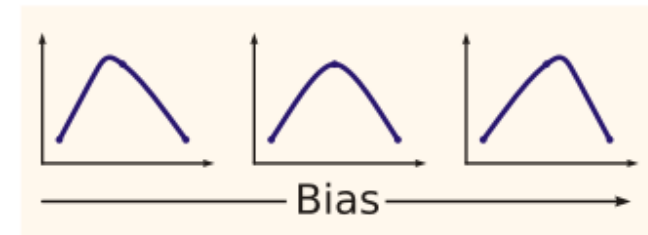
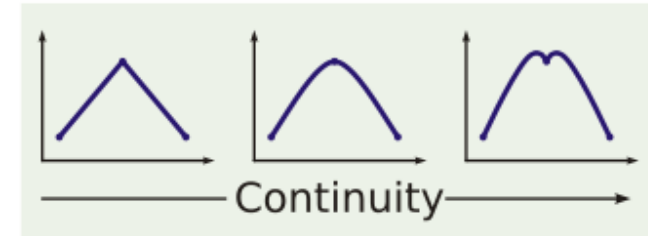
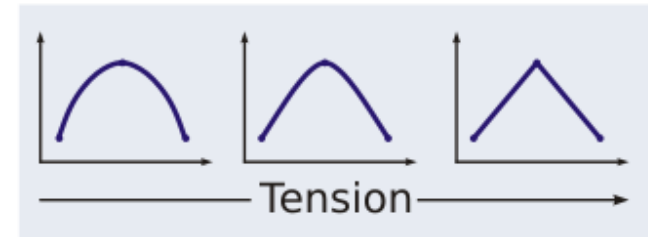
- Es kann auch unterschiedliche T, C, Bs je Eckpunkt geben
- Interpolation zwischen p_0 und p_1 dann über:

$$p(t) = h_{00}p_0 + h_{10}p_1 + h_{01}mOUT_0 + h_{11}mIN_1$$

- *Vorsicht:* Gleichungen sind im Einheitsintervall; falls unterschiedliche Dauer zwischen p_{i-1} zu p_i und p_i zu p_{i+1} , müssen diese auch in die Gleichung einfließen → Teilen durch $(t_i - t_{i-1})$ bzw. $(t_{i+1} - t_i)$

Auswirkungen der drei Parameter:

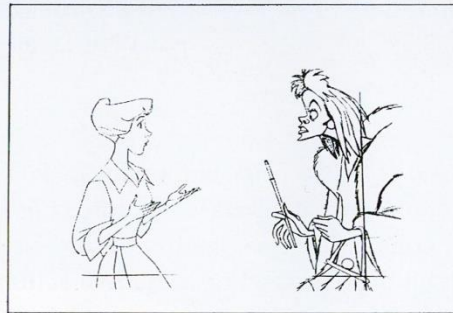
Tension (=Änderung der Tangentenlänge)	$T = -1 \rightarrow$ rund	$T = +1 \rightarrow$ eckig
Continuity (=Unterschiedliche Gewichtung von p_{i-1} und p_{i+1} auf ein- und ausgehende Tangente)	$C = -1 \rightarrow$ konvexe Ecken	$C = +1 \rightarrow$ konkave Ecken
Bias (=Grundsätzliche Gewichtung von p_{i-1} und p_{i+1} auf beide Tangenten)	$B = -1 \rightarrow$ Vorlauf	$B = +1 \rightarrow$ Nachlauf



Doris H. U. Kochanek and Richard H. Bartels, [Interpolating splines with local tension, continuity, and bias control](#), ACM SIGGRAPH 1984, vol. 18, no. 3, pp. 33-41.

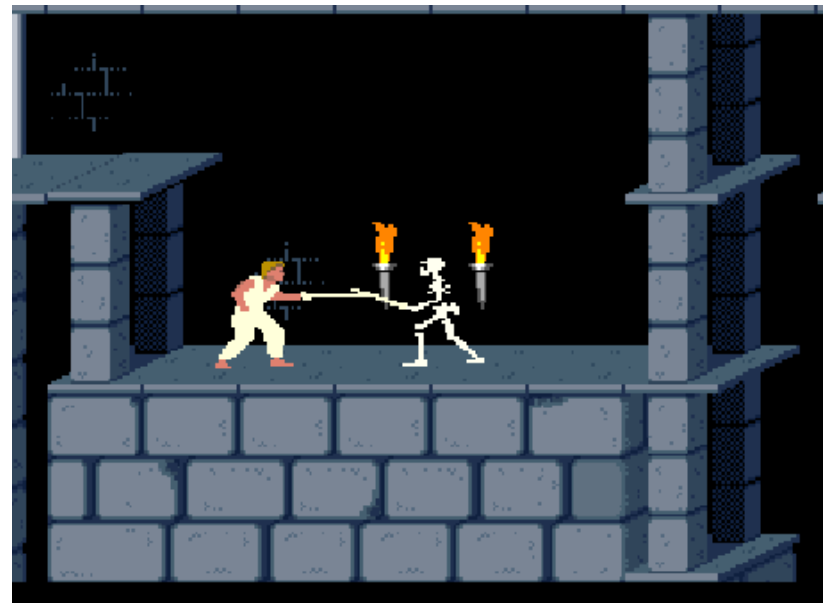
- Prinzip:
 - Animatoren übertragen Liveaufnahme (Key-) Frame für (Key-)Frame auf Animation
- Vorteil:
 - Mögliche Anpassung der Animation durch Animatoren
- Probleme:
 - Benötigt Schauspieler
 - Zeitintensiv
 - Übertragung oft nicht einfach (3D-Information?)

101 Dalmatiner



In Spielen ?

Prince of Persia (1989)



- Prinzip:
 - Wie bei Animationsfilmen erstellen Animators die Animationen
- Vorteile:
 - Große Ausdrucksstärke und Kontrolle
 - Gut mit AI kombinierbar
- Probleme:
 - Zeitaufwändig/Kostenintensiv
 - Bei AI Integration Entwicklungsaufwand/Tools
- *Beispiel:* The Last Guardian



- Prinzip:
 - Verwendung spezieller Sensoren (z. B. IR-Kamera-Array und Anzug mit IR-reflektierenden Kugeln), um Bewegungen eines Schauspielers aufzuzeichnen
 - Aufgezeichnete Daten werden verwendet, um Figur zu animieren.
- Vorteil:
 - Exakte Nachbildung der Schauspielerbewegungen im 3D-Raum → Sehr natürliche Bewegungen
- Probleme:
 - Benötigt Schauspieler
 - Evtl. Anpassung an Figur nötig
 - Meist sehr spezielle Animationen → schlechte Wiederverwendbarkeit
 - Keine Anpassung an aktuelle Spielsituation





Quantic Dream – Heavy Rain

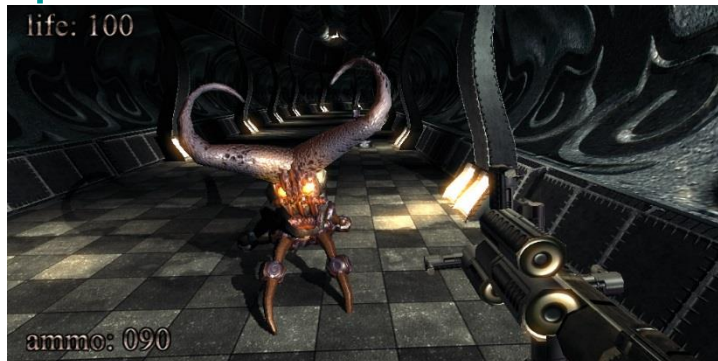
Gollum in „Der Herr der Ringe“

- Prinzip:
 - Animationen werden synthetisch durch einen Satz bestimmter Regeln erstellt, der ggf. auch dynamisch die aktuelle Spielsituation berücksichtigt
- Vorteile:
 - Reduzierung des Aufwands bei der Erstellung von Animationen
 - Animationen können zur Laufzeit angepasst werden
- Probleme:
 - Erstellung von komplexen/natürlichen Animationen oft schwierig
 - Animatoren benötigen Programmierkenntnisse

- <https://www.youtube.com/watch?v=LNidsMesxSE>

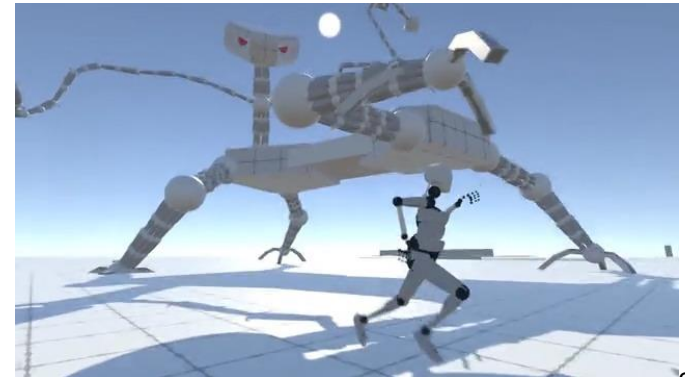


- https://www.reddit.com/user/Mystic_Mak



Sommersemester 2019

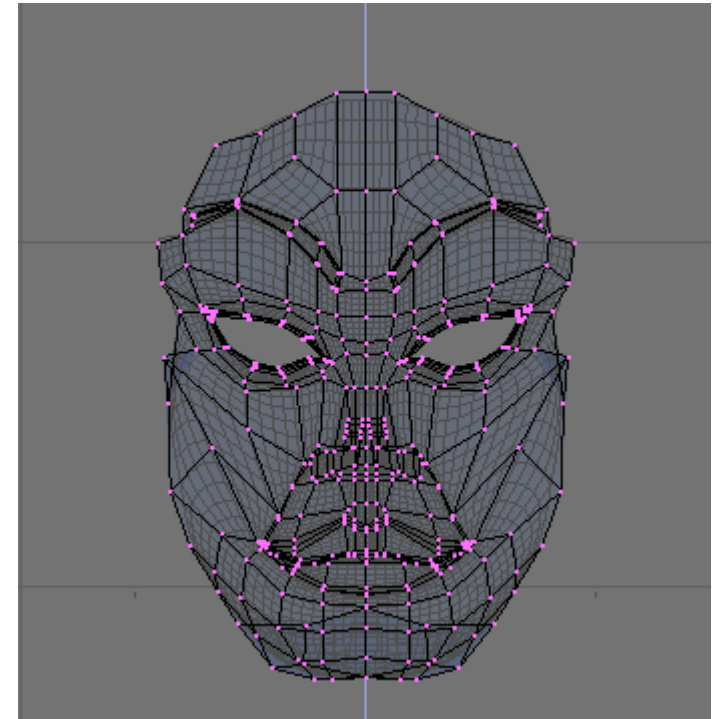
Einführung in die Spieleprogrammierung



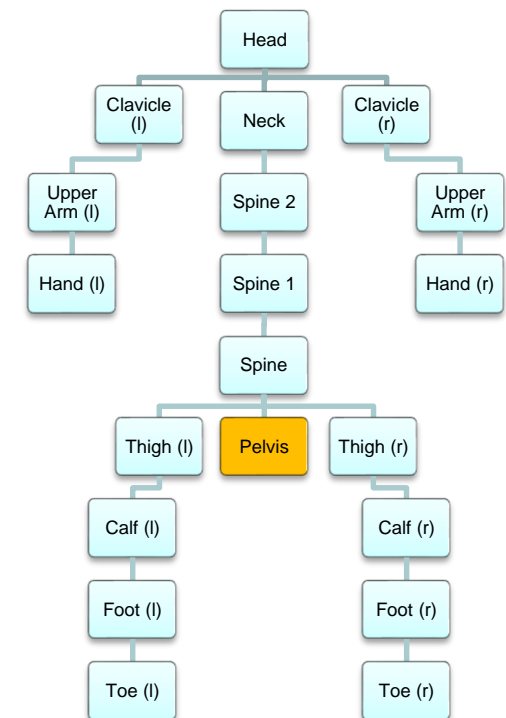
28

- Prinzip:
 - Physik als Modell zur Animationserstellung (Physik fester Körper vs. deformierbarer Körper)
 - Festlegung der Animation auf abstrakterer Ebene, z. B. für Trefferanimation: Anwendung eines Impulses an bestimmten Körperpunkt → Ragdoll-Effekt
- Vorteile:
 - Reduzierung des Aufwands bei der Erstellung von Animationen
 - Realistische Animationen
- Probleme:
 - Komplexität des physikalischen Modells
 - Schwere Kontrollierbarkeit
- *Beispiel:* Euphoria Engine (siehe Foliensatz 02); *hier:* neben Körperform/-masse auch virtuelle Muskeln und (motorische) Nerven

- Die Bewegungen der einzelnen Punkte (Vertices) eines Meshes (Netz aus Punkten = Oberfläche) werden direkt gespeichert/wieder abgespielt
- Vorteil:
 - Kontrolle jedes Punktes
 - hoher Detailgrad möglich
- Nachteil:
 - Speicherintensiv
 - Anpassungen in Echtzeit schwierig

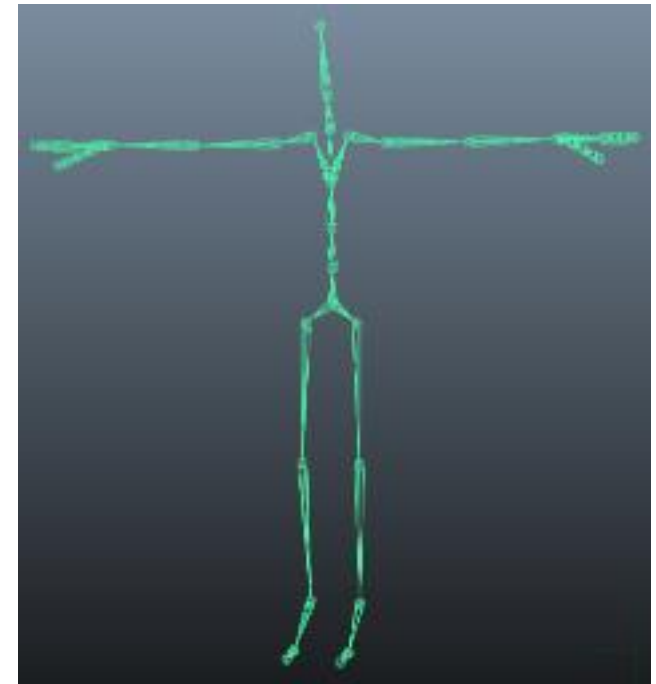


- *Rigging*: Ein Modell bekommt zusätzlich zur (Haut-) Oberfläche ein Skelett (meist hierarchisch organisiert)
- Verbinden (*Skinning*) der Hautpunkten (*Vertices*) mit Knochen (*Bones*) bzw. Gelenken (*Joints*)
- Transformation eines Knochens ändert ein ganzes Set an Hautpunkten ähnlich dem menschlichen Skelett
- *Vorteil*:
 - Es müssen nicht mehr alle Punkte separate animiert werden
 - Weniger Speicherintensiv
 - Wiederverwendbarkeit einer Animation bei anderen Charakteren mit gleichem Skelett
 - Bei Menschen/Tieren natürliche Nachbildung von Skelettbewegungen
 - Echtzeitanpassungen einfacher
- *Nachteil*:
 - Keine genaue Kontrolle je Vertex mehr
 - *Problem*: Dehnung/Stauchung der Haut an Gelenken
 - Noch keine perfekte Nachbildung der natürlichen Skelettbewegung, da Muskeln „fehlen“

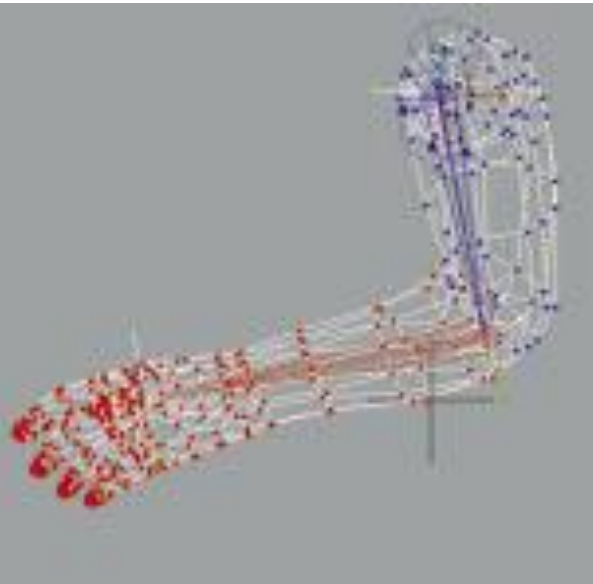
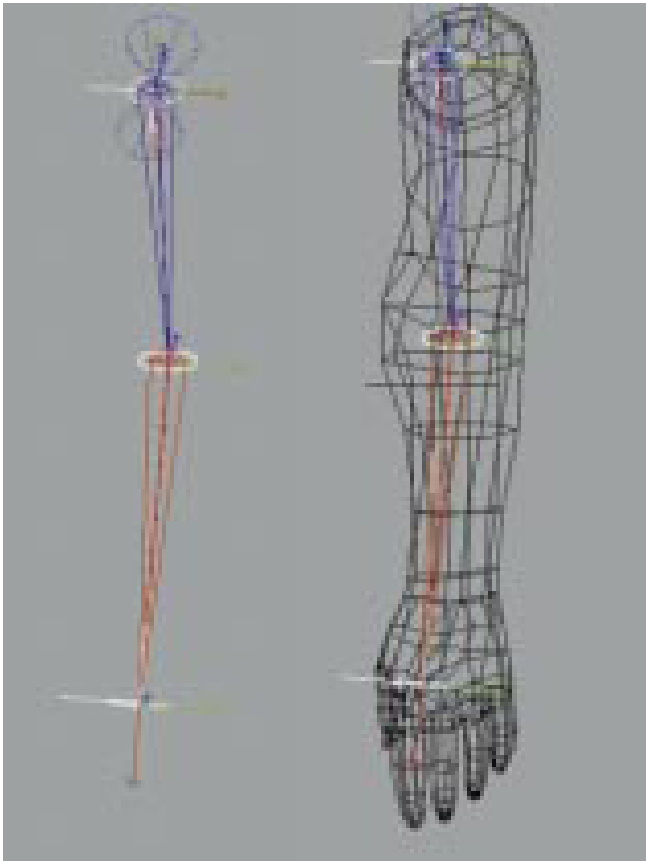


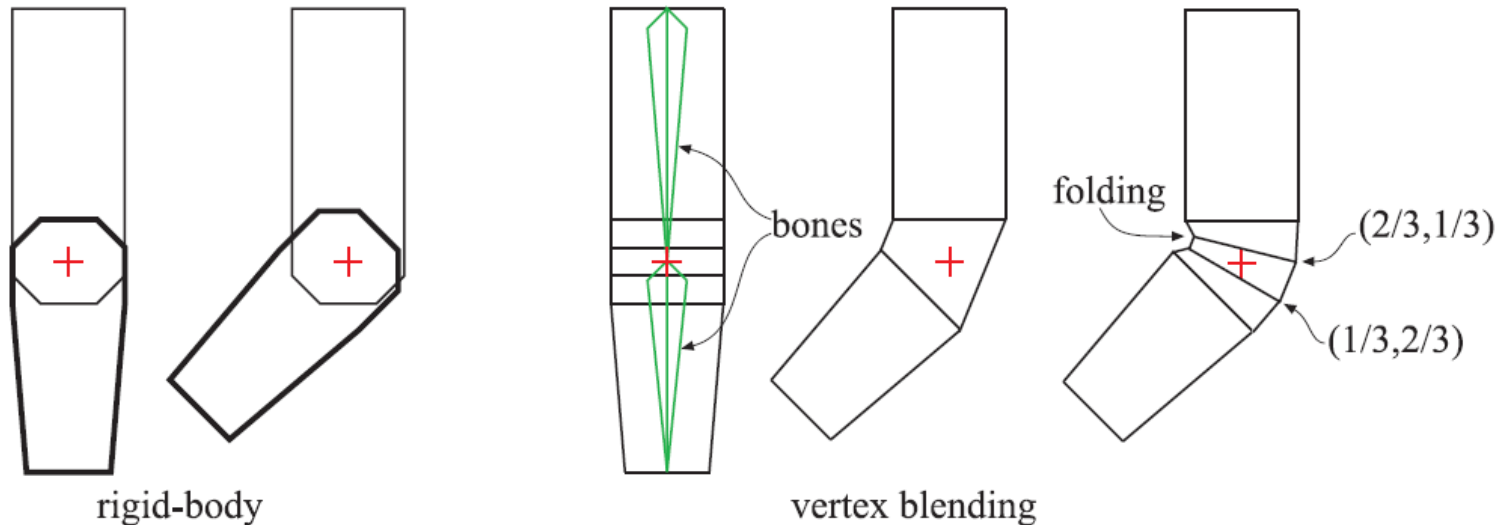
Hierarchisches Skelettmodel (in Unity3D)

- HIPS - spine - chest - shoulders - arm - forearm - hand
- HIPS - spine - chest - neck - head
- HIPS - UpLeg - Leg - foot - toe - toe_end



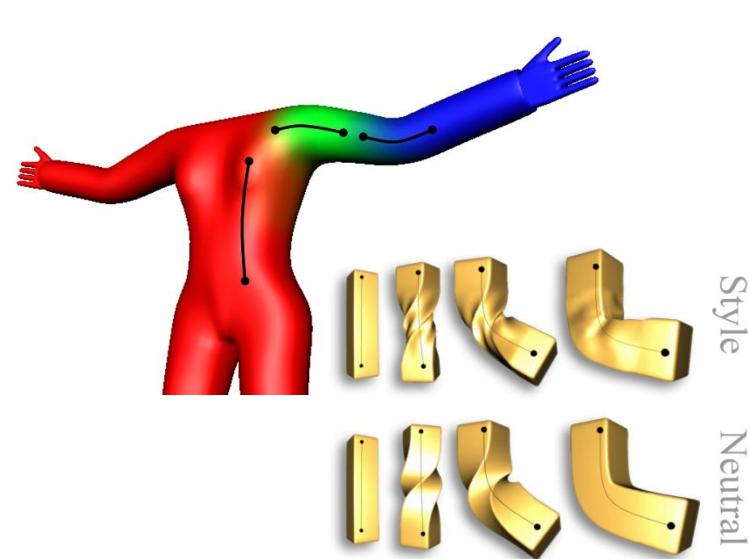
- Skinning geschieht im Vertex-Shader
- Die Positionen eines Vertices werden anhand der Transformationen der beeinflussenden Joints verändert



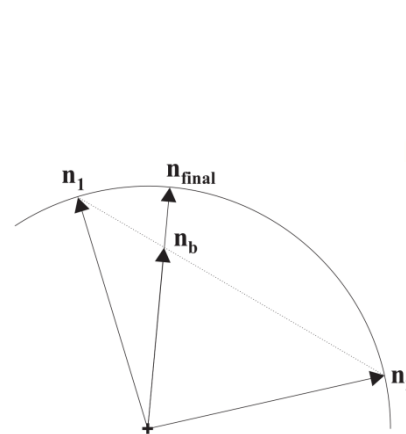


- *Problem bei Gelenken:* Rigid Body Transformation unnatürlich → Haut muss gestreckt/gestaucht werden (=Vertex Blending)
- Einfaches Strecken (jeder Punkt wird nur von einem Knochen beeinflusst) liefert sehr unnatürliche Ergebnisse
- *Lösung:* Punkte nahe den Gelenk werden von mehreren Knochen beeinflusst (=Smooth Skinning oder Linear Blending) → natürlichere Faltung/Dehnung

- Auch *Linear Blending* liefert manchmal ungewünschte Effekte
- *Lösungen*:
 - *Spline Skinning*: Knochen werden als spezielle Splines dargestellt → rundere Übergänge an Gelenken, realistische Verdrehungen, z. B. bei Metall möglich, benötigt aber anderes Skelett
 - *Dual Quaternion Blending*: Benutzt gleiches Skelett; zunächst normales lineares Blending (n_b), danach aber Anpassung um den Abstand Punkt-zu-Gelenk konstant zu halten (n_{final}) (vgl. Horde3D::Quaternion::nlerp)



S. Forstmann et al., *Deformation styles for spline-based skeletal animation*. ACM SIGGRAPH/Eurographics., 141-150 (2007)



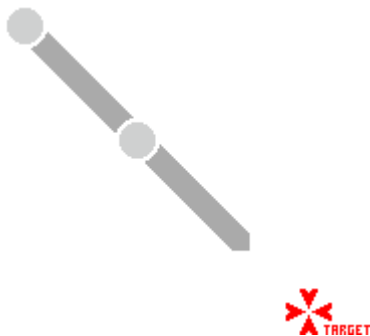
L. Kavan et al., *Geometric skinning with approximate dual quaternion blending*, ACM Trans. Graph., 27, 4, Article 105 (2008)

- Kinematische Kette (Kinematic Chain):
 - Hierarchie von Gelenken
 - Transformationen je Gelenk (z. B. Denavit-Hartenberg-Matrizen=Rotation+Translation)
 - Transformationen beeinflussen alle Kindknoten
 - Gelenke können Bewegungsbeschränkungen je Freiheitsgrad besitzen
 - Direkte (vorwärts) Kinematik und Inverse Kinematik zur Modifikation der Gelenktransformationen

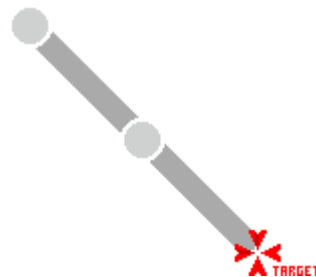
- Direkte Änderung der Gelenkrotationen
- Alle Kindknoten werden mit beeinflusst
- *Beispiel:* Änderung der Rotation des Schultergelenks → Geänderte Position des Fingers, da sich dessen Transformation an Hand der kinematischen Kette wie folgt berechnet:
 - *Schulterposition + (geänderte) Schulterrotation + Translation zu Ellenbogen + Ellenbogenrotation + Translation zu Hand + ... + ... + Translation zu Finger + Fingerrotation = Transformation des Fingers*

- (Gewünschte) Position des Endknotens ist bekannt
- Berechnung der einzelnen Gelenkrotationen entlang der kinematischen Kette
- *Vorstellung:* Hand soll an bestimmte Position bewegt werden → Ellenbogen- und Schulter müssen sich entsprechend rotieren

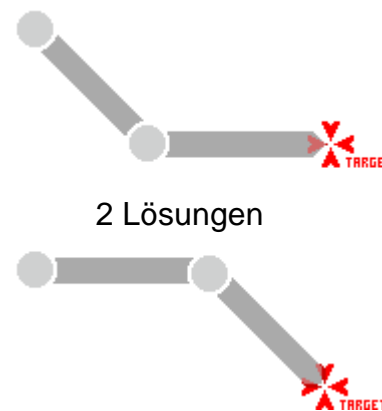
- Transformationsmatrizen je Gelenk ergeben zusammen ein Gleichungssystem mit mehreren Unbekannten für die Gesamtbewegung
- Berechnung möglicher Lösungen
- Probleme:
 - Großer Lösungsraum (viele Lösungen)
 - Leerer Lösungsraum (Ziel unerreichbar)
 - Rechenaufwändig



keine Lösung



1 Lösung

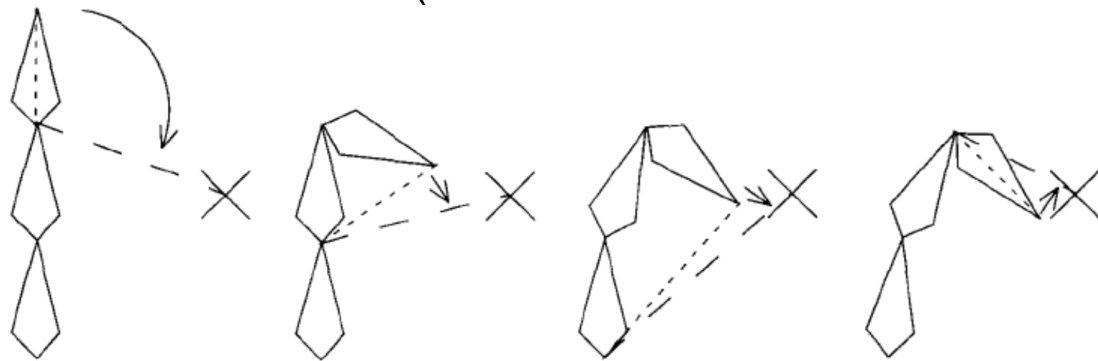


2 Lösungen



viele Lösungen
bei mehreren
Gelenken

- Annäherung an eine optimale Lösung durch mehrere Iterationen (numerische Methode), z. B. mit:
 - **Jacobian Inverse Method:** Mehrere Gelenke werden gleichzeitig iterativ verändert um sich der Lösung anzunähern
 - **Cyclic Coordinate Descent:** Vom Endknoten aus wird ein Gelenk nach dem anderen immer soweit gedreht, dass Abstand des Endknotens zu Zielpunkt minimal wird (lokale Minima) = “Greedy”-Ansatz; weniger aufwändig, aber auch unnatürlicher (Problematisch bei inkrementellen IK-Animationen)



```
while ( current_node != root_node )
{
    angle = current_node.calculate_angle(endEffektor, target);
    axis = current_node.calculate_axis(endEffektor, target);
    current_node.rotate(angle, axis);
    current_node = current_node.parent();
}
```

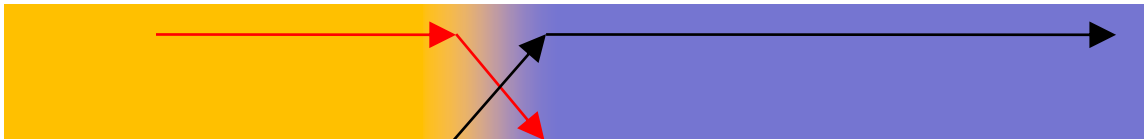
- Fließender Übergang zwischen Animationen (*cross fade*)



Animation 1

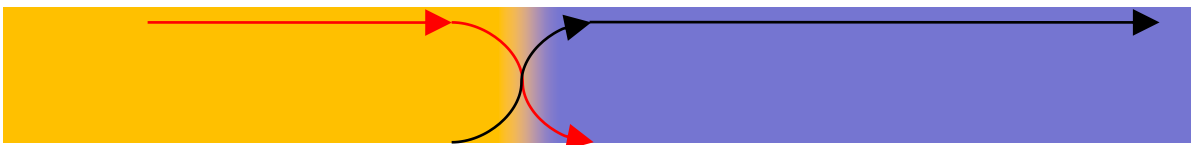


Animation 2



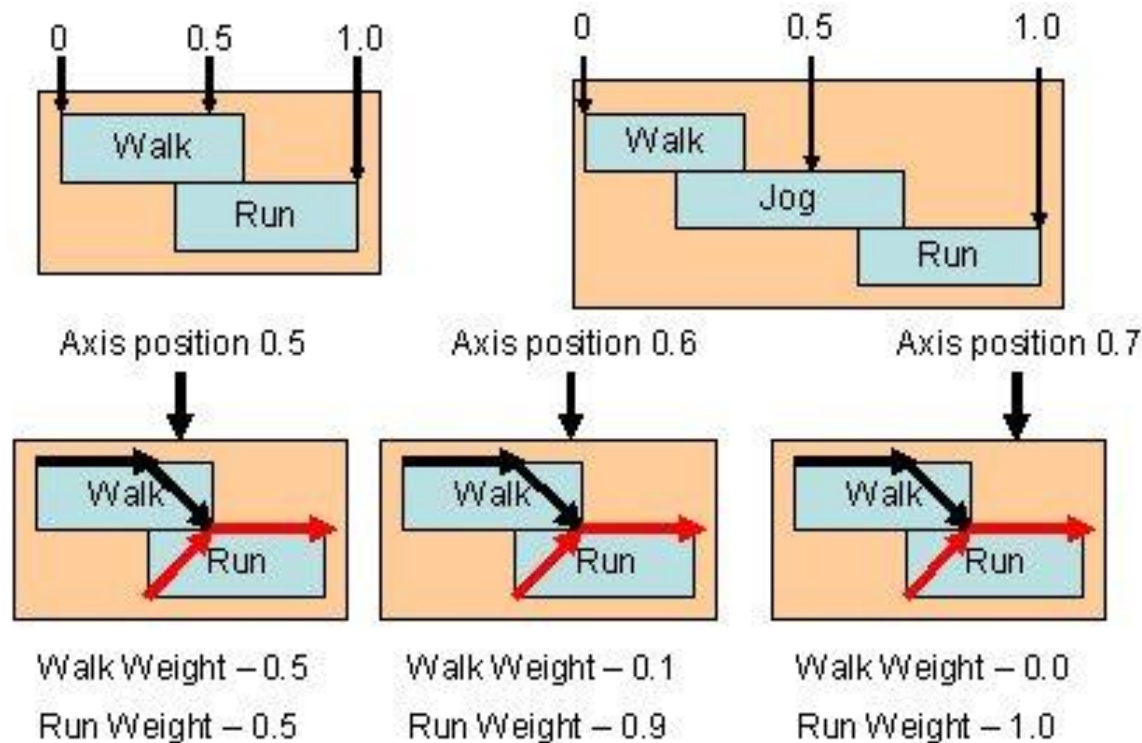
Animation 1+2

- Für alle Arten von Animationen möglich
- Blending kann auch nicht linear sein

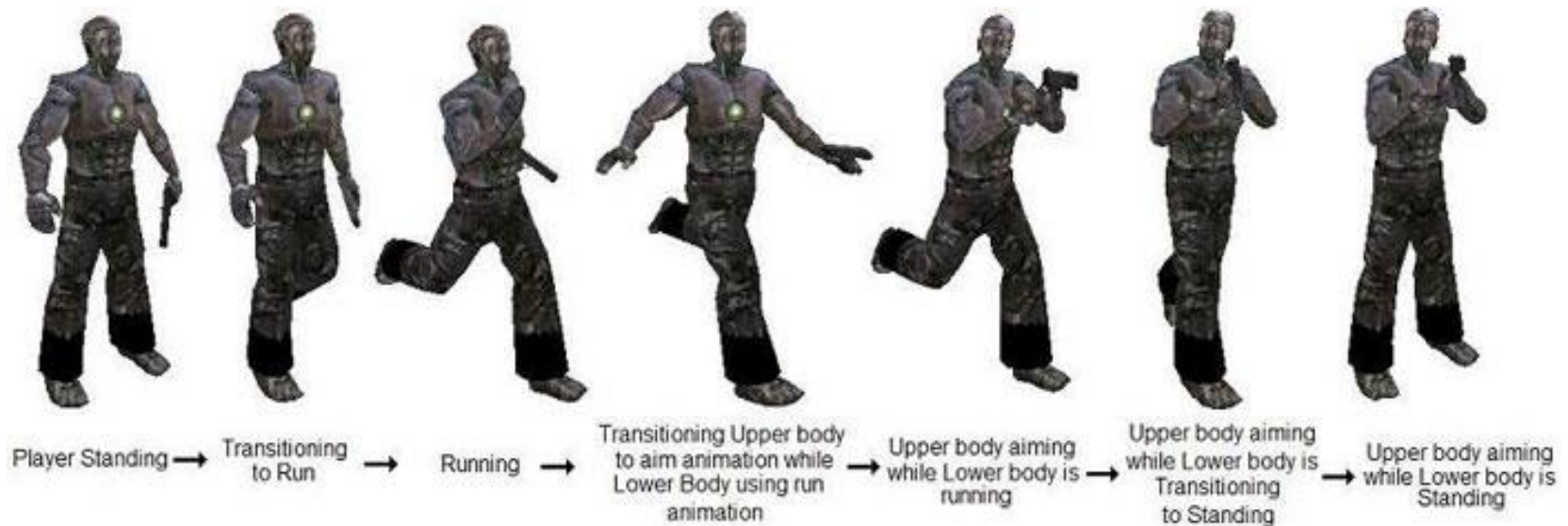


Animation 1+2

- Animationen befinden sich auf unterschiedlichen *Ebenen (Layern)* mit unterschiedlichen *Gewichten*
- Gewichte bestimmen, „wieviel“ von der Animation angewandt wird
- *Beispiel:* Überblenden von Laufanimationen unterschiedlicher Geschwindigkeiten



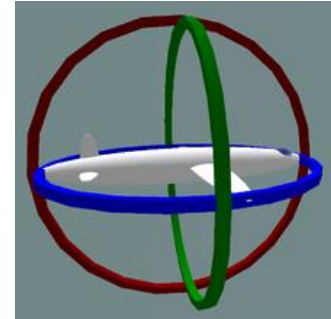
- Gleichzeitiges Abspielen verschiedener Animationen mit unterschiedlicher Gewichtung/Priorität
- Zusammensetzung der Gesamtanimation aus mehreren Teilanimationen, z. B. je Körperbereich
- Alternative für IK durch Überblendung verschiedener Skelettposes?



- Eulerwinkel (r.x,r.y,r.z oder α, β, γ oder pitch,yaw,roll)

- Nachteile:

- Anwendung der Rotation in mehreren Schritten wie beim Kardanring (=Gimbal; *hier*: Rotation der inneren Ringe wird „zuerst“ angewandt, äußere Ringe drehen die inneren mit)
 - Reihenfolge wichtig
 - Gimbal-Locks (= Falls eine Achse eine bestimmte Einstellung hat, fallen die beiden anderen Achsen zusammen)
 - Verlust eines Freiheitsgrades
 - Inkrementelle Animationen funktionieren nicht mehr
 - Mehrdeutigkeit (z. B. 180° Rotation um z-Achse = 180° Rotation um x-Achse + 180° um die y-Achse)



- Quaternionen (x, y, z, w)

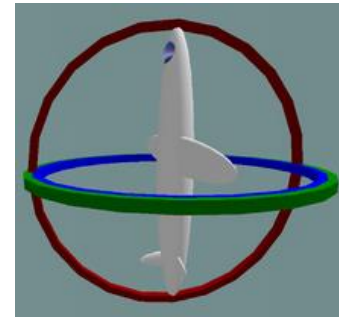
- Grundlage: vierdimensionaler Zahlenraum ähnlich den (zweidimensionalen) komplexen Zahlen
 - x, y, z= Vektor- bzw. Imaginärteil, w= Skalar- bzw. Realteil
 - Rotationen werden als Achse v + Drehwinkel α dargestellt ($0 < \alpha < 2\pi$), die Komponenten des Quaternions berechnen sich dann wie folgt:

$$w = \cos \frac{\alpha}{2}; (x, y, z) = v \cdot \sin \frac{\alpha}{2}$$

➤ Vektor (x,y,z) definiert die Achse v , dessen Länge sowie w definieren die Drehung α um diese Achse

- Vorteile:

- Eindeutige Darstellung
 - Kein Gimbal-Lock, da Rotation „in einem Schritt“
 - Quaternionen numerisch stabiler als Matrizen
 - einige Operationen einfacher (z. B. Interpolation mit gleichbleibender Winkelgeschwindigkeit mit *slerp*)
 - einfachere Umwandlung in Matrizen als bei Eulerwinkeln



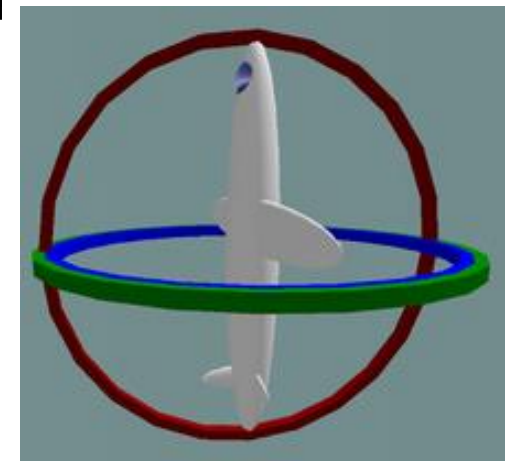
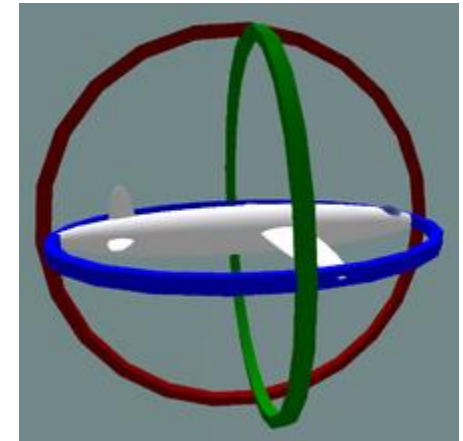
$$R = \begin{bmatrix} \cos \beta & 0 & -\sin \beta \\ 0 & 1 & 0 \\ \sin \beta & 0 & \cos \beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix} \begin{bmatrix} \cos \gamma & \sin \gamma & 0 \\ -\sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

mit $\alpha = 90^\circ$ folgt :

$$R = \begin{bmatrix} \cos \beta & 0 & -\sin \beta \\ 0 & 1 & 0 \\ \sin \beta & 0 & \cos \beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \cos \gamma & \sin \gamma & 0 \\ -\sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$


$$R = \begin{bmatrix} \cos \beta \cos \gamma - \sin \beta \sin \gamma & 0 & -\sin \beta \cos \gamma - \cos \beta \sin \gamma \\ \cos \beta \sin \gamma + \sin \beta \cos \gamma & 0 & \sin \beta \sin \gamma - \cos \beta \cos \gamma \\ 0 & 1 & 1 \end{bmatrix}$$

$$R = \begin{bmatrix} \cos(\beta + \gamma) & 0 & -\sin(\beta + \gamma) \\ \sin(\beta + \gamma) & 0 & -\cos(\beta + \gamma) \\ 0 & 1 & 1 \end{bmatrix}$$



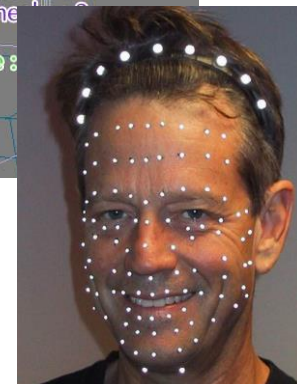
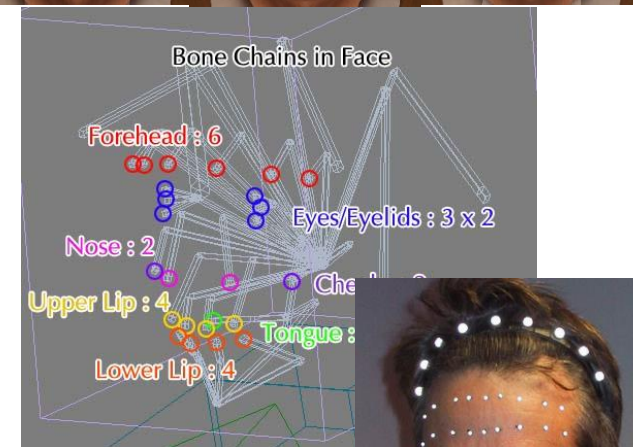
Mehr zu Rotationen: <http://www.euclideanspace.com/maths/geometry/rotations/index.htm>
<https://sundaram.wordpress.com/2013/03/08/mathematical-reason-behind-gimbal-lock-in-euler-angles/>



- Alle Spiele mit virtuellen Charakteren und Dialogen, z.B.:
 - Half-Life 2
 - Mass Effect 3
 - Heavy Rain
 - Battlefield 4 

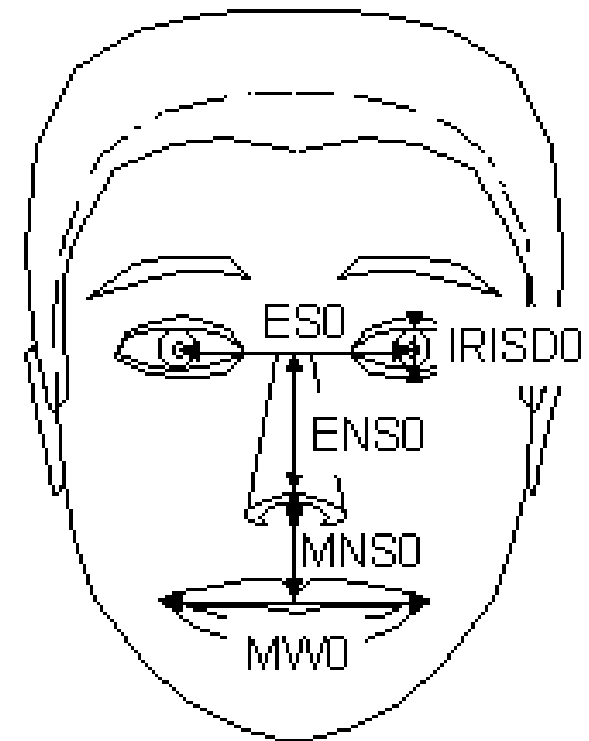


- Gesichtsanimation besteht aus:
 - Gesichtsausdrücke (Emotionen und sonstige Mimik)
 - *Sonderfall*: Sprachanimation (=Lippenbewegungen beim Sprechen)
 - Augen- und Kopfbewegungen
- Realisierung von Gesichtsausdrücken, z. B. mit:
 - Morph Target Animation
 - Speichern von Veränderten Vertexpositionen im Gesicht; spätere Anwendung mit Gewichtung; Kombination verschiedener Morph Targets möglich
 - Knochen Animation
 - Gesicht mit (nicht unbedingt der Natur entsprechenden) Knochenhierarchie versehen, um die Vertices der Gesichtshaut ähnlich zur normalen Skelettanimation zu deformieren
 - Facial Motion Capture (*auch*: Performance Capture)
 - Motion Capturing für Gesichter

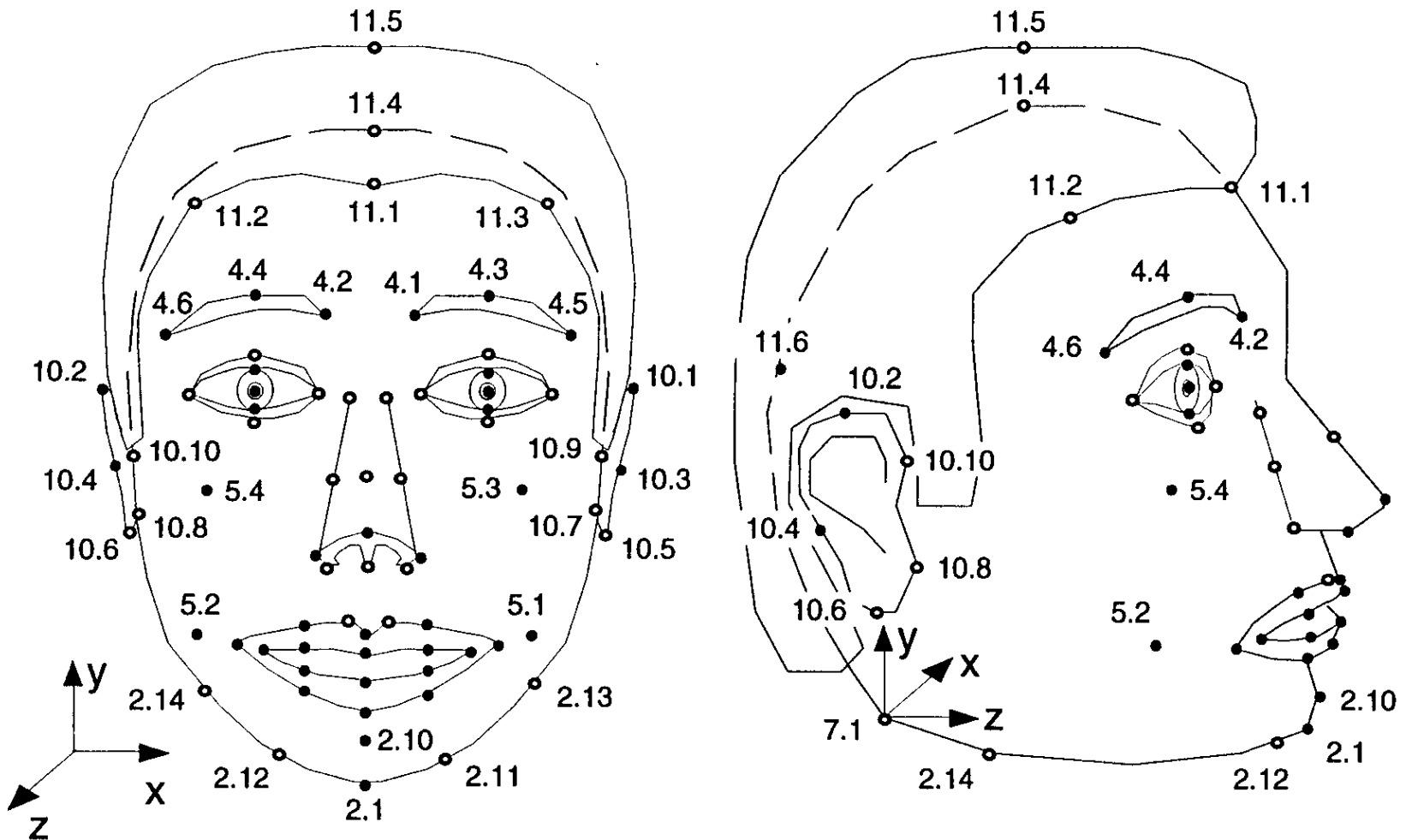




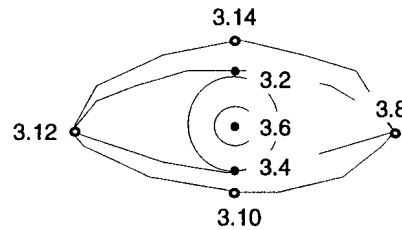
- MPEG-4 besteht aus verschiedenen Standards, neben Audio-/Video-Komprimierung, auch Abschnitt 7.15.3: Face animation parameter (FAP) data
- FAP Units (FAPU) werden als Maßeinheiten verwendet, z.B. Eye Separation $ES = ES0 / 1024$
- *Neutrales Gesicht* als Referenz:
 - Rechtshändiges Koordinatensystem
 - Kopfachsen sind parallel zum Weltkoordinatensystem
 - Blick in Richtung der z-Achse
 - Durchmesser der Pupille $1/3$ von IRISD0
 - Lippen berühren sich auf einer horizontalen Linie zwischen den Mundwinkeln
 - Zunge ist flach und die Zungenspitze berührt die oberen und unteren Zähne



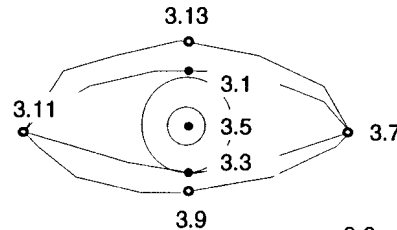
Facial Definition Points (FDP)



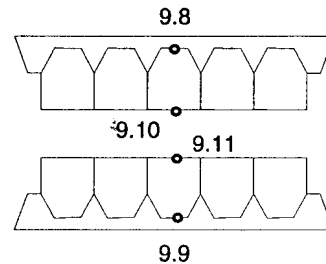
Facial Definition Points (FDP)



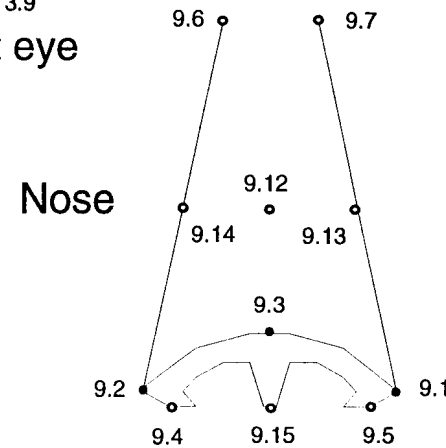
Right eye



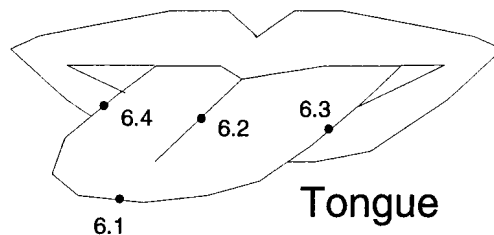
Left eye



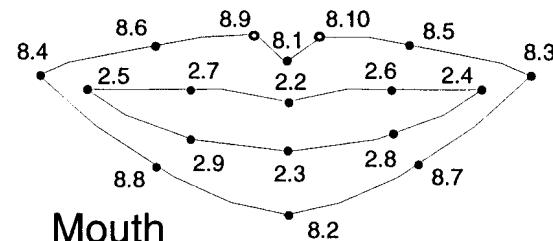
Teeth



Nose



Tongue



Mouth
















- MPEG-4 spezifiziert 84 Merkmalspunkte (FDPs) als Referenzpunkte für Gesichtsanimationen
- Jedes MPEG-4-verträgliche Gesichtsmodell muss diese enthalten
- Gesichtsanimationen werden durch die Bewegung von Merkmalspunkten definiert
- Übertriebene Werte erlauben Cartoon-artige Gesichtsausdrücke

- Es gibt 68 FAPs zur Repräsentation einer kompletten Menge von Basisanimationen für das Gesicht.
- Zwei Arten von Parametern höherer Ebene:
 - Viseme (FAP 1) sind das visuelle Gegenstück zu einem Phonem (die Standardmenge enthält 14 statische Viseme) → Lippenbewegungen
 - Gesichtsausdrücke (FAP 2) definieren die Gesichtsanimationen für die 6 Grundemotionen (Freude, Ärger, Trauer, Ekel, Angst und Überraschung)

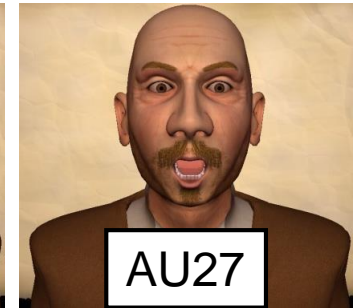
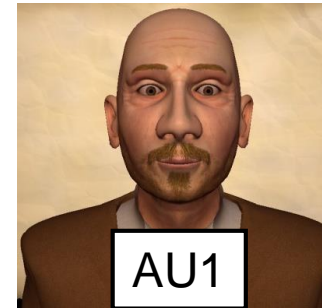
ID	FAP name	FAP description	Unit	Uni- or Bidirectional	Motion direction
...					
37	squeeze_l_eyebrow	Horizontal displacement of left eyebrow	ES	B	right
38	squeeze_r_eyebrow	Horizontal displacement of right eyebrow	ES	B	left
39	puff_l_cheek	Horizontal displacement of left cheek	ES	B	left
40	puff_r_cheek	Horizontal displacement of right cheek	ES	B	right
...					

- Für die Darstellung und Animation von Körpern existiert in MPEG-4 analog zur Gesichtsanimation die Body Definition (BDP) und Body Animation Parameter (BAP).
- Standardposition:
 - Stehende Position, Blick in Richtung z-Achse
 - Füße zeigen nach vorne
 - Arme seitlich platziert, wobei Handflächen nach innen gerichtet sind
 - Hände zeigen in Richtung y-Achse, bis auf den Daumen, der eine Neigung von 45 Grad aufweist.

- Facial Action Coding System (*FACS*) entwickelt von Paul Ekman und Wallace Friesen (1976)
- *FACS* klassifiziert mimische Muskelbewegungen im Gesichts- und Kopfbereich in 44 Bewegungseinheiten (*Action Units=AUs*) aus einzelnen oder mehreren Muskeln
 - *Sonderfall Action Descriptors*: Bewegungen, die sich auf mehrere/größere Muskelgruppen beziehen, z.B. Kopf-/Augenorientierung, aufgeblasene Backen, ...
 - AUs bekommen Intensität von A (Angedeutet) bis E (Maximum)
 - Kombination mehrere AUs möglich

<p>AU1</p>  <p>Inner brow raiser</p>	<p>AU2</p>  <p>Outer brow raiser</p>	<p>AU4</p>  <p>Brow lowerer</p>	<p>AU5</p>  <p>Upper lid raiser</p>	<p>AU6</p>  <p>Cheek raiser</p>
<p>AU7</p>  <p>Lid tighten</p>	<p>AU9</p>  <p>Nose wrinkle</p>	<p>AU12</p>  <p>Lip corner puller</p>	<p>AU15</p>  <p>Lip corner depressor</p>	<p>AU17</p>  <p>Chin raiser</p>
<p>AU23</p>  <p>Lip tighten</p>	<p>AU24</p>  <p>Lip presser</p>	<p>AU25</p>  <p>Lips part</p>	<p>AU26</p>  <p>Jaw drop</p>	<p>AU27</p>  <p>Mouth stretch</p>

- Designer erzeugt die AUs z.B. mit Morph Targets
- Gesichtsausdrücke werden erzeugt, indem man einzelne AUs (mit bestimmter Intensität) aktiviert
- Z.B. in der Source Engine, Horde3D, ...



* = AU1, 2, 4, 5, 20



Emotional Expressions



Conversational Expressions



Physical Conditions



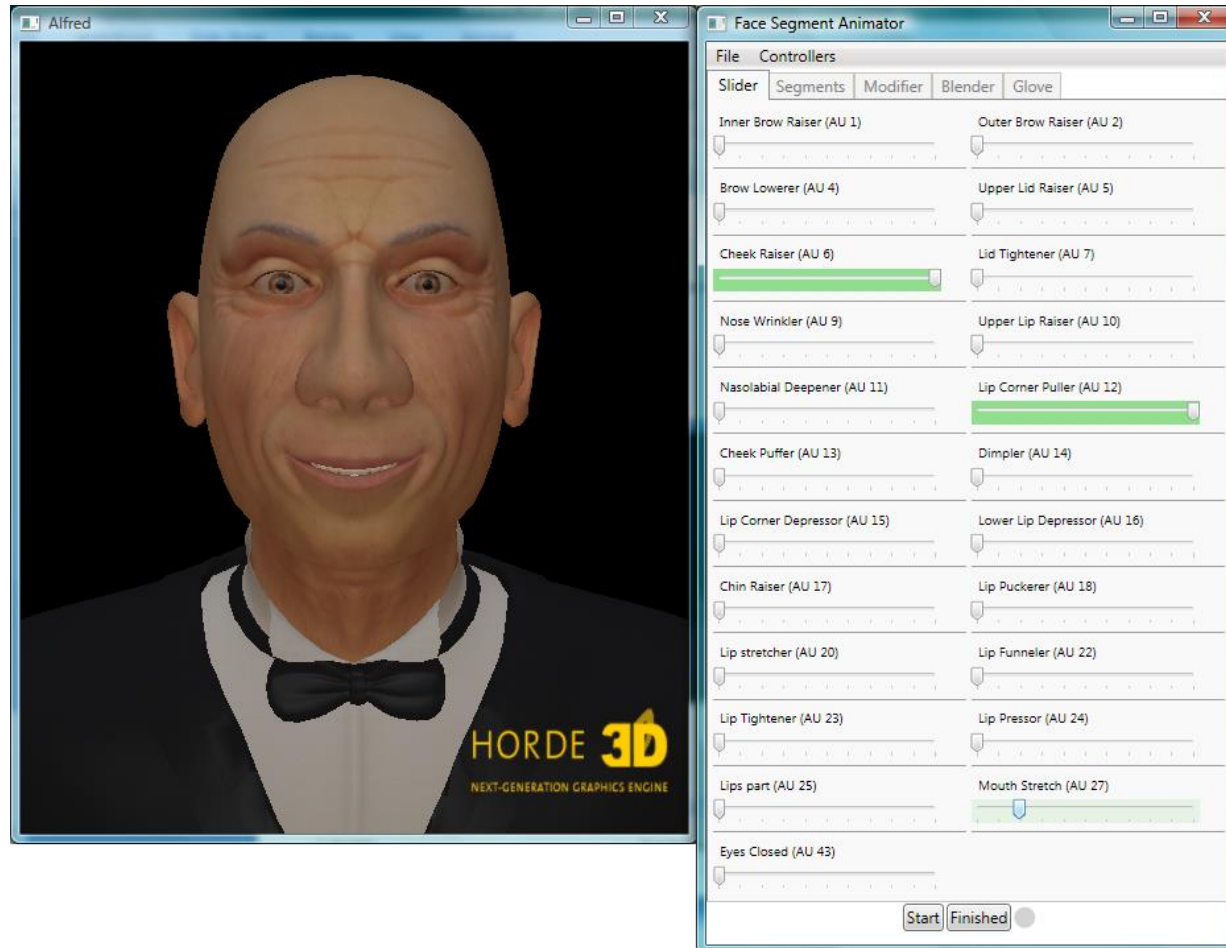
Cognitive Expressions

- Das „Facial Expression Repertoire“ der Filmakademie Baden-Württemberg bietet eine Menge von Gesichtsausdrücken kodiert nach FACS

<http://research.animationsinstitut.de/facial-research-tools/expression-repertoire/>



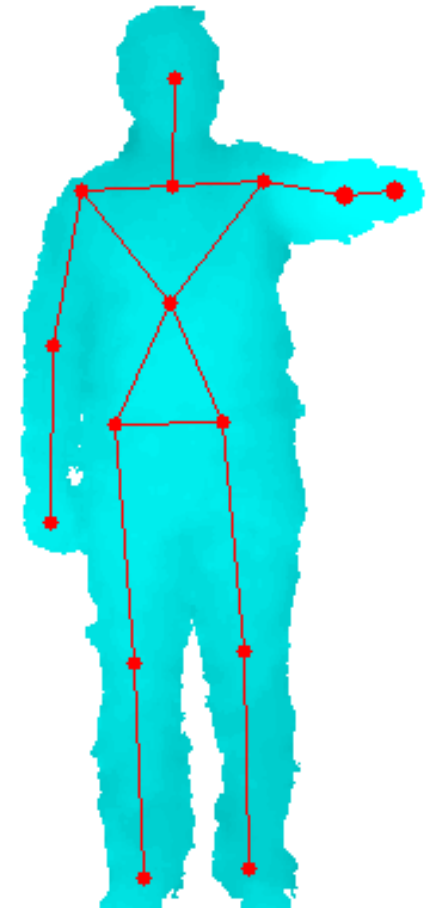
<http://evolution.anthro.univie.ac.at/institutes/urbanethology/projects/simulation/emosym/index.html>



<http://hcm-lab.de/projects/GameEngine/doku.php/showcase>

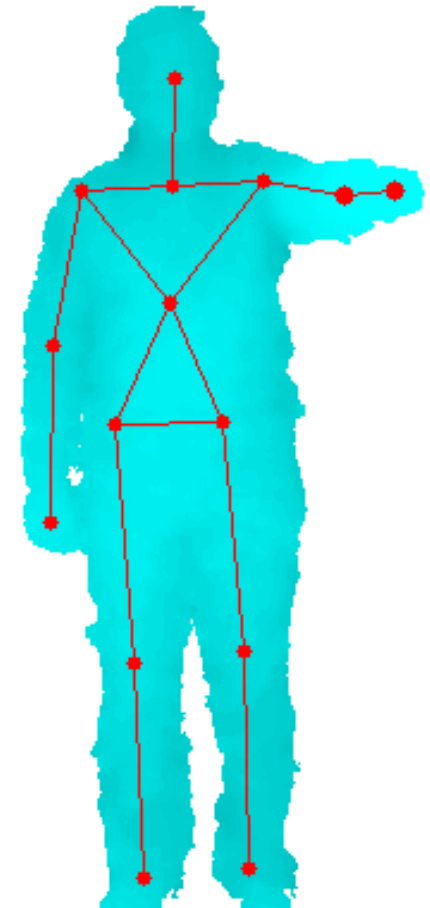


- Soll ähnlich wie FACS funktionieren, nur eben mit Körpersprache anstatt Mimik
- Wird von Paul Ekman (und seiner Firma) seit 2010 entwickelt (ursprünglich geplante Veröffentlichung: 2012)
- Soll alle Teile des menschlichen Skeletts mit einbeziehen

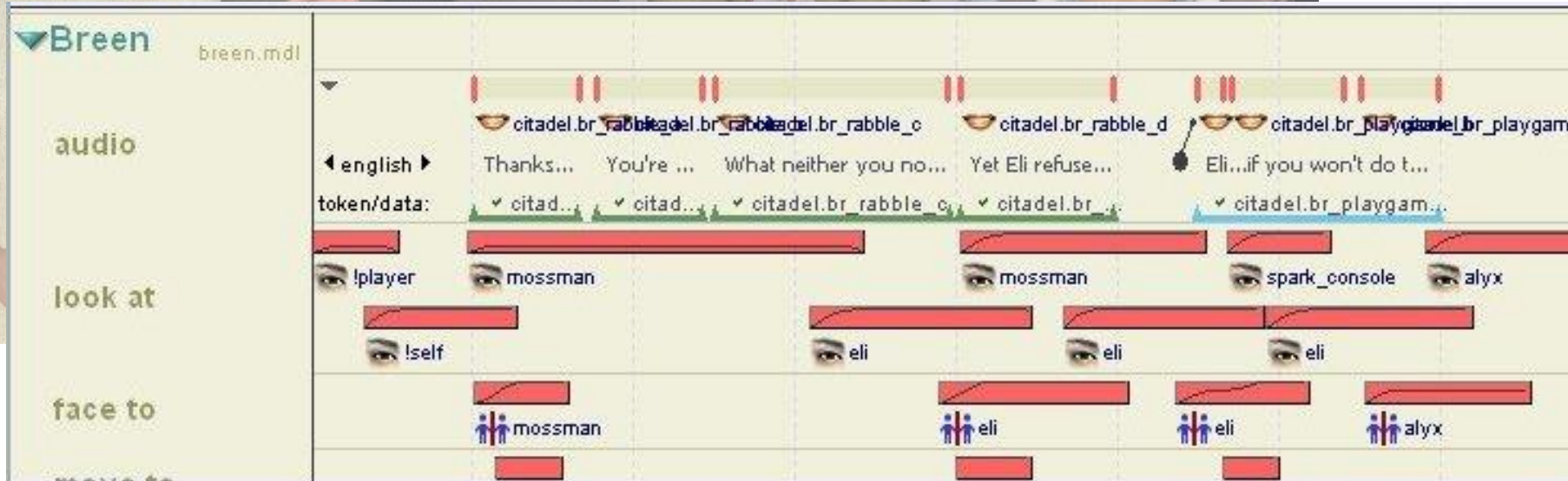


- Es soll dafür über 400 verschiedene primäre Codes + Varianten und weitere Modifikatoren (Bewegung, Ausmaß, Intensität, Abstand, Richtung, Höhe, ...) enthalten
- Es gibt allerdings (noch) keinerlei Interpretation oder Semantik

<http://www.ekmaninternational.com/paul-ekman-international-plc-home/news/eia-announcement.aspx>



FacePoser (Valve, Sourceengine)



http://developer.valvesoftware.com/wiki/Choreography_creation

Neben Gesichtsanimation aber auch Audio, Gestik, Posen, Bewegungen, ...

- Trennung von allgemeiner Verhaltensbeschreibung und der eigentlichen Umsetzung der Animationen auf einem speziellen Charakter
- **FML**
Hierarchische Gesichtsanimationen, zeitlicher Verlauf, ...
- **VHML**
Mensch-Computer Interaktion, Gesichtsanimation, Körperanimation, Dialogmanagement, Emotionale Darstellung, ...
- **MPML**
Präsentationssprache für virtuelle Charaktere
- **BML**
Verhaltensbeschreibung für virtuelle Charaktere

- Face Modeling Language

- Gesichtaktivitäten (*talk*=Textausgabe, *expr*=(emotionaler)Gesichtsausdruck, *hdmv*=Kopfbewegung, *fap*=MPEG-4 FAPs) werden gruppiert (*par*=Parallele Ausführung, *seq*=Sequentielle Ausführung *excl*=Wahl einer Aktion)
- Wiederholungen (*repeat*), Bindung an Events (*event*, *kbd*=Tastendruck), ...

```
<fml>
  <story>
    <act>
      <par>
        <hdmv type="yaw" value="15" begin="0" end="2000" />
        <expr type="joy" value="60" begin="0" end="2000" />
      </par>
      <excl event_name="kbd" repeat="kbd;F3_up" >
        <hdmv type="yaw" value="40" begin="0" end="2000" event_value="F1_up" />
        <hdmv type="yaw" value="-40" begin="0" end="2000" event_value="F2_up" />
      </excl>
    </act>
    ...
  </story>
</fml>
```

<http://dipaola.org/lab/research/iface/fml.html>

- Virtual Human Markup Language
 - Text mit zusätzliche Informationen in XML-Tags für bestimmte Passagen, z. B. Emotionen, Pausen, Betonungen, ...

```
<vhml>
  <person disposition="angry">
    First I speak with an <emph>angry</emph> voice and look
    very angry, <pause length="short"/> <surprised
    intensity="50"> but suddenly I change to look more
    surprised.</surprised>
  </person>
</vhml>
```

<http://www.vhml.org/>

- Multimodal Presentation Markup Language
- Sequentielle und parallele Aktionen
(`<Sequential>`, `<Parallel>`)
- Aktionen als Funktionsaufruf auf
einem virtuellen Charakter

```
<Action>
```

```
ken.turnHead(10, 0.2, 0.3, 0.2)
```

```
</Action>
```

```
<Action>
```

```
yuuki.speak("How're you doing?")
```

```
</Action>
```

```
<Action>
```

```
ken.gesture("BEAT_SINGLE", 0.2, 0.6)
```

```
</Action>
```

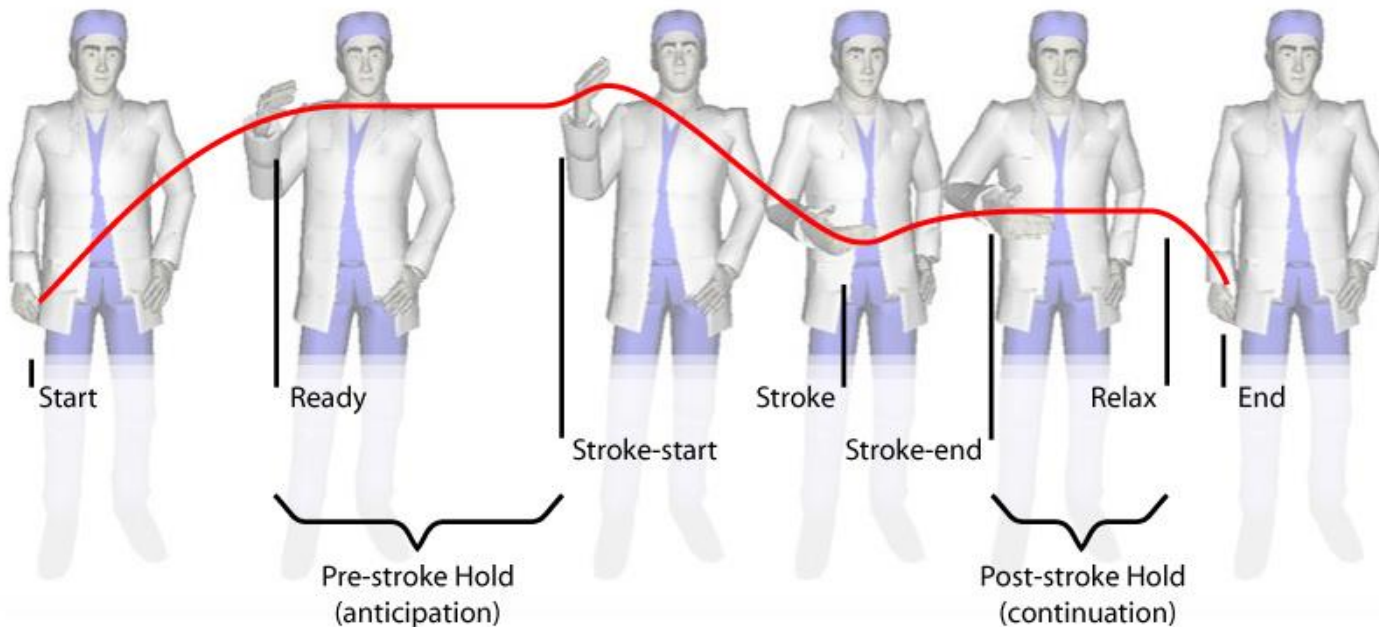


<http://research.nii.ac.jp/~prendinger/GALA2006/>

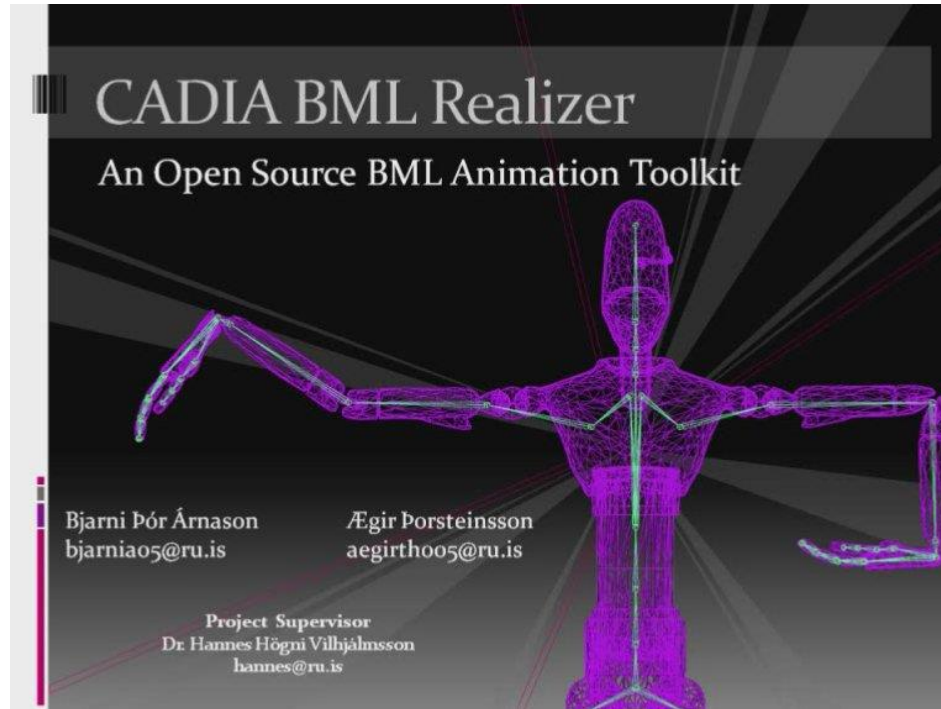
- Behavior Markup Language
- Eine Markupsprache (XML) zur Verhaltenssteuerung von virtuellen Charakteren mit Synchronisation der einzelnen Events
- Tags: `<head>`, `<torso>`, `<face>`, `<gaze>`, `<body>`, `<legs>`, `<gesture>`, `<speech>`, `<lips>`, `<wait>`

<http://www.mindmakers.org/projects/bml-1-0/wiki>

- BML benutzt Gestenphasen nach McNeill (Preparation[, Hold], Stroke[, Hold], Retraction)



- ```
<bml>
 <gaze target="PERSON1" />
 <speech> Welcome to my humble abode </speech>
 <head type="NOD" />
</bml>
```
- ```
<bml>
  <speech id="s1" start="0.0" type="audio/x-wav"
    ref="utterancel.wav" text="this is very nice">
    <tm id="tm1" time="0.1" /> <!-- This is -->
    <tm id="tm2" time="1.1" /> <!-- very nice -->
  </speech>
  <gesture id="g1" stroke="s1:tm2" type="BEAT">
  <head id="h1" stroke="g1:stroke" type="NOD">
  <gaze id="l1" ready="s1:tm1" relax="s1:tm2"
    target="book1">
</bml>
```



- <http://cadia.ru.is/projects/bmlr/>
- <http://www.smartbody-anim.org/>