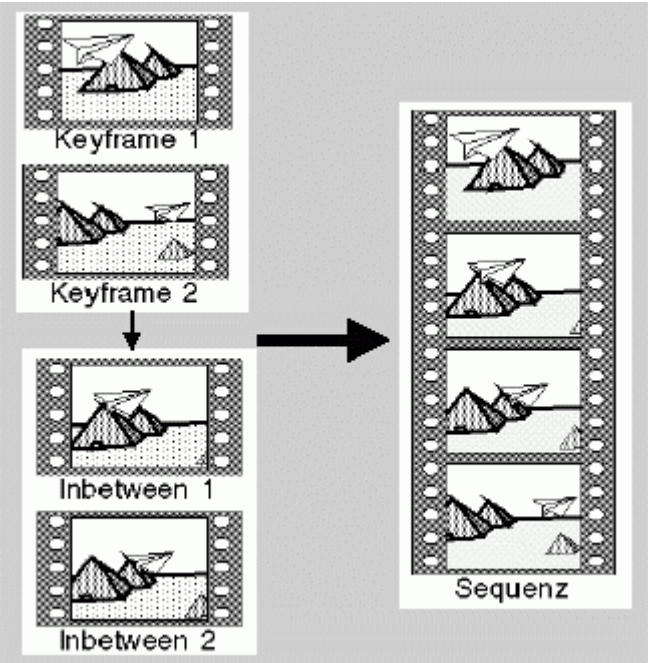
Keyframe-Animation

Computer-Version der traditionellen (Zeichentrick-) Keyframe-Animation

Keyframes: ausgewählte Standbilder, die vorgegeben (modelliert, akquiriert) sind
Zwischenbilder (*inbetweens*) werden konstruiert / berechnet.



2 Ansätze:

- bildbasiert, d.h. Interpolation der Pixel
- parametrisch, d.h. Interpolation im Modell (im Objektraum)

Bildbasierte Animation:

- Modifikation auf Pixelebene (2D)
- als Postproduction bei Filmen (z.B. Indiana Jones IV)
- Vorläufer aus der Biologie (D'Arcy Thompson: *On growth and form*, demonstriert morphologische Verwandtschaft zwischen verschiedenen Arten durch geometrische, stetige Transformation der Formen) und in alten Filmen (The Wolfman, 1941)
- Transformation von einem Objekt in ein anderes (Metamorphose), Ziel: möglichst stufenloser Übergang

"Morphing": Begriff gebräuchlich für 2D und 3D;
hier 2D

Morphing: Metamorphose von Objekt *A* zu Objekt *B*, d.h. Pixel ändert Position und Wert

Warping: Morphing von Objekt *A* zu verzerrtem Abbild *A'*, Verschieben von Pixeln bei gleichem Farbwert

Farbtransformation: Pixelwert-Veränderung bei gleicher Position, z.B. *Cross-Dissolve* (fade-in *B* + fade-out *A*)

Morphing von Bildern

Morphing= Warping+Farbtransformation

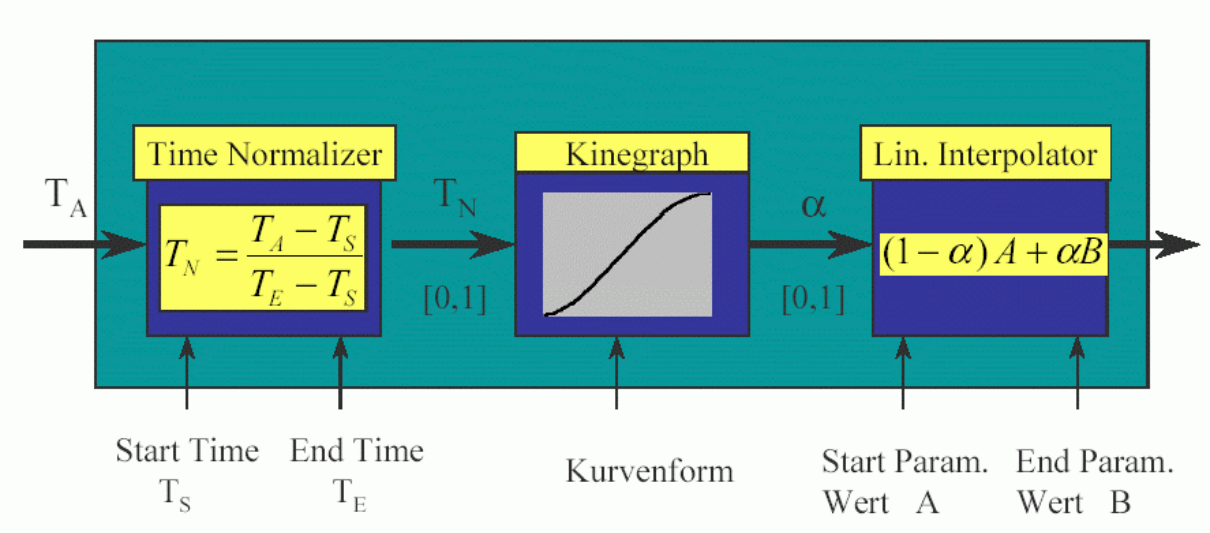
Sei C(x,y): Pixel der Farbe C an Position (x,y)

Morphing: S(x,y) -> T(x‘,y‘)
Warping: S=T und (x‘,y‘) = *warp*(x,y) und warp ist der benutzte Warpingalgorithmus (z.B. two spline mesh Verfahren, feature based morphing)
Farbtransformation: S(x,y) -> T(x,y), z. B. *cross-dissolve*: Pixel (x,y) hat Farbe I zum Zeitpunkt $t, t_{\min} \le t \le t_{\max}$

$$I(x,y,t) = S(x,y)(1 - \frac{t - t_{\min}}{t_{\max} - t_{\min}}) + T(x,y)(\frac{t - t_{\min}}{t_{\max} - t_{\min}})$$

die Gewichtungsfunktionen für $S(x, y)$ und $T(x, y)$ können verändert werden, z.B. langsamere Veränderung am Anfang, dann immer schneller!
"Kinegraph" legt den Verlauf der Gewichtsfunktion fest

Allgemeiner Parameter-Interpolator (auch für parametrische Interpolation im Objektraum):



Übliche Kinegraphen (Kurvenformen):

- hold (konstante Phasen, Plateau)
- linear
- ease in (slow in) – "weiches" Einsetzen
- ease out (slow out) – analog für die Endphase
- ease in ease out
- ...
- sketched (von Hand vorgegeben)
- splined
- Sprünge

Übliche Ease-Funktionen:

- quadratisch: $\alpha = 2T_N^2$ für $T_N = [0,0.5]$ ease in
  $\alpha = 2T_N - 2T_N^2$ für $T_N = (0.5,1]$ ease out
- sinusoidal: $\alpha = \sin(T_N \cdot \frac{\pi}{2})$
- diese Funktionen garantieren keine Kontinuität der Änderungsgeschwindigkeit an den Key-Punkten

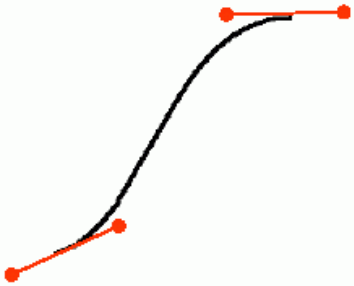
⇒ Verwendung von Splinefunktionen als Kinegraph

Basisgleichung: $\alpha = a \cdot T_N^{\,3} + b \cdot T_N^{\,2} + c \cdot T_N + d$

mit

$$a = m + n - 2$$
$$b = -2m - n + 3$$
$$c = m$$
$$d = 0$$

m: Steigung am Start-Key
n: Steigung am End-Key



übliche Kontrolltechniken für Spline-Kinegraphen:

- Benutzer gibt Steigungswerte explizit an (numerisch oder über grafische Editoren)
- System macht Annahmen über Steigungswerte, z.B. Gerade zwischen vorhergehendem und nachfolgendem Key-Wert
- Benutzer gibt Zeitdauer (in Frames) für ease-in und ease-out an

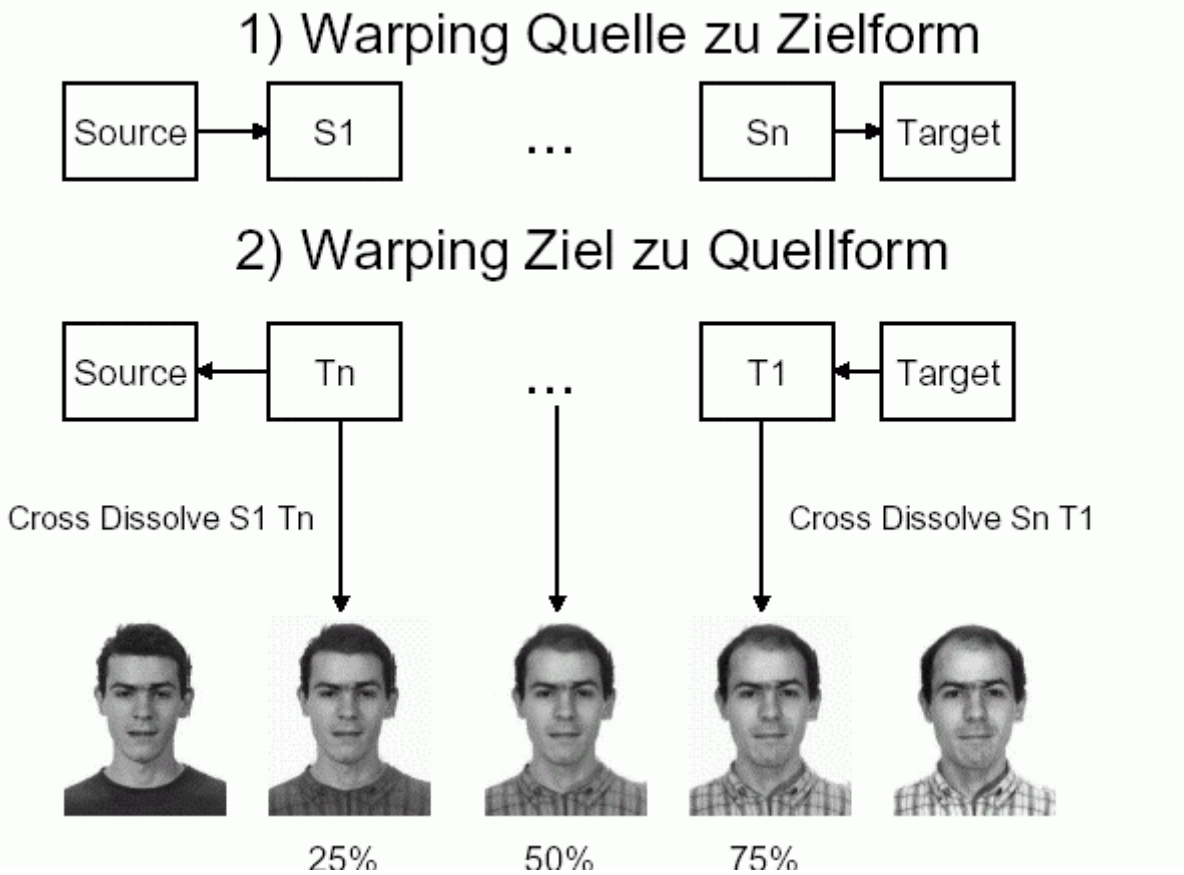
zurück zum Morphing:

Morphing-Sequenz erfordert Übergänge der Form (Quelle zu Zielform, invers: Ziel zu Quellform) und für jede Zwischenform gewichtete Mischung der Farbwerte von Quelle und Ziel (Cross dissolve)

verschiedene Verfahren für das Warping (gesteuerte Form-Interpolation):

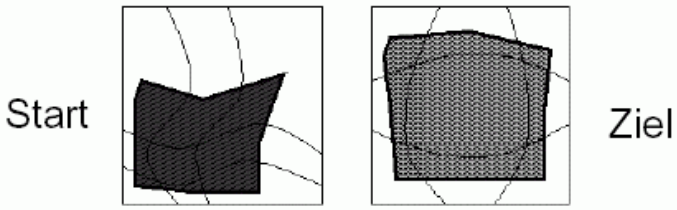
- Two-spline mesh warping
- feature-based warping

Konstruktion einer Morphing-Sequenz

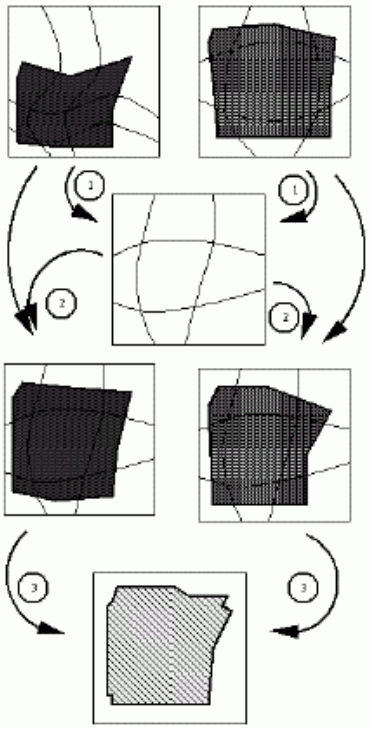


Two-Spline Mesh Warping

- Entwickelt von Douglas Smythe bei ILM (Willow, Indiana Jones IV, Abyss)
- Benutzer spezifiziert Punktgitter durch das Splines gelegt werden (spline mesh)



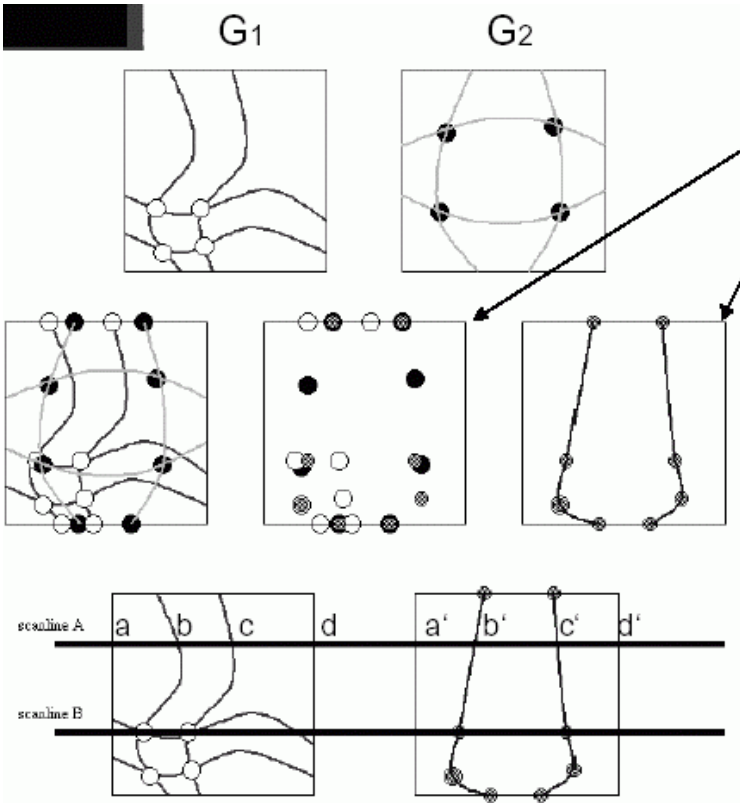
- Übergang von Netz 1 zu Netz 2 durch Berechnung von **Inbetweens**
- Interpolation der Kontrollpunkte
- 2-Pass Verfahren



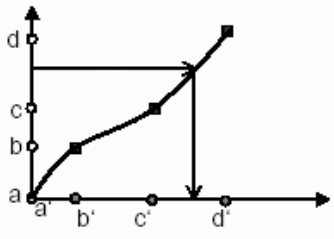
Punkte an den Rändern
bleiben konstant

Wie erhält man Inbetweens?

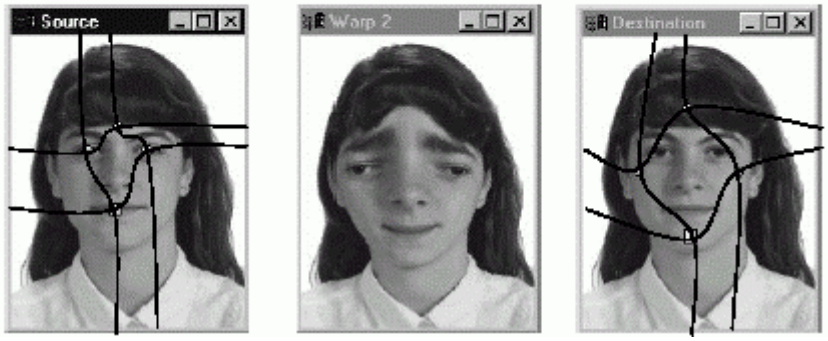
1) Berechne Zwischennetz
2) Warpe Start- und Zielbild in das Zwischennetz
3) Cross-Dissolve der beiden Bilder



1) Verschiebe Kontrollpunkte: X-Position von G2, Y-Position von G1
2) Berechne vertikale Splines durch neue Punkte
3) Basis für ersten Durchlauf einer Scanlinie
4) Berechne X-Abschnitte der Kontrollpunkte in G1 und G2 (a,b,c,d)
5) Berechne Mapping für genau diese Scanlinie
6) Transformiere Pixel



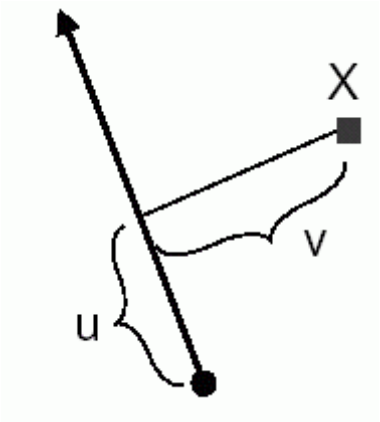
Erster Pass: Warping in X-Richtung
Zweiter Pass: Analog in Y-Richtung



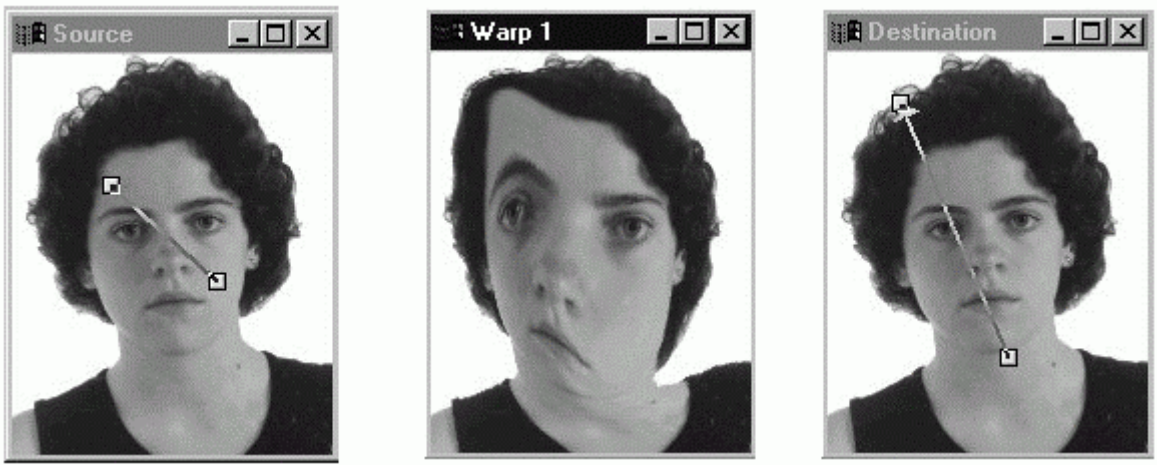
Beispiel

Feature-based Morphing:

- entwickelt von Beier & Neely (Pacific Data Images)
- Animator hat bessere Kontrolle über Morphingprozess
- Dichte des Netzes beim Two-Spline Mesh Warping ist homogen; hier kann dagegen die Dichte durch eine interaktiv bestimmte Zusatzlinie markiert werden
- Startlinie wird auf Ziellinie abgebildet
- es wird mit einem relativen (2D-) Koordinatensystem bzgl. dieser Startlinie gearbeitet (*u*, *v* - System)



Beispiel:

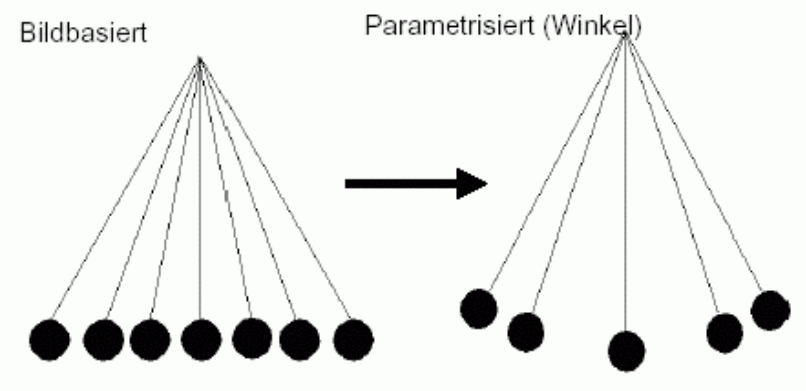


3D-parametrisches Keyframing

- 3D-Szenenbeschreibung wird parametrisch dargestellt (im Objektraum)
- Parameterwerte dieser Beschreibung werden für Keyframes (ausgewählte Zeitpunkte) eingestellt
- Inbetweens (bzw. die entsprechenden Parameter) werden durch Interpolation ermittelt

Gründe für Übergang zu 3D-Parametrisierung und -Interpolation:

- Interpolation auf Pixelebene kann zu Schwierigkeiten bei 3D-Strukturen führen
- unnatürliche Effekte bei bildbasierter Interpolation:



Auswahl der zu interpolierende Parameter durch das Animationssystem oder durch den Benutzer

- potenziell kommt jedes Attribut der erstellten Modelle und der Szene in Frage
- z.B. Position, Orientierung, Materialeigenschaften, Gestalt, Textur, Umgebung...
- 1 Parameter für jeden Freiheitsgrad des Modells: können sehr viele sein, Über-Parametrisierung (= zu viel Aufwand)!
- potenziell für jeden Parameter andere key-Zeitpunkte
- nicht jedes Attribut ist wirklich geeignet zur Interpolation
- kein Animationssystem unterstützt / kennt alle Modellierungs-verfahren
- Art der Interpolation ist wichtig, muss zum Typ des Parameters passen!

Beispiel: Interpolation von Positionen (3D-Vektoren)

Finde Weg zwischen zwei Positionen, oder genauer:
Finde Werte *P* zwischen zwei Stützstellen *X, Y* mit $P, X, Y \in \Re^m$

$$X = \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix}, Y = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \xrightarrow{???} P = \begin{pmatrix} p_1 \\ \vdots \\ p_m \end{pmatrix}$$

$$p(t) = (1-t)X + tY, t \in [0,1]$$

Was kontrolliert *t* ? Frameanzahl F_{end}-F_{start} oder Zeit T_{end}-T_{start}

$$t = \frac{T - T_{start}}{T_{end} - T_{start}} \qquad t = \frac{F - F_{start}}{F_{end} - F_{start}}$$

Lineare Interpolation

$$p(t) = (1-t)X + tY, t \in [0,1]$$

Animation von Werten (Beispiel Objektposition)

- Gegeben Startposition X, Endposition Y, Anfangszeit T_{start}, Endzeit T_{start} (analog auch Frames F, F_{end}, F_{start})
- Berechne t mittels Zeit T, T_{end}, T_{start} (oder F, F_{end}, F_{start})
- Berechne Objektposition P(x,y,z) mit t, X,Y
- Render Objekt an Position P(x,y,z)

Nicht-lineare Interpolation

- Bewegungen sind in der Realität stets nicht-linear und folgen dynamischen Gesetzmäßigkeiten
- Kinematische Gesetze beschreiben dies vereinfacht
- Approximation durch handhabbare mathematische Funktionen:

Trigonometrische Funktionen
Parametrische Überblendung (Quadratisch, Kubisch)
Hermite Überblendung
Parabolische Überblendung
Splines, insbesondere Catmull-Rom spline

Nichtlineare Funktionen:
häufig verwendet: Sinusfunktion ("Ein- und Ausschwingen")

$p(t) = (1–t)X + tY$ mit $t = \sin \alpha$, $0° \leq \alpha \leq 180°$

weitere Techniken:
Verwendung von Polynomen (quadratisches, kubisches Überblenden)

Techniken für parametrische Überblendung

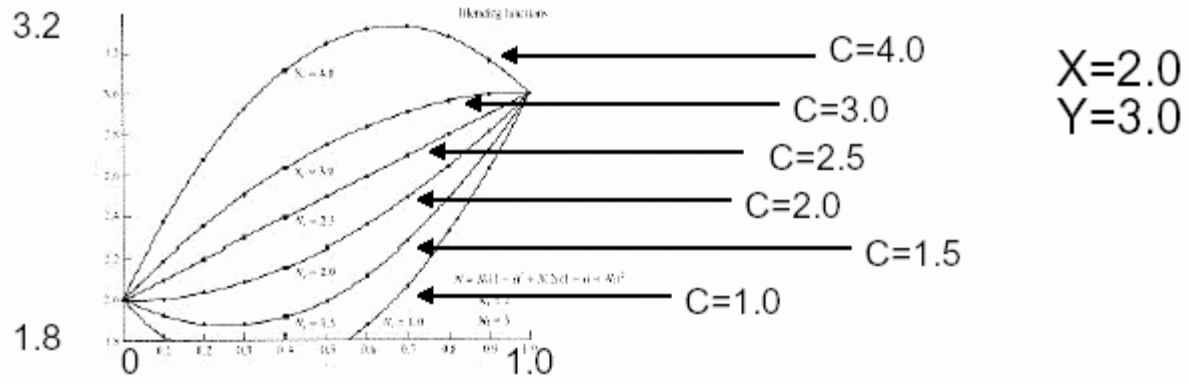
$$p(t) = (1-t)X + tY, t \in [0,1]$$

Quadratische Überblendung

$$p(t) = (1-t)^2 X + t^2 Y + 2t(1-t)$$

C: Parameter für Kurve

$$p(t) = (1-t)^2 X + t^2 Y + 2t(1-t)C$$



- Kubische Überblendung erfolgt analog mit Parametern C und D

$$p(t) = (1-t)^3 X + tY^3 + 3t(1-t)^2 C + 3t^2(1-t)D$$

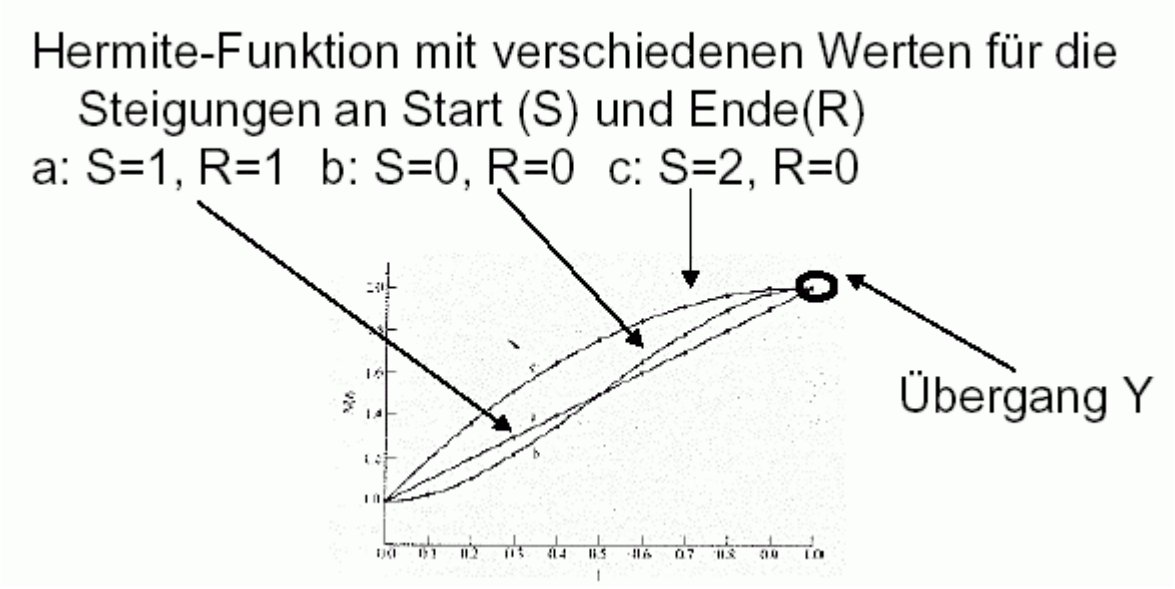
- Nur durch Auswahl des Parameters ist es schwer die Form der Kurve zu erahnen
- Grafische Schnittstelle ist daher unbedingt notwendig
- Start und Ende der Animation sind hier kritisch!

Hermite-Überblendung:

- Kontrolliere Start und Ende des Interpolationsvorgangs durch explizite Parameter S,R
- Benutze kubische Funktion für Überblendung
- Darstellung in Matrixform

$$p(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} * \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ S \\ R \end{bmatrix}$$

Beispiel:



weitere Ansätze:

- Parabolische Überblendung (parabolic blending; siehe Kap. 9)
- Bézier-Kurven
- B-Splines
- andere Spline-Varianten

Problem:
Bei Positions-Interpolation entspricht die *Geschwindigkeit* der gewünschten Positionsveränderung oft nicht der Parametrisierung der gewählten Spline-Kurve

⇒ notwendiger Zwischenschritt:
Bogenlängenparametrisierung

- wenn parametrische Kurve $Q(u)$ gegeben: bestimme Funktion, die pro Parameter-Intervall du die zurückgelegte Weglänge (Bogenlänge) $s = A(u)$ auf der Splinekurve bestimmt
- dann Reparametrisierung der Splinefunktion zu $Q(s)$

Reparametrisierung notwendig, da erst die Splinekurve bekannt sein muss, damit die Bogenlänge als Parameter für die Splinebeschreibung dienen kann

- Problem: Bogenlängenfunktion $A(u)$ kann i.allg. nicht analytisch bestimmt werden

⇒ numerische Integration

aber das ist aufwändig

⇒ Approximation durch akkumulierte Bogensehnen (Verfahren der "Vorwärtsdifferenzierung") mit Stützstellen, die möglichst gleichmäßig auf der Kurve verteilt liegen

Interpolation von Rotationswerten

in 2D einfach, es genügt die Interpolation des Winkels

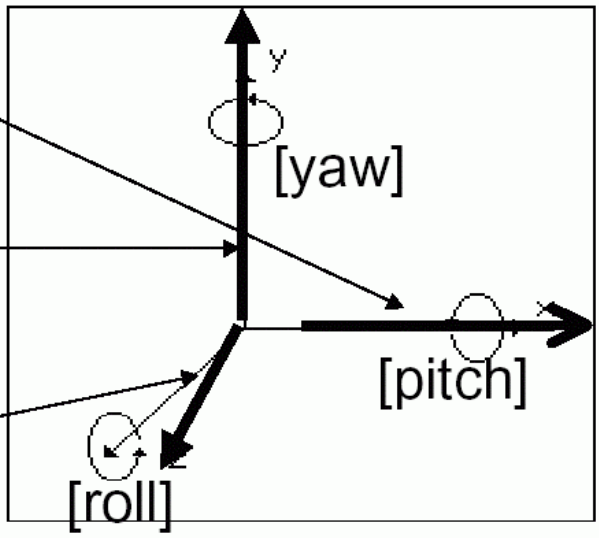
in 3D prinzipiell Kombination von Rotationen um 3 Achsen (x, y, z)
verschiedene Bezeichungsweisen der Rotationen um die Achsen:
X, Y, Z; pitch, yaw, roll (Navigation); H, L, U (Turtle-Geometry)
Orientierung der Achsen nach der Rechte-Hand-Regel

Darstellung in homogener Koordinatendarstellung

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(pitch) & -\sin(pitch) & 0 \\ 0 & \sin(pitch) & \cos(pitch) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(yaw) & 0 & \sin(yaw) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(yaw) & 0 & \cos(yaw) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(roll) & -\sin(roll) & 0 & 0 \\ \sin(roll) & \cos(roll) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Die entsprechenden Rotationswinkel um die 3 Grund-Achsen werden manchmal als "Euler-Winkel" bezeichnet
(Achtung: nicht für jede Bewegung eindeutig bestimmt!)

– es bietet sich an, diese für die Interpolation zu benutzen

Interpolation der Eulerwinkel

Eulerwinkelrotation

yaw: Rotation um Y-Achse
pitch: Rotation um X-Achse
roll: Rotation um Z-Achse

Für rechtshändiges Koordinatensystem:
positive Rotation ist, in Richtung des Ursprungs der betreffenden Achse betrachtet, stets gegen den Uhrzeigersinn

Folge von Rotationen ist nicht kommutativ

daher Festlegung, z. B. [Vince,92] roll->pitch->yaw :

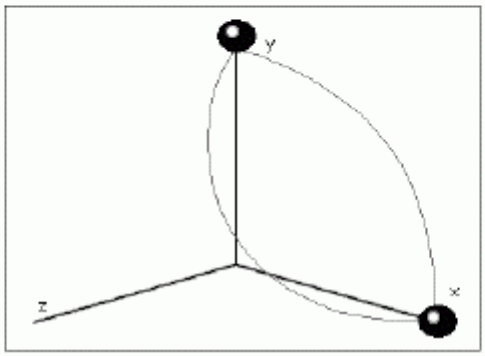
ergibt als Matrix:

[newPos] = [yaw][pitch][roll] [oldPos]

aber: diese Vorgehensweise ist oft zu abstrakt, nicht intuitiv

Probleme bei Eulerwinkel: Interpolation

- Weg ist nicht eindeutig vorhersagbar
- Nichtintuitive Bewegung bei Interpolation
- Schwierig vorhersagbar

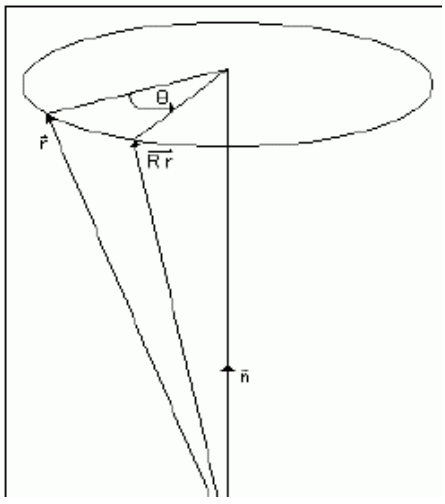


Alternative:
Charakterisierung der räumlichen Drehung durch Rotationsachse und Winkel
(so bei VRML und Java 3D)

Andere Darstellung aus Rotationsachse n und Winkel θ
Wie bei Eulerwinkel wird daraus Rotationsmatrix berechnet, also statt R([roll],[pitch],[yaw]) -> $R(\theta,n)$
Drehung eines Vektors r zur Position Rr um Winkel θ

Berechnung der Rotation
(vgl. Watt, S 359)

$$Rr = (\cos\theta)r + (1-\cos\theta)\, n(n\cdot r) + \sin(\theta)n\times r$$



- wenn Drehachse fest: Interpolation unproblematisch
- sonst erst Interpolation der Achsen, dann der Winkel
- auch hier kann es zu nichtintuitiven Ergebnissen kommen

weitere Alternative: *Quaternionen*
- Verallgemeinerung der komplexen Zahlen
  (hier in der Vorlesung nicht behandelt)
- können zur Charakterisierung von Rotationen in 3D heran-gezogen werden
- komponentenweise (lineare) Interpolation möglich

<u>Zusammenfassung zum parametrischen 3D-Keyframing</u>

Vorteile:
- universelles Animationsverfahren: anwendbar auf alle Modell-, Szenen- und Rendering-Parameter
- Parameter und Änderungsverhalten vollständig unter Benutzerkontrolle
- kombinierbar mit anderen Verfahren

Nachteile:
- nicht jeder Parameter eignet sich für eine Interpolation
- bei einigen Parametern zusätzliche Umformungen / Darstellungen nötig – Bsp. Rotationen (Euler-Winkel etc.); Farbspezifikationen (evtl. Wechsel des Farbmodells, um natürlichere Farbabstände zu erhalten)
- oft Schwierigkeit, verschiedene Parameter miteinander zu koordinieren: beachte Abhängigkeiten zwischen Parametern
- ggf. sehr viele Parameter zu kontrollieren: einige Hundert oder Tausend
- wünschenswert: Berücksichtigung von Einschränkungen, die sich z.B. durch physikalische Randbedingungen ergeben (dies ist zunächst nicht gewährleistet – sehr große Freiheiten!)

Constraints

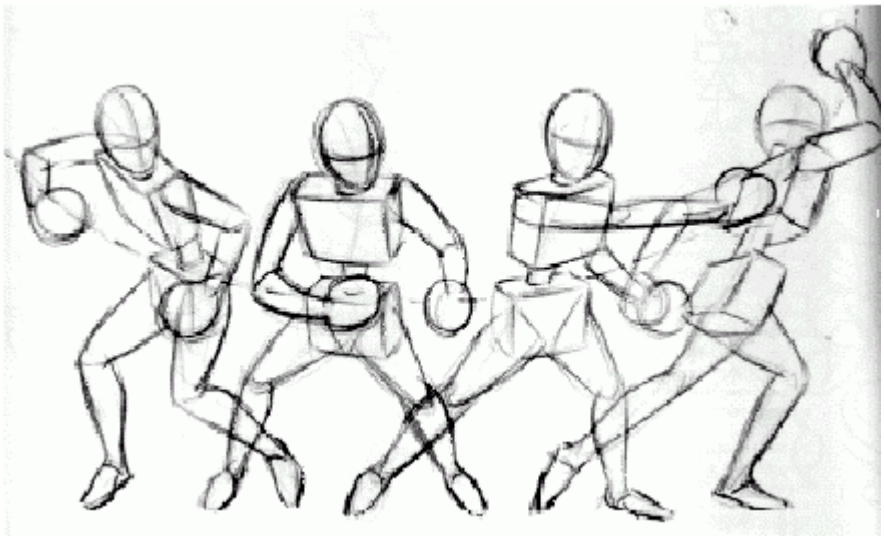
Universeller Ansatz zur besseren Handhabung vieler Parameter

- Constraints sind Bedingungen für Parameterwerte
- beschreiben Interaktionen und Randwerte
- einfache: Gleichheit, Max, Min... z.B.: Kamera soll immer auf Mittelpunkt eines bestimmten Objekts gerichtet sein
- komplexere: on-top, in-plane, on-path etc.
- u.U. nur iterativ / approximativ lösbar, oder gar nicht lösbar (Widersprüche)

Spezialfall:
mechanische Constraints

- Kollisionserkennung
- kinematische Modelle (Verbindung von Objekten an Gelenken)

Kinematik:
motiviert durch Animation von gegliederten Strukturen (*articulated structures*)



- Anwendung bei gelenkartig verbundenen Objekten, z.B. Tiermodellen, Menschmodellen
- Geometrisches Modell wird um “Skelett” erweitert: idealisiert eine

kinematische Kette

Vorwärtsrechnung

- Eingabe: Gelenkstellungen
- Ausgabe: Effektorstellung
- Berechnung der Endgliedstellung bei vorgegebenen Gelenkeinstellungen

Kinematik

Rückwärtsrechnung:

- Eingabe: Effektorstellung
- Ausgabe: Gelenkstellungen
- Berechnung der Gelenkstellungen bei vorgegebener Effektorstellung

Inverse Kinematik

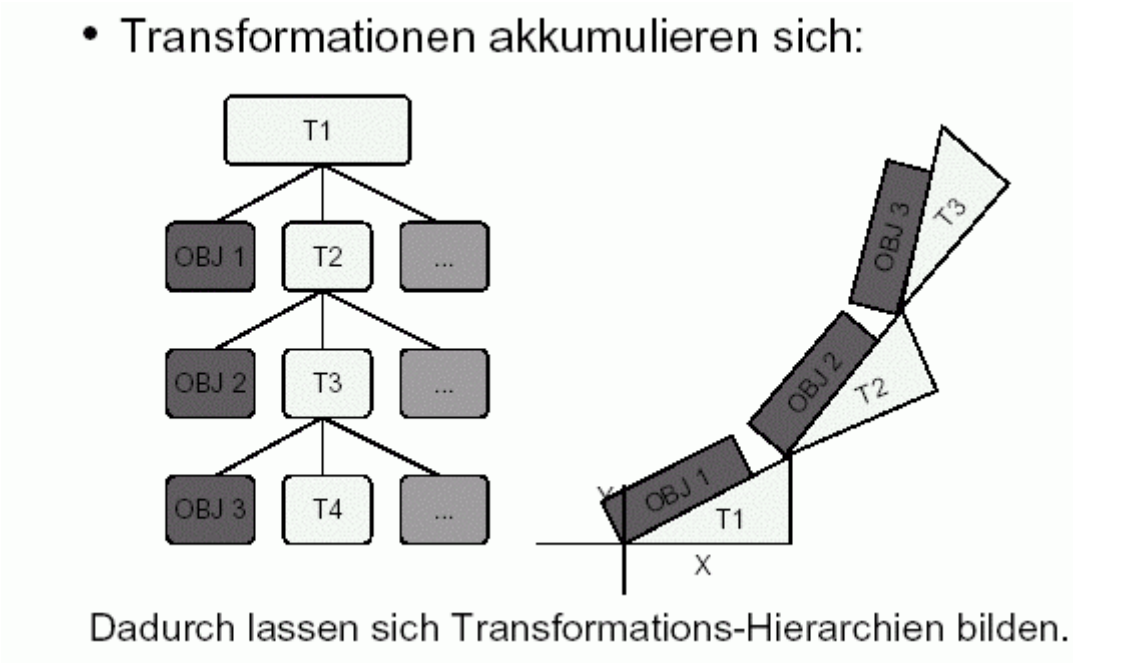
(Anwendung auch in der Robotik)

Vereinfachung durch Standardisierung nach Denavit / Hartenberg:

- es werden nur Schub- und Drehgelenke behandelt
- Beschreibung eines Gelenks durch konstante, konstruktionsbedingte Werte und durch Gelenkvariablen (aktuelle Stellung des Gelenks)

aus solchen Primitivgelenken lassen sich beliebige kinematische Ketten bilden

Vorwärtsrechnung: Produkt von Matrizen
Grundlage: Transformationen akkumulieren sich (vgl. Szenengraph!)

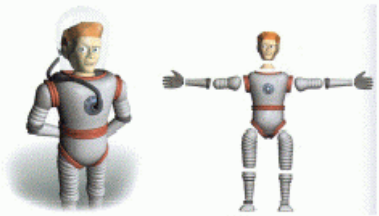


- mit diesem Ansatz mathematische Beschreibung größerer kinematischer Ketten möglich
- in der Praxis häufig gekoppelt mit "Skin models"
- "interessante" Ketten sind häufig unterbestimmt – weitere Constraints müssen eingeführt werden, um eindeutige Lösung zu gewährleisten
- Bewegungen trotzdem oft "unnatürlich"
- Problem, dass zunächst nur "Skelette" animiert werden: Übertragung auf 3D-Körper nichttrivial

"Skinning":
bei skelettierten Objekten müssen geeignete Maßnahmen ergriffen werden, um die "Haut" der Bewegung anzupassen.

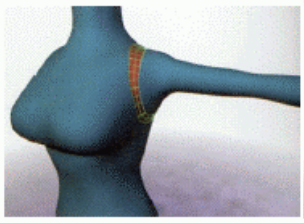
Methoden für segmentierte Objekte

- „versteckte“ Kugelgelenke (filler objects)



- Kleidung

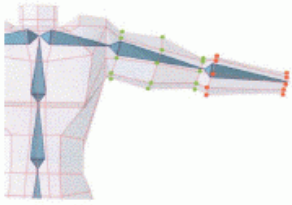
- Blend Objects

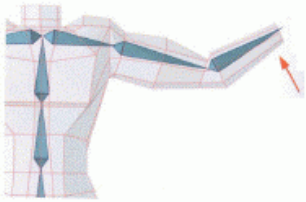


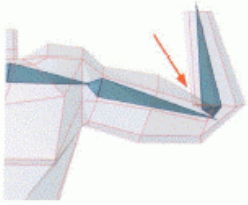
Skin surfaces:

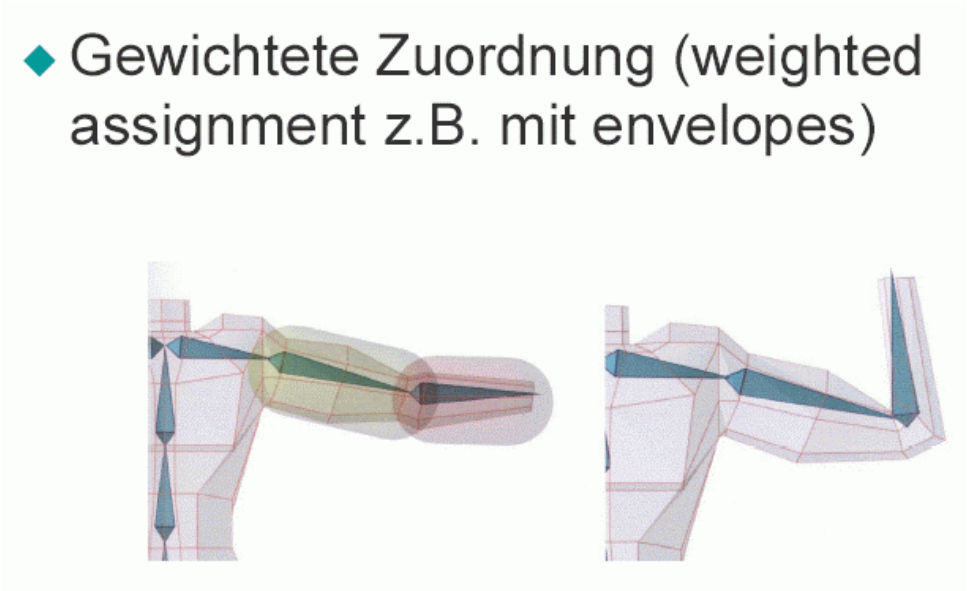
Die definierenden Eckpunkte sind relativ zu den Gelenken (Verbindungen) definiert:

- Direkte Zuordnung (direct assignment)









Kinematik vs. Dynamik

Kinematik:

- Bewegungslehre ohne Berücksichtigung der auftretenden Kräfte
- betrachtet Position, Geschwindigkeit, Beschleunigung...

Dynamik:

- Bewegungslehre unter Berücksichtigung von Kräften
- betrachtet Massen, Trägheitsmomente, Kräfte, Impuls, Energie...

Animationen mit Dynamik-Simulation:

- z.B. Feder-Masse-Dämpfer-Systeme
- häufig auf Felddynamik erweitert: Gravitation, Ladung...
- Beeinflussung vieler Objekte gleichzeitig: Partikel
- Spezialfall der allgemeinen Simulation
- oft schwierig zu benutzen, da visuelle Effekte schwer vorhersehbar!

Simulation

- basiert auf Repräsentation von Aspekten der realen Welt in einem abstrakten Modell
- Kinematik und Dynamik sind nur Spezialfälle, weitere Einsatzgebiete z.B.: Strömungslehre, Thermodynamik, Radiometrie (Radiosity-Ansatz!), Elektrodynamik, biologisches Wachstum, chemische Reaktionen...
- potenziell sehr viele verschiedene Basis-Modelle, Sprachen und Methoden verfügbar, z.B. Petrinetze, Simula, HLA...
- methodisch oft eher Experimentieren als Konstruieren oder Gestalten!

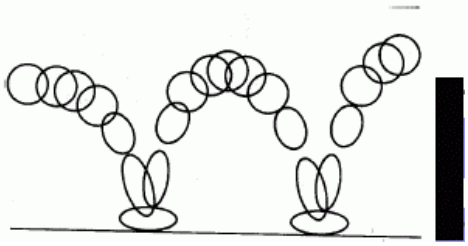
- sehr leistungsfähige Simulatoren vorhanden
- Simulation erlaubt Kontrolle von Animation auf höherem (abstrakterem) Niveau als Geometrie und visuelle Merkmale
- Grundlage der "interaktiven" Animation
- Verknüpfung mit anderen Animationsmethoden möglich, aber selten implementiert

Erweiterungen von Dynamik in der Animation:

Soft-object-Animation

Motivation und Bedeutung

- Generell: weitere Freiheiten für den Animator
- Ein wichtiges Prinzip aus der traditionellen (Walt Disney-) Animation:
  - **Squasch and Stretch**
  - **Stretch and Squeeze**



⇒ die traditionellen Grenzen zwischen Modellierung und Animation verwischen!

bei der Soft-object-Animation sind immer 2 Prozesse beteiligt:

- Methode zur Objektdeformation (shape change) (polygonale Modelle: Verschieben der Eckpunkte; parametrische Modelle: Verändern der Kontrollpunkte)
- Methode zur Animation des Deformationsprozesses

Deformation polygonaler Repräsentationen:

nur die Positionen der Eckpunkte werden verändert, Topologie bleibt erhalten
Probleme, wenn

- sich nach der Transformation Linien überschneiden (lässt sich nicht mehr rendern)
- für die entstehenden Krümmungen zu wenige Eckpunkte definiert sind: "3D-Aliasing"; Lösung ggf. durch Subdivision (Erhöhung der Auflösung des Modells)

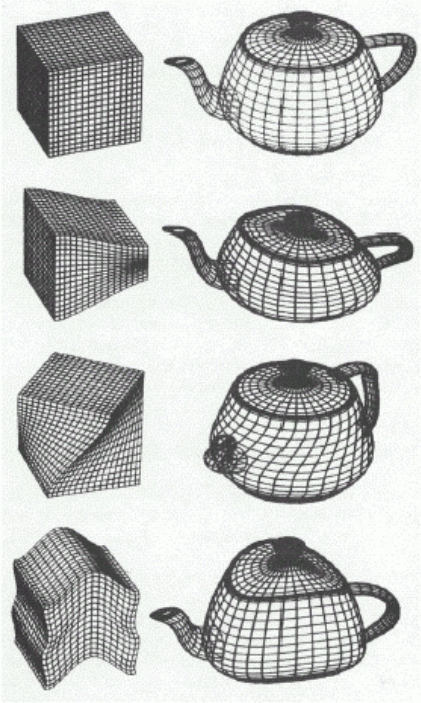
Deformation parametrischer Repräsentationen (z.B. bikubische Flächen):

- Vorteil: trotz Verschiebung der Kontrollpunkte entstehen immer glatte Flächen $\Rightarrow$ keine Rendering-Probleme
- nur wenige Kontrollpunkte (verglichen mit polygonalen Modellen) – ggf. Probleme, die gewünschten Veränderungen zu erzielen
- evtl. auch hier Verfeinerungen nötig
  (Bézier: de Casteljau-Algorithmus;
  B-Splines: Oslo-Algorithmus)

Nichtlineare globale Deformation ("3D-Warping")
nach Barr 1984

- Beschreibung mit Hilfe von Transformationsmatrizen
- 3 Haupttransformationen: Tapering (verjüngen, anspitzen), Twisting (verdrehen), Bending (biegen, krümmen).

Nichtlineare globale Deformation Tapering



$(X,Y,Z) = F(x,y,z)$

(x,y,z) : Eckpunkt im nichtdeformierten Objekt

(X,Y,Z) : Eckpunkt im deformierten Objekt

Beispiel : Skalierung

Nichtdeformiert : $(X,Y,Z) = (s_x x, s_y y, s_z z)$

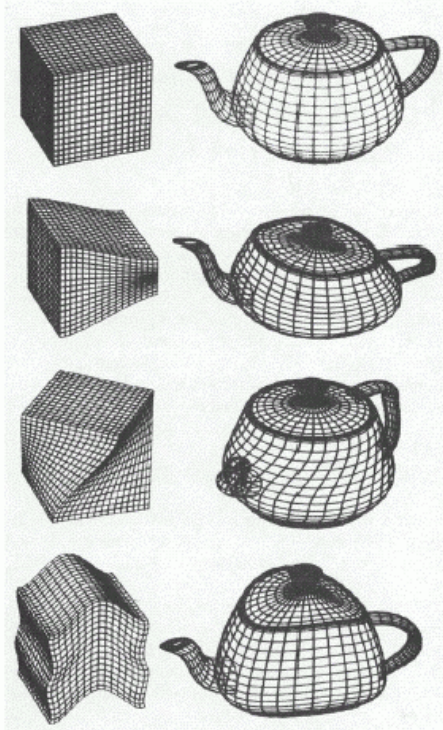
Tapering entlang der z - Achse :

$(X,Y,Z) = (rx, ry, z)$ mit

$r = f(z)$ als lineares oder

nichtlineares Tapering - Profil

Nichtlineare globale Deformation Twisting



basiert auf der Rotation,

z.B. um die z - Achse

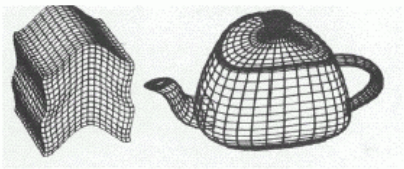
$$(X,Y,Z) = (x\cos\Theta - y\sin\Theta,\; x\sin\Theta + y\cos\Theta, z)$$

$$\Theta = f(z)$$

$f(z)$ repräsentiert die Verdrehungs - rate pro z - Einheit

Nichtlineare globale Deformation
Bending

Ein Bending ist definiert als eine zusammengesetzte Transformation (Rotation und Translation) in einer Region: z.B. entlang der y-Achse:



$y_{min} \le y \le y_{max}$

Krümmung : k^{-1}

Zentrum der Krümmung : y_0

Verdrehungswinkel : $\Theta = k(y'-y_0)$ mit

$$y' = \begin{cases} y_{min} & \text{für } y \le y_{min} \\ y & \text{für } y_{min} < y < y_{max} \\ y_{max} & \text{für } y \ge y_{max} \end{cases}$$

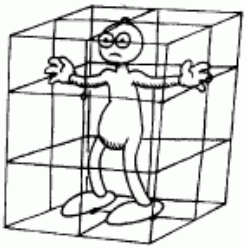
Die Deformationstransformation ist dann

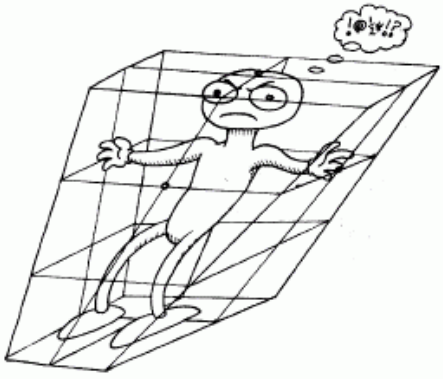
$X = x$

$$Y = \begin{cases} -\sin\Theta(z-k^{-1}) + y_0 & y_{min} \le y \le y_{max} \\ -\sin\Theta(z-k^{-1}) + y_0 + \cos\Theta(y-y_{min}) & y < y_{min} \\ -\sin\Theta(z-k^{-1}) + y_0 + \cos\Theta(y-y_{max}) & y > y_{max} \end{cases}$$

$$Z = \begin{cases} \cos\Theta(z-k^{-1}) + k^{-1} & y_{min} \le y \le y_{max} \\ \cos\Theta(z-k^{-1}) + k^{-1} + \sin\Theta(y-y_{min}) & y < y_{min} \\ \cos\Theta(z-k^{-1}) + k^{-1} + \sin\Theta(y-y_{max}) & y > y_{max} \end{cases}$$

Freiform-Deformation (Gitter-Deformation) nach [Sederburg 86]





Idee: 1. Man faßt ein Objekt in ein umgebendes Gitter (einen Raum) ein.

2. Man deformiert das Gitter (den Raum) durch Modeling-Transformationen

3. Deformationen des Gitters (des Raumes) werden auf das Objekt übertragen.

Modellvorstellung z.B. : Objekte sind durch Federn in dem Gitter gehalten

benutzbar für polygonale und parametrische Objekte:
Eckpunkte ←→ Kontrollpunkte

(vgl. 2D-Morphing)

Beispiel:

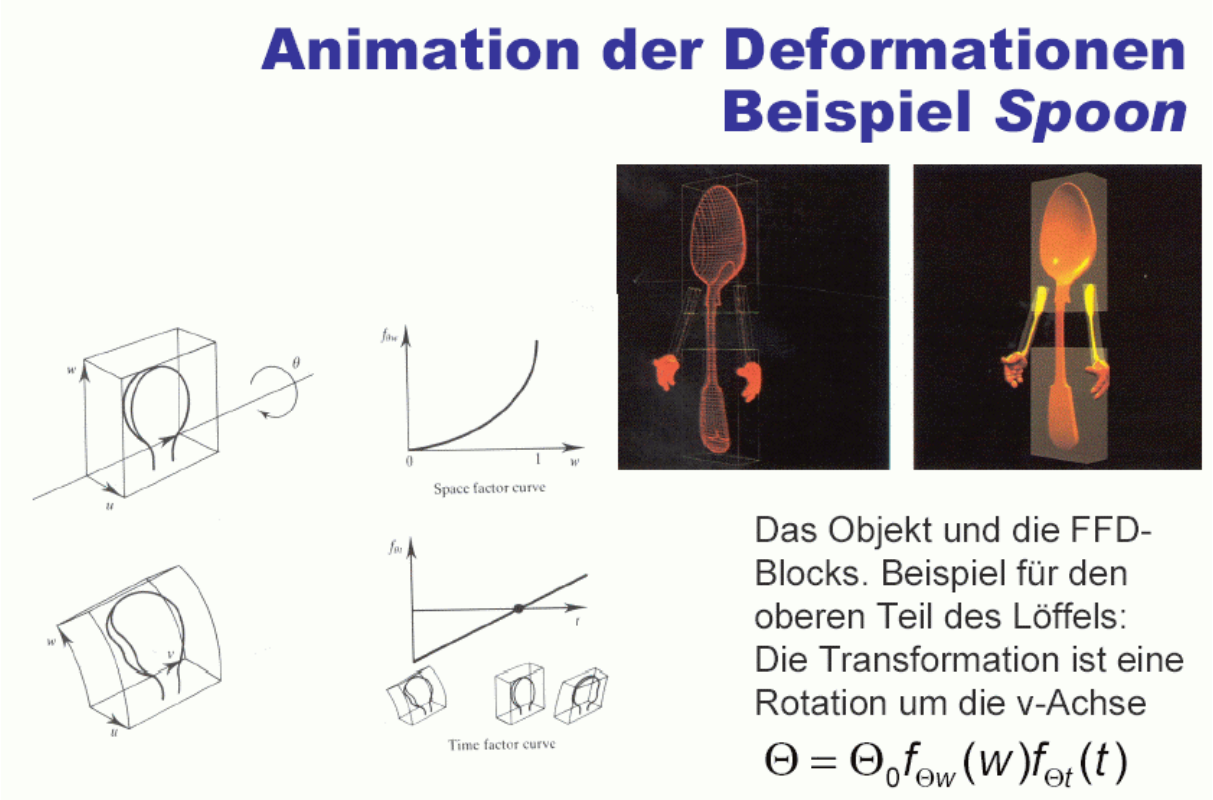
Animation der Deformationen:

Transformationen sind Funktionen des Ortes (sog. Faktorkurven) – diese werden um eine zeitliche Komponente erweitert

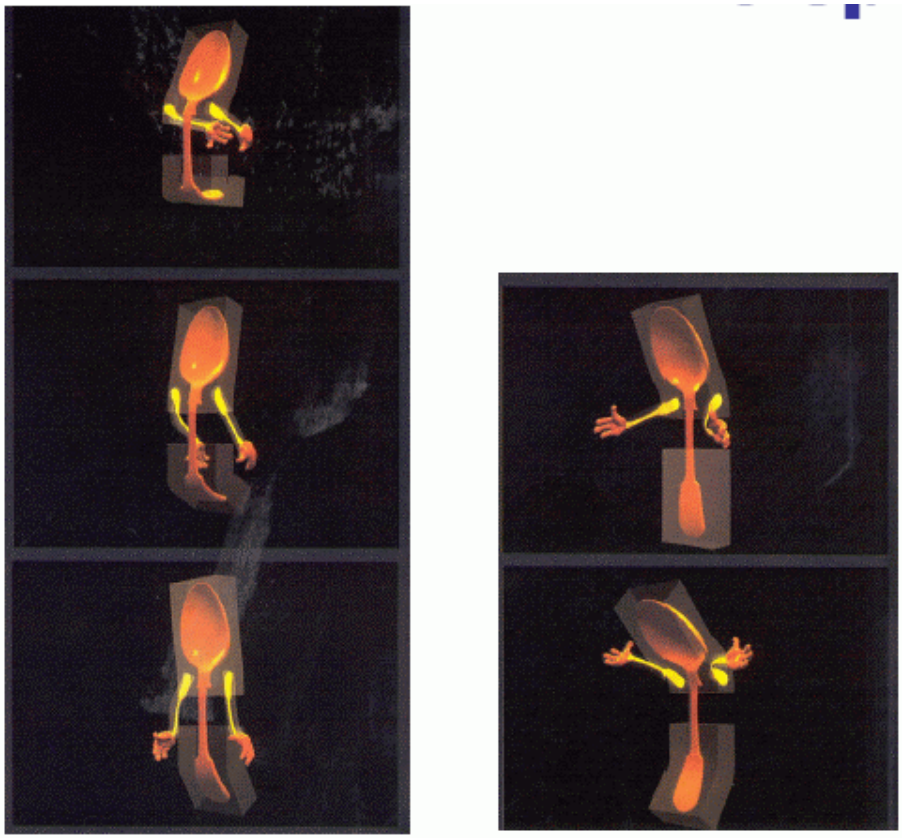
Der Benutzer zerlegt die Deformation in 2 Komponenten:
einen Satz von Transformationen, die den Gesamtumfang der Deformation beschreiben, zusammen mit den entsprechenden Parametern,
einen Satz von Faktorkurven in Raum und Zeit, die die Veränderungen der Parameter (wo und wann) beschreiben

Faktorkurven: oft als Bézierkurven spezifiziert (vgl. Kinegraph)

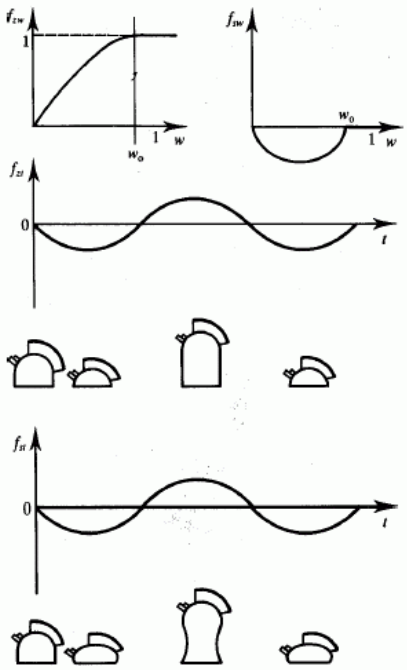
Beispiele:



(FFD = Freiformdeformation, 3D-Warping)



Animation der Deformationen
Beispiel *Kettle*



(Parameterraum (u,v,w) entspricht der Bounding-Box

Die Transformationen sind:

- eine vertikale Translation und
- eine Skalierung in der (x,y)-Ebene durch Parameter s.

Die Deformationen erfolgen im wesentlichen im Kesselbauch ($w<w_0$): Ausguß und Griff bleiben weitgehend unverändert. Es wird versucht, das Objektvolumen annähernd konstant zu halten (= traditionelles Prinzip!)

$$z = z_0 f_{zw}(w) f_{zt}(t)$$

$$s = 1 + s_0 f_{sw}(w) f_{st}(t)$$