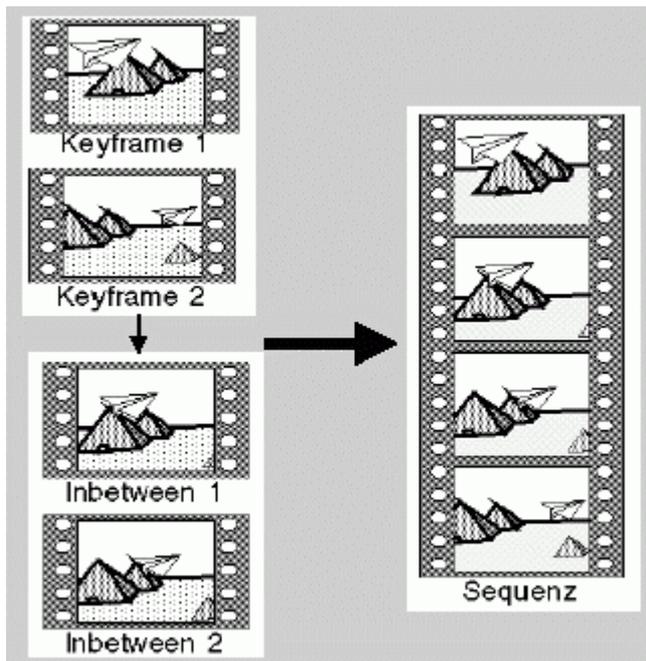


## Keyframe-Animation

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 Postproduktion 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**:  $S(x,y) \rightarrow T(x',y')$

**Warping**:  $S=T$  und  $(x',y') = \text{warp}(x,y)$  und warp ist der benutzte Warpingalgorithmus (z.B. two spline mesh Verfahren, feature based morphing)

**Farbtransformation**:  $S(x,y) \rightarrow T(x,y)$ , z. B. *cross-dissolve*: Pixel (x,y) hat Farbe I zum Zeitpunkt

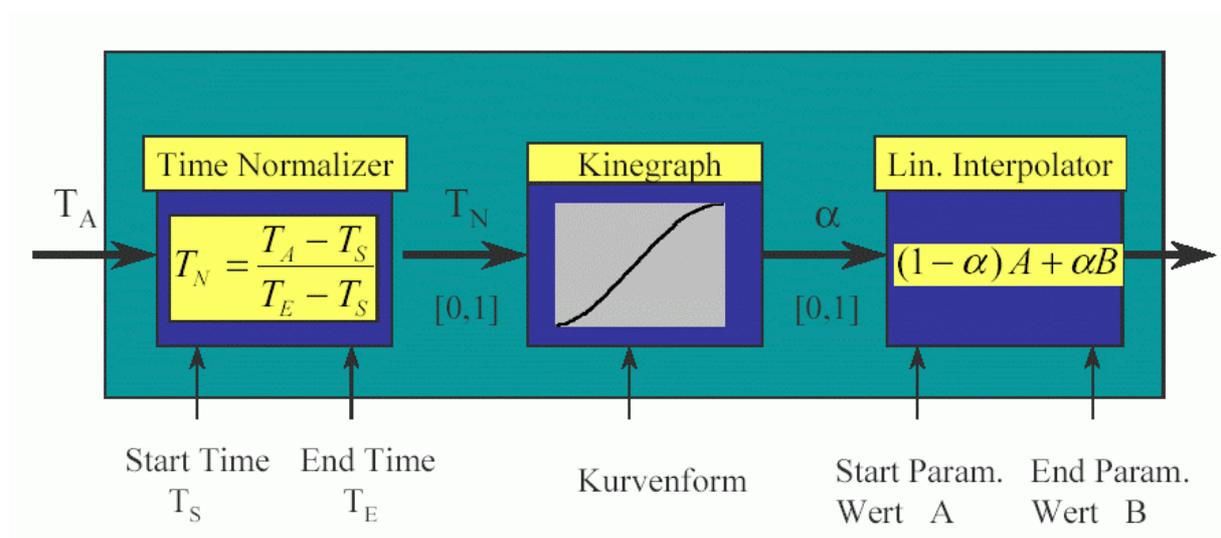
$$t, t_{\min} \leq t \leq t_{\max}$$

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

die Gewichtungsfunktionen für S(x, y) und T(x, y) können verändert werden, z.B. langsamere Veränderung am Anfang, dann immer schneller!

"Kinograph" legt den Verlauf der Gewichtungsfunktion 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:  $\alpha = \sin\left(T_N \cdot \frac{\pi}{2}\right)$

- ◆ 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

$$a = m + n - 2$$

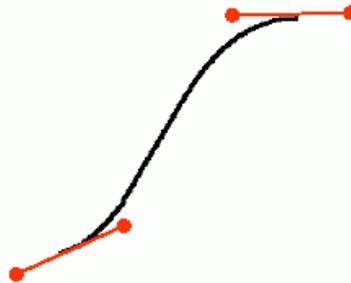
$$b = -2m - n + 3$$

$$c = m$$

$$d = 0$$

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 Warming (gesteuerte Form-Interpolation):

- Two-spline mesh warping
- feature-based warping

# Konstruktion einer Morphing-Sequenz

1) Warping Quelle zu Zielform



2) Warping Ziel zu Quellform



Cross Dissolve S1 Tn

Cross Dissolve Sn T1



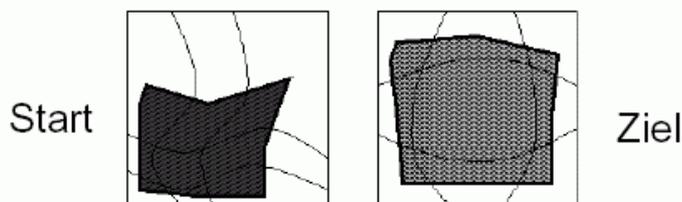
25%

50%

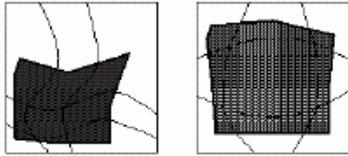
75%

## Two-Spline Mesh Warping

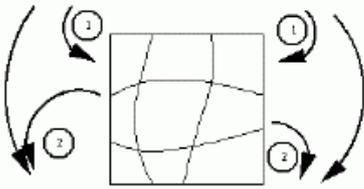
- Entwickelt von Douglas Smythe bei ILM (Willow, Indiana Jones IV, Abyss)
- Benutzer spezifiziert Punktgitter durch das Splines gelegt werden (spline mesh)



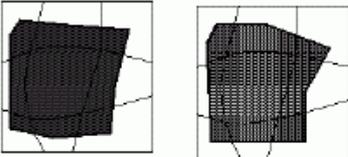
- Übergang von Netz 1 zu Netz 2 durch Berechnung von **Inbetweens**
- Interpolation der Kontrollpunkte
- 2-Pass Verfahren



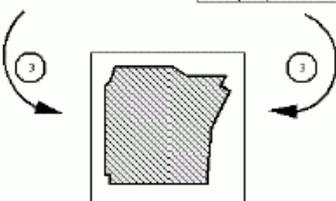
Punkte an den Rändern  
bleiben konstant  
**Wie erhält man Inbetweens?**



1) Berechne Zwischennetz  
2) Warpe Start- und Zielbild in  
das Zwischennetz

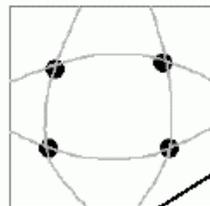
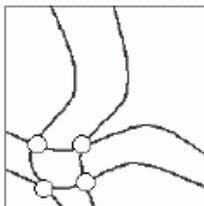


3) Cross-Dissolve der beiden  
Bilder



G1

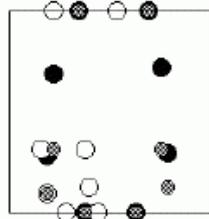
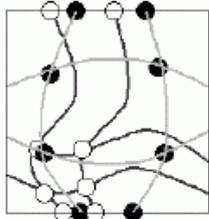
G2



1) Verschiebe Kontrollpunkte:  
X-Position von G2, Y-Position  
von G1

2) Berechne vertikale Splines  
durch neue Punkte

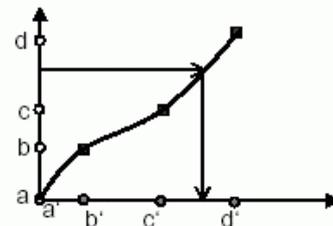
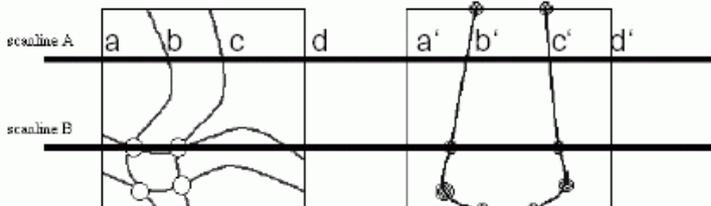
3) Basis für ersten Durchlauf  
einer Scanlinie



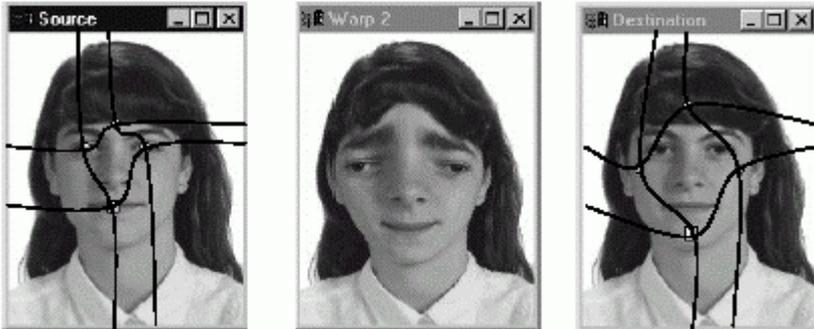
4) Berechne X-Abschnitte der  
Kontrollpunkte in G1 und  
G2 (a,b,c,d)

5) Berechne Mapping für  
genau diese Scanlinie

6) Transformiere Pixel



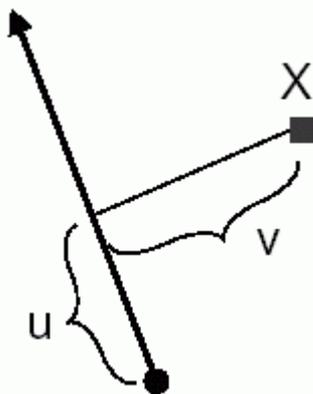
Erster Pass: Warping in X-Richtung  
Zweiter Pass: Analog in Y-Richtung



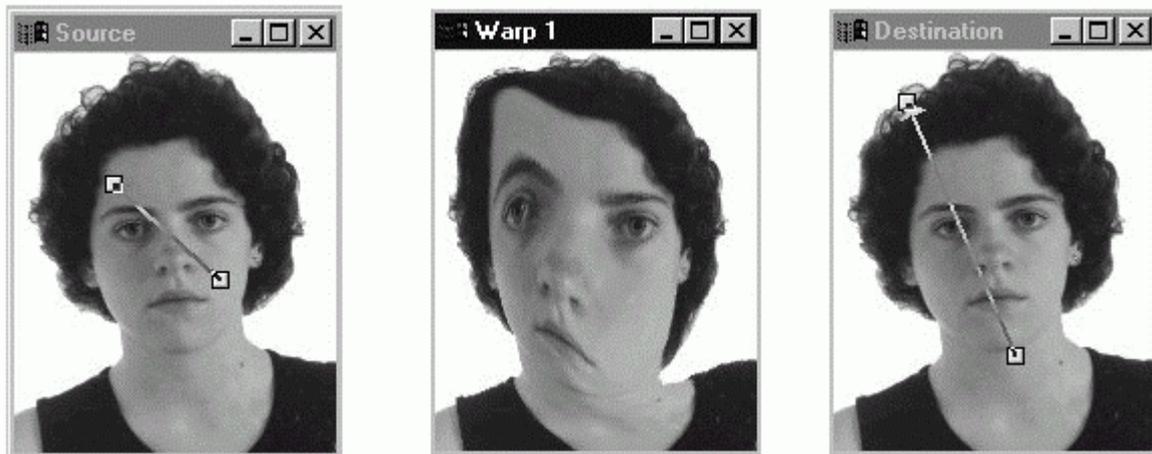
Beispiel

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:

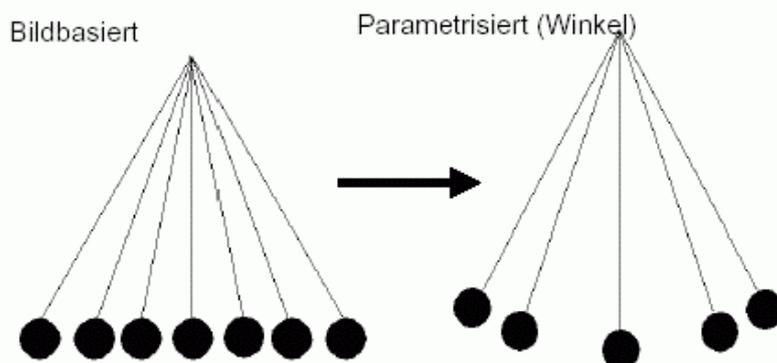


### 3D-parametrisches Keyframing

- 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 Modellierungsverfahren
- 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  $P, X, Y \in \mathbb{R}^m$

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

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

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

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

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

# 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  $T_{end}$  (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")

$$p(t) = (1-t)X + tY \text{ mit } t = \sin \alpha, 0^\circ \leq \alpha \leq 180^\circ$$

weitere Techniken:

Verwendung von Polynomen (quadratisches, kubisches Überblenden)

## Zusammenfassung zum parametrischen 3D-Keyframing

### *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

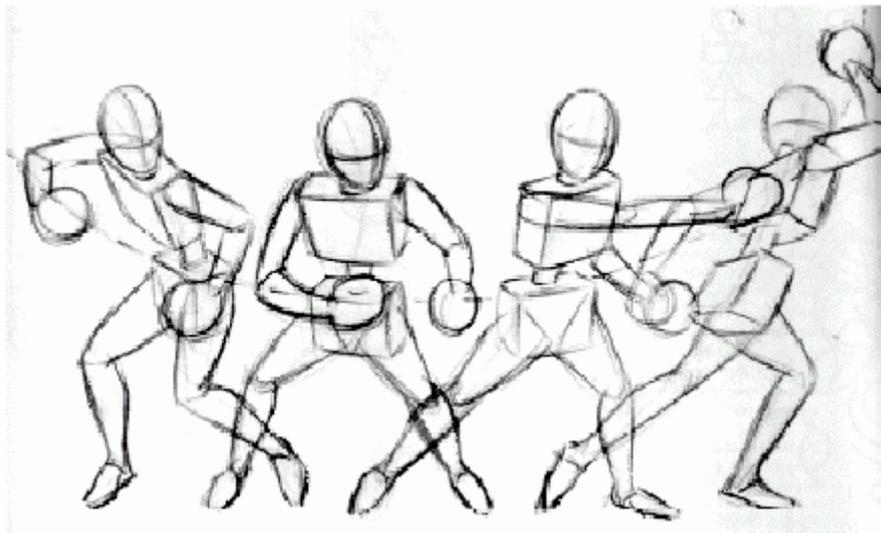
- Constraints sind Bedingungen für Parameterwerte
- beschreiben Interaktionen und Randwerte
- einfache: Gleichheit, Max, Min... z.B.: Kamera soll immer auf Mittelpunkt eines bestimmten Objekts gerichtet sein
- komplexere: on-top, in-plane, on-path etc.
- u.U. nur iterativ / approximativ lösbar, oder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

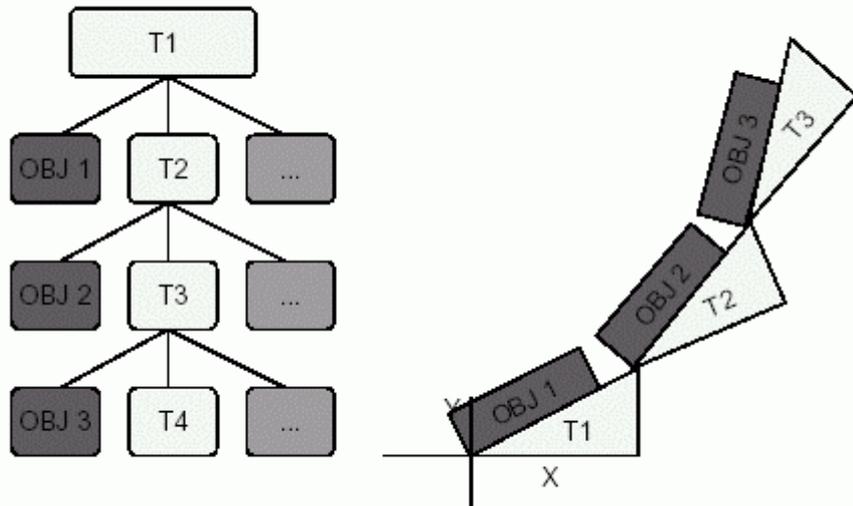
- es werden nur Schub- und Drehgelenke behandelt
- Beschreibung eines Gelenks durch konstante,  
konstruktionsbedingte Werte und durch Gelenkvariablen  
(aktuelle Stellung des Gelenks)

aus solchen Primitivgelenken lassen sich beliebige  
kinematische Ketten bilden

Vorwärtsrechnung: Produkt von Matrizen

Grundlage: Transformationen akkumulieren sich (vgl.  
Szenengraph!)

- Transformationen akkumulieren sich:



Dadurch lassen sich Transformations-Hierarchien bilden.

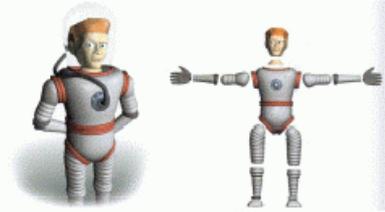
- mit diesem Ansatz mathematische Beschreibung größerer kinematischer Ketten möglich
- in der Praxis häufig gekoppelt mit "Skin models"
- "interessante" Ketten sind häufig unterbestimmt – weitere Constraints müssen eingeführt werden, um eindeutige Lösung zu gewährleisten
- Bewegungen trotzdem oft "unnatürlich"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

"Skinning":

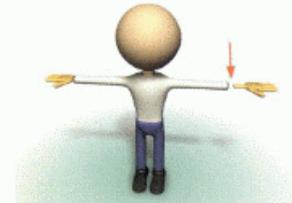
bei skelettierten Objekten müssen geeignete Maßnahmen ergriffen werden, um die "Haut" der Bewegung anzupassen.

# Methoden für segmentierte Objekte

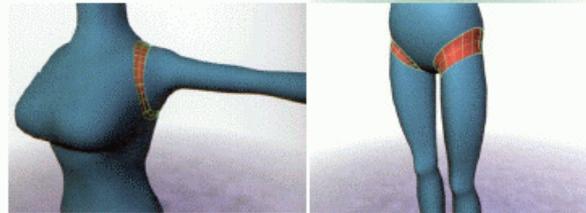
- „versteckte“ Kugelgelenke (filler objects)



- Kleidung



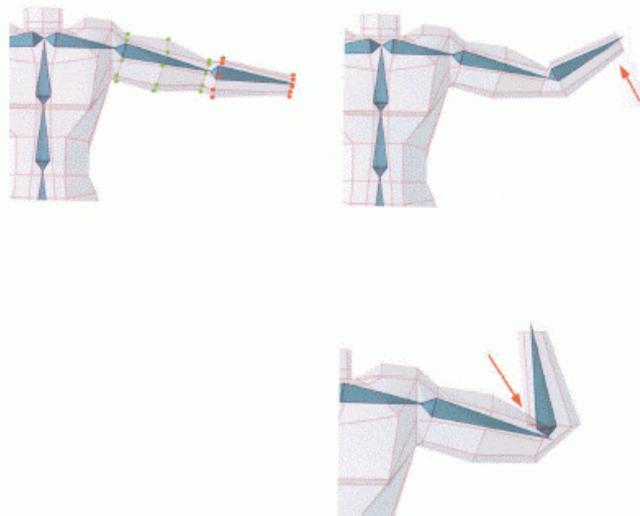
- Blend Objects



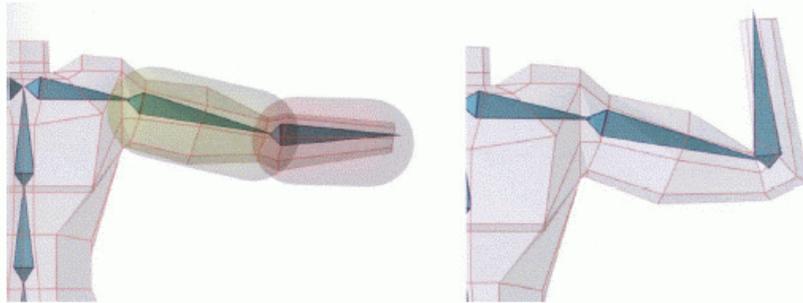
Skin surfaces:

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

- ◆ Direkte Zuordnung (direct assignment)



- ◆ Gewichtete Zuordnung (weighted assignment z.B. mit envelopes)



### *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

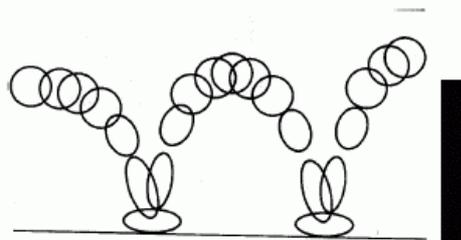
Erweiterungen von Dynamik in der Animation:

### *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):

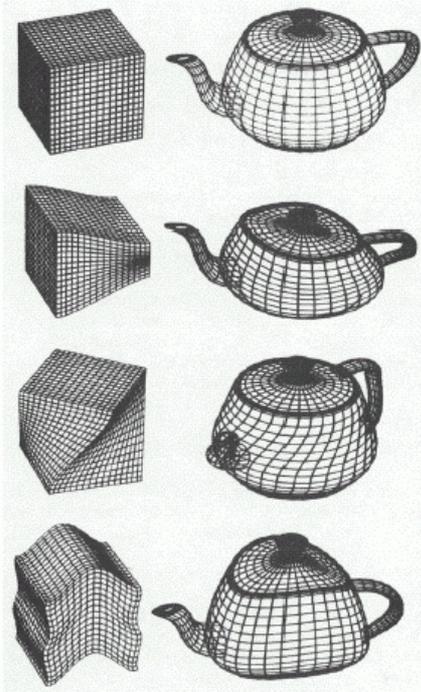
- Vorteil: trotz Verschiebung der Kontrollpunkte entstehen immer glatte Flächen  $\Rightarrow$  keine Rendering-Probleme
- nur wenige Kontrollpunkte (verglichen mit polygonalen Modellen) – ggf. Probleme, die gewünschten Veränderungen zu erzielen
- evtl. auch hier Verfeinerungen nötig (Bézier: de Casteljau-Algorithmus; 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:  $(X, Y, Z) = (s_x x, s_y y, s_z z)$

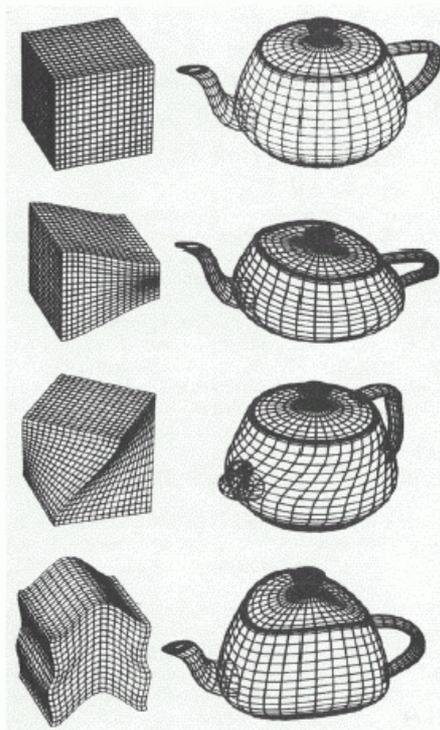
Tapering entlang der z - Achse:

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

$r = f(z)$  als lineares oder

nichtlineares Tapering - Profil

## Nichtlineare globale Deformation Twisting



basiert auf der Rotation,

z.B. um die z - Achse

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

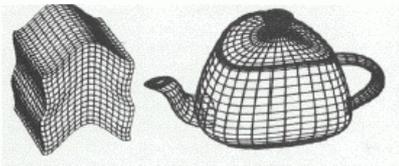
$$x \sin \Theta + y \cos \Theta, z)$$

$$\Theta = f(z)$$

$f(z)$  repräsentiert die Verdrehungsrate 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:



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

Krümmung :  $k^{-1}$

Zentrum der Krümmung :  $y_0$

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

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

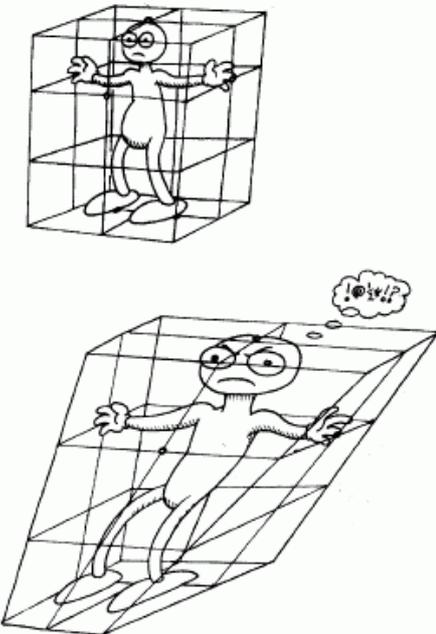
Die Deformationstransformation ist dann

$$X = x$$

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

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

## 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.

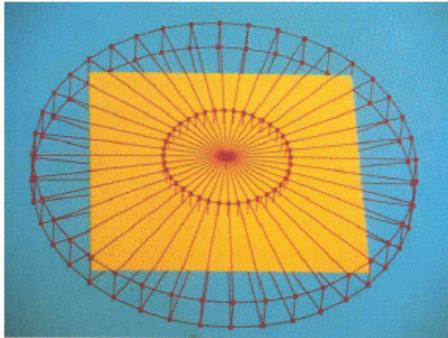
Modellvorstellung z.B. : Objekte sind durch Federn in dem Gitter gehalten

benutzbar für polygonale und parametrische Objekte:  
Eckpunkte  $\leftrightarrow$  Kontrollpunkte

(vgl. 2D-Morphing)

Beispiel:

## Erweiterte Freiform-Deformation Beispiele nach S. Coquillart



Animation der Deformationen:

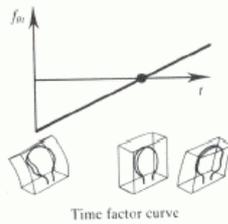
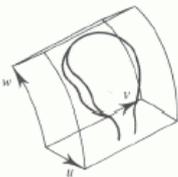
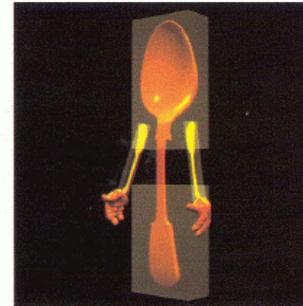
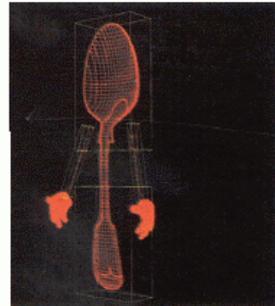
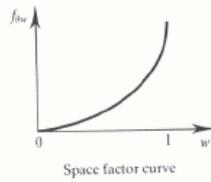
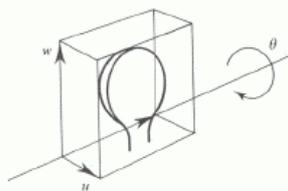
Transformationen sind Funktionen des Ortes (sog. Faktorkurven) – diese werden um eine zeitliche Komponente erweitert

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:

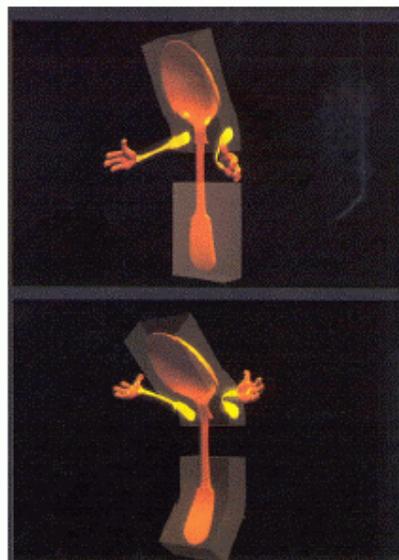
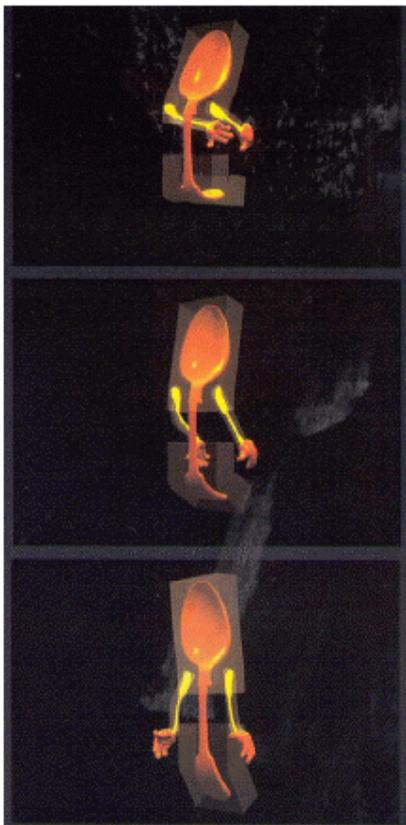
## Animation der Deformationen Beispiel *Spoon*



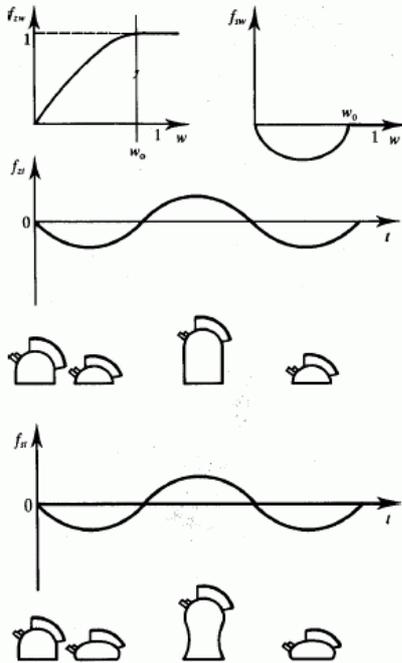
Das Objekt und die FFD-Blocks. Beispiel für den oberen Teil des Löffels:  
Die Transformation ist eine Rotation um die v-Achse

$$\Theta = \Theta_0 f_{\Theta w}(w) f_{\Theta t}(t)$$

(FFD = Freiformdeformation, 3D-Warping)



# Animation der Deformationen Beispiel Kettle



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

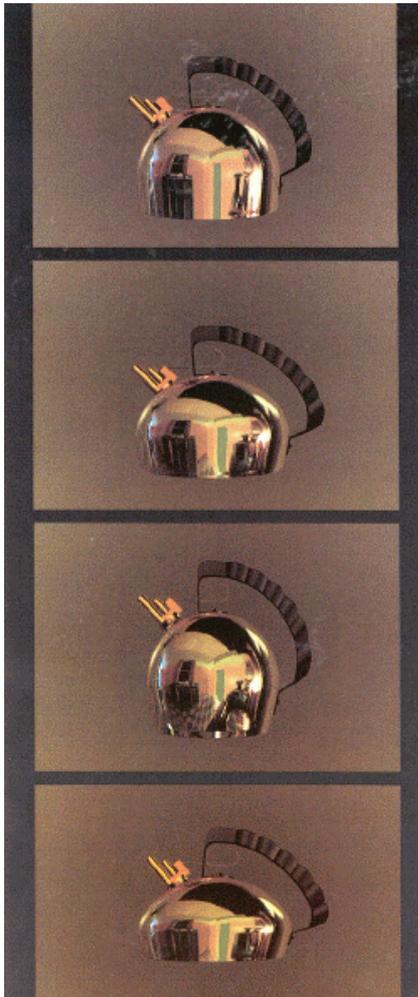
Die Transformationen sind:

- 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!)

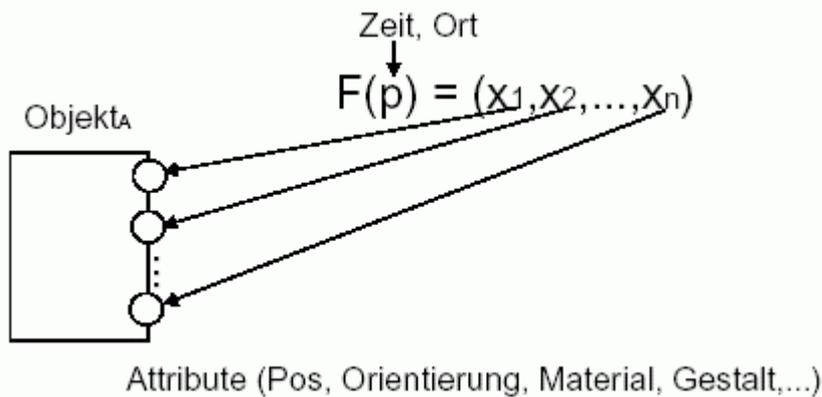
$$z = z_0 f_{zw}(w) f_{zt}(t)$$

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



## Prozedurale Animation

Idee: Modifizieren der Objekte (und Erzeugen, Löschen) mittels algorithmischer Vorschriften  
(vgl. prozedurale Modellierung, Kap. 10)



Art des verwendeten Algorithmus: sehr unterschiedliche Ansätze

- stochastische Animation (Partikelsysteme)
- verhaltensbasierte Animation (Fische, Vögel, Insekten)
- wissensbasierte Ansätze, z.B. genetische Algorithmen

typisch: emergentes globales Verhalten durch lokale Regeln

Anwendungen:

Partikelsysteme

Pflanzen, Feuer, Wassertropfen

verhaltensbasierte Ansätze

Schwärme, agentenorientierte Modelle, Artificial Life

Modellierung von Strukturen

Textilien, elastische Oberflächen

*Partikelsysteme*

(vgl. Kap. 10)

Partikel: größere Anzahl an betrachteten Einheiten mit Attributen, die Aussehen festlegen  
oft grafische Primitive wie Punkte, Linien, einfache 3D-Körper

- stochastische Komponente: Kontrolleinheit, die die Partikel beeinflusst
- nichtdeterministische Modifikation der Attribute (z.B. Farbe, Geschwindigkeit), meist innerhalb vordefinierter Grenzen
- Verwendung zuerst von Reeves in "Star Trek II" zur Animation der Entwicklung eines Planeten

5 Schritte für die Berechnung eines Frames:

- (1) erzeuge neue Partikel im System
- (2) Zuordnung von initialen Attributwerten für jedes neue Partikel
- (3) jedes Partikel, das seine maximale Lebensdauer erreicht hat, wird gelöscht
- (4) aktuelle Partikel werden gemäß der Berechnungsvorschrift bewegt
- (5) Alle Partikel innerhalb vordefinierter Grenzen werden gerendert

## Verhaltensbasierte Animation

Flocks, Herds, Schools  
basiert auf Partikelansatz,  
aber:

*„Each boid is an entire polygonal object rather than a graphical primitive  
Each boid has a local coordinate system.  
There are a fixed number of boids - they are not created or destroyed.  
Traditional rendering methods can be used because there are a small number of boids.  
Boids behavior is dependent on external as well as internal state. In other words, a boid reacts to what other boids are doing around it. “*



Boids (Bird objects)

Idee: Durch einfache lokale Regeln komplexes globales Verhalten

C. W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model",  
Computer Graphics, vol. 21, no. 4, pp 25-34, 1987.

# Regeln für Boids

## 3 Basisprinzipien für Boids:

- (1) *Kollisionsvermeidung*: Ein Boid versucht, Kollisionen zu vermeiden
- (2) *Geschwindigkeit annähern*: Ein Boid versucht, seine Geschwindigkeit der seiner Nachbarn anzupassen
- (3) *Schwarmzentrierung*: Ein Boid versucht, sich stets in der Nähe von anderen Boids aufzuhalten.

Kollisionsvermeidung:

mit anderen Schwarmobjekten und mit "fremden" Szenenobjekten – gewisser Mindestabstand wird während der Bewegung gewährleistet

Strategien zur Kollisionsvermeidung (Reynolds 1988):

- wegsteuern von Oberflächen (virtuelles Kraftfeld benutzen), Weg durch Öffnung suchen oder an Oberfläche entlang fliegen
- von Objekt-Mittelpunkten wegsteuern
- virtuelle Fühler ausstrecken, um mögliche Oberflächen direkt auf Kollisionskurs zu entdecken
- Ansteuern der in die eigene Sichtebene projizierten Umrisskanten (bzw. mit sicherem Abstand daran vorbei)

Schwarmverhalten (flock centering):

- jedes Boid will beim Schwarm bleiben
- mittlere Boids bleiben bei ihrer Richtung, äußere drängen nach innen
- wichtig: Wahrnehmungsbereich der Boids (endliche Distanz)
- keine Information über globalen Schwarm-Mittelpunkt,
- keine Gefolgschaft eines bestimmten "Führers"!

Navigation und Geschwindigkeit:

- Boid hat nur kleine Zahl von "wahrnehmbaren" Nachbarn, die zur Ermittlung der Durchschnittsgeschwindigkeit benutzt werden, an die das Boid sich anpasst
- Navigationsmodul kombiniert die 3 low-level Steuerprinzipien: Kollisionsvermeidung > Geschwindigkeitsannäherung >

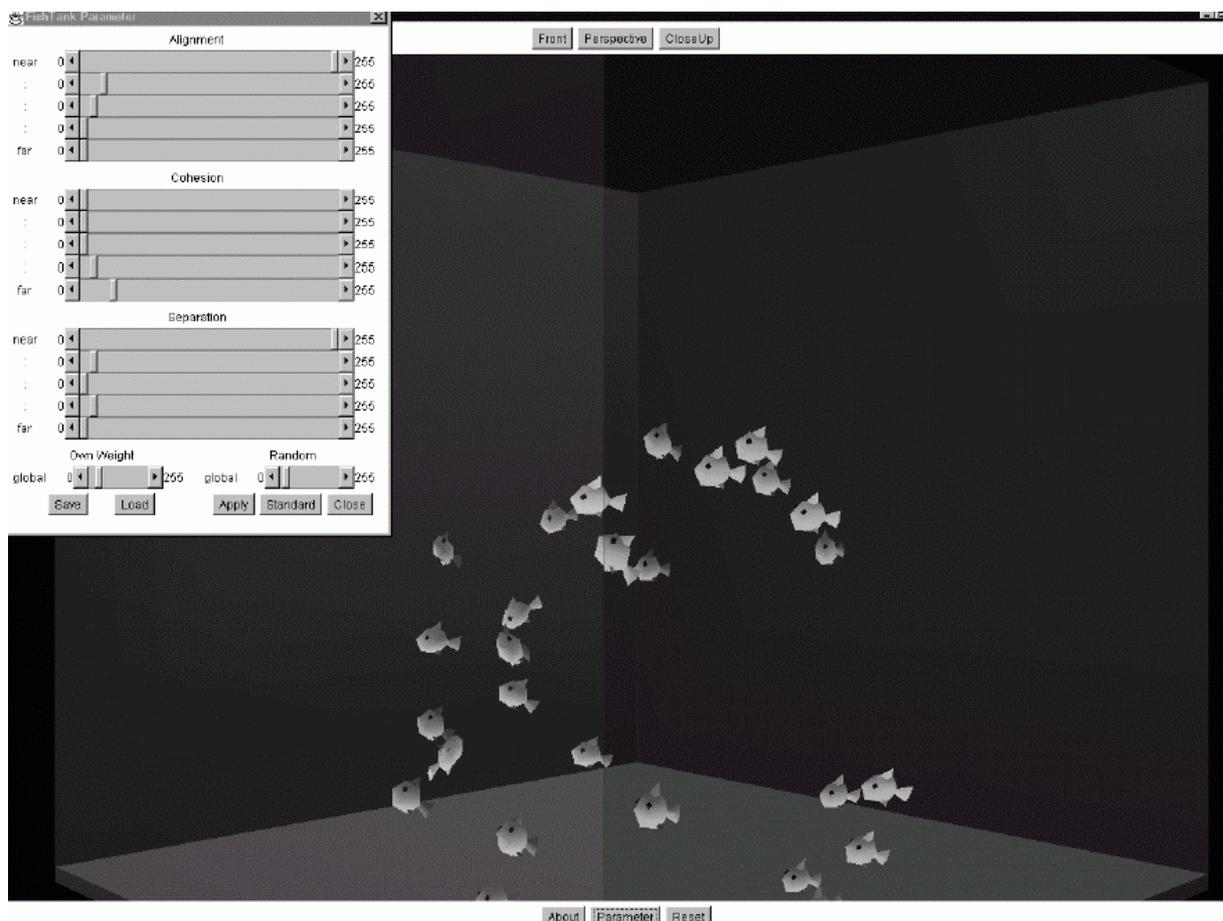
## Schwarmzentrierung

- Soll-Geschwindigkeit wird benutzt, um Ist-Geschwindigkeit allmählich anzupassen, ebenso für die Richtung: Vermeidung abrupter Änderungen
- je nach den Gewichtungsfaktoren im Navigationsmodul können unterschiedliche Verhaltensweisen modelliert werden

## Komplikation: Einführung von Raubvögeln

– zusätzliche Verhaltensregeln (Fernhalten von den Raubvögeln, Vermeiden ihrer Flugrichtung)

interaktive Systeme mit Möglichkeit der Regelung der Verhaltenspräferenz-Parameter

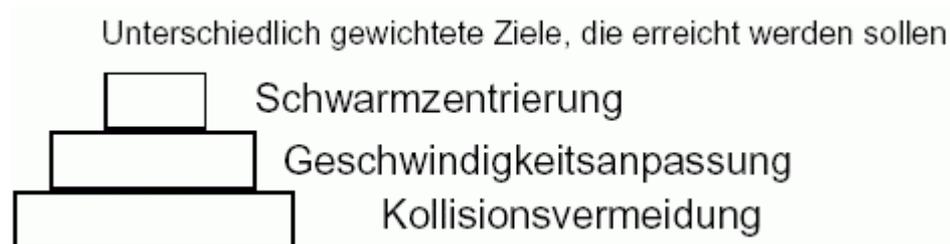


## Agentenbasierte Animation

wenige Einheiten mit sehr komplexen Verhaltensweisen

- *autonom*: Verhalten wird durch den Agenten selbst bestimmt, ohne Kontrolle von außen
- *reaktiv*: Agent kann Ereignisse wahrnehmen und auf dieser Grundlage seine Aktionen abstimmen
- *proaktiv*: Agent reagiert nicht nur auf Reize, sondern kann von sich aus die Initiative ergreifen
- *sozial*: strukturierte Kommunikation mit anderen Agenten ist möglich

Orientierung u.a. an psychologischen Modellen (z.B. hierarchische Bedürfnisstrukturen), Rückgriff auf KI-Methoden und wissenschaftliche Ansätze (Regelsysteme, neuronale Netze)



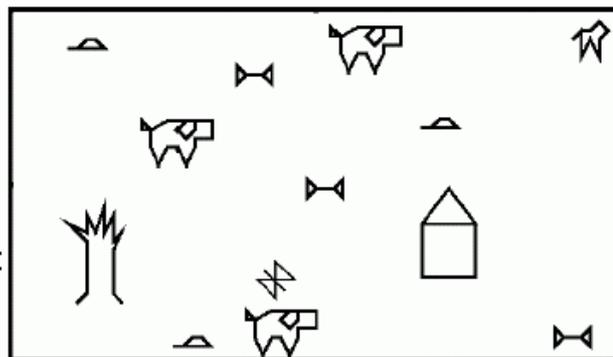
Beispiel:

“Hund jagt Katze” - Welt  
komplexes Verhalten der  
Elemente (Artificial  
Life)

Repräsentation durch 3D-  
Animationen

Verhaltenshierarchie Hund:

- \* keine Kollision
- \* kein Hunger
- \* Fange Katze
- \* Hilf anderen Hunden beim Jagen



Rules look like...

“If cat is nearly out of sight, moderately decrease speed and look for food. Otherwise, if cat is far away fully accelerate or if it is near in consideration with the cat’s position, direction and speed.”



## Vergleich Partikel, Boids, Agents

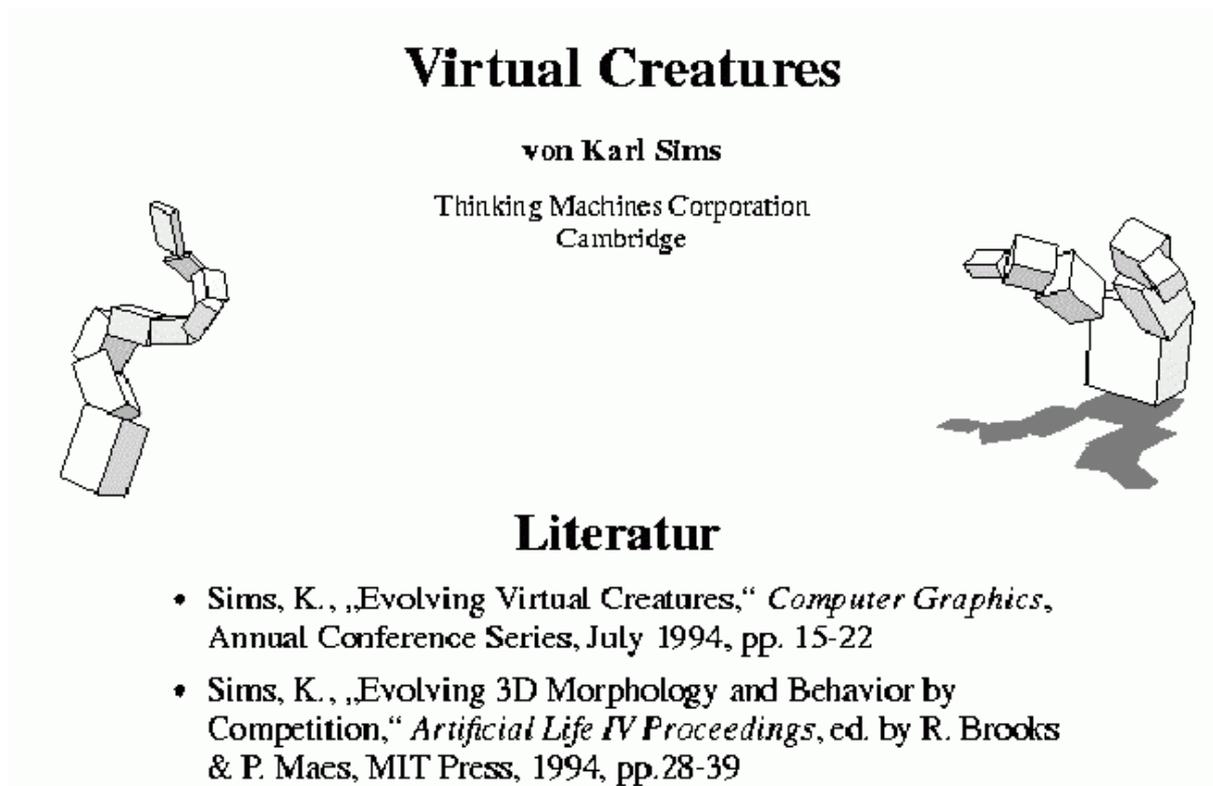
	Anzahl	Physikalische Regeln	Kollision	Kontrolle	Art der Strategien
Partikel	viele	ja	Nicht mit anderen Partikeln, sonst ja	niedrig	meist nur initiales Setzen von Parametern
Boids	mittel	teilweise	Vermeidung	mittel	meist durch einfache Regeln
Agenten	wenig	Weniger, meist eigenes Modell	Vermeidung, ggf. Planung für optimierte Wege	hoch	Komplexe Regeln, ggf. erweitert durch Hierarchien und lernende Ansätze

## "Virtual Creatures"

Anwendung wissensbasierter Techniken für Modellierung und Animation von virtuellen Kreaturen

- genetische Algorithmen
- künstliche neuronale Netze
- "Artificial Life" (AL)

wegweisende Artikel von Karl Sims (SIGGRAPH 1994 und Artificial Life IV - Konferenz 1994)



**Virtual Creatures**

von Karl Sims

Thinking Machines Corporation  
Cambridge

**Literatur**

- Sims, K., „Evolving Virtual Creatures,“ *Computer Graphics, Annual Conference Series*, July 1994, pp. 15-22
- Sims, K., „Evolving 3D Morphology and Behavior by Competition,“ *Artificial Life IV Proceedings*, ed. by R. Brooks & P. Maes, MIT Press, 1994, pp.28-39

Ziel:

Es sollen 3D-Individuen entwickelt werden, die in einer virtuellen Welt spezielle Aufgaben besonders gut bewältigen können

Struktur (Morphologie) und Verhalten (Regeln, Kontrollstrukturen) sind durch den genetischen Algorithmus variierbar

## Grundschrirte des genetischen Algorithmus:

1. Generiere Startpopulation mit Default-Parametern und Zufallsstreuung
2. Bestimme Qualität ("Fitness") der einzelnen Individuen
3. Selektion der Besten für Schritt 4
4. Erzeuge Nachkommengeneration – Anwendung von Mutationen und Crossing-Over (Modifikation und Mischung der Eltern-Parametersätze)
5. Gehe zu Schritt 2.

zu klärende Fragen:

- wie werden Morphologie und Funktionsweise (Verhalten) der Individuen codiert?
- wie wird die Fitness der Individuen ermittelt?
- wie werden die Codes an die Nachkommengeneration vererbt?

verschiedene Varianten für Morphologie und Verhalten möglich.

## Kodierung der Kreaturen: Gestalt

Genotyp: Strukturbeschreibung

Phänotyp: Ausprägung

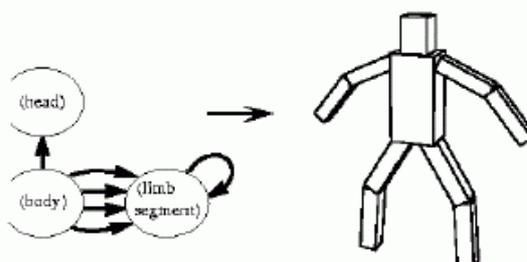
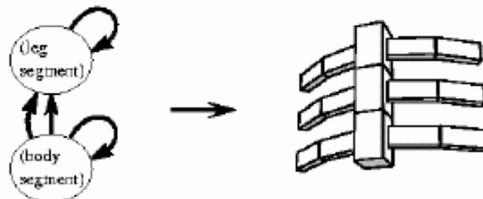
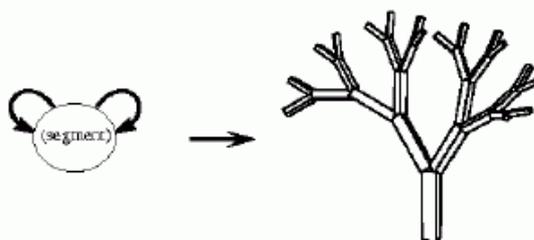
Die Erbinformationen über die Gestalt der Lebewesen wird als *gerichteter Graph* gespeichert.

Die **Knoten** entsprechen je einem **Körperteil** und enthalten folgende Informationen:

- Größe
- Gelenktyp
- Rekursionslimit
- Neuronen
- Verbindungen

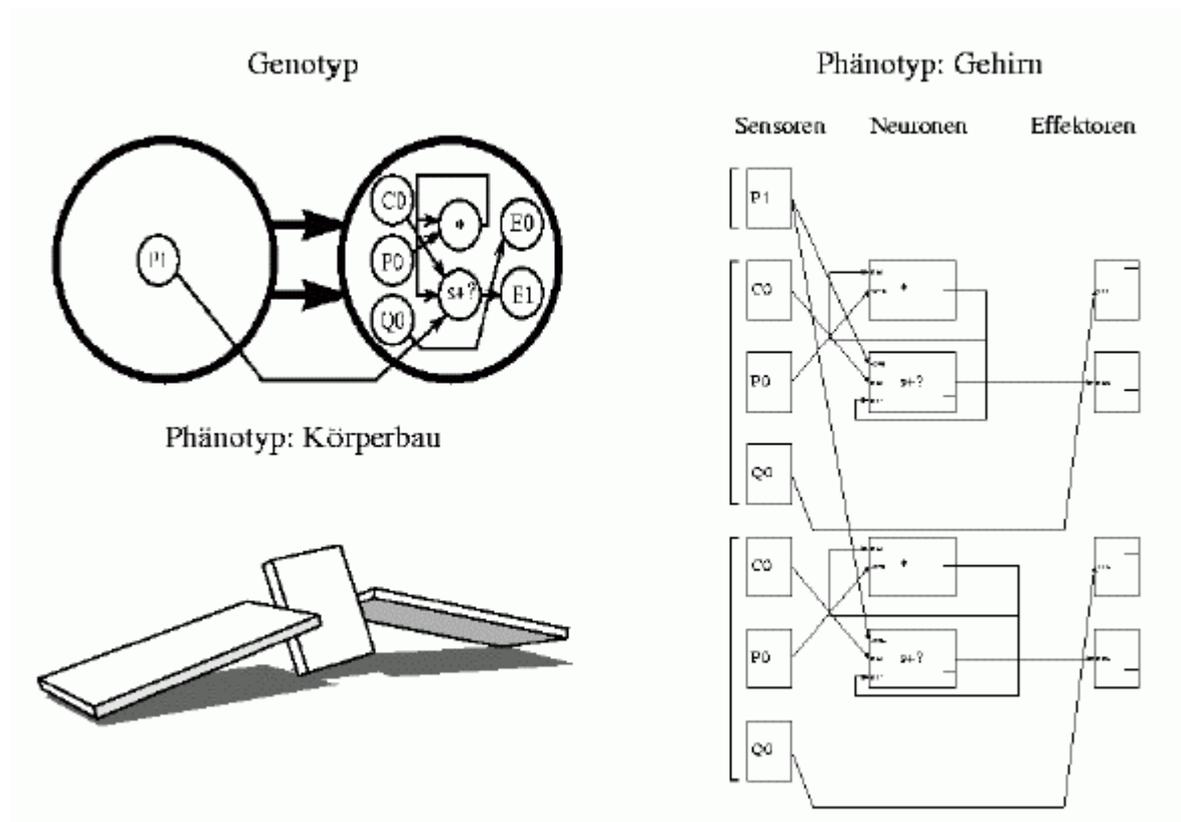
Die **Kanten** entsprechen den **Gelenken** und enthalten folgende Informationen:

- Position
- Orientierung
- Skalierungsfaktor
- Nur-Ende-Flag



## Aufbau einer Kreatur:

- segmentiertes System mit Gelenken
- Winkelsensoren und Effektoren für jedes Gelenk
- neuronale Schaltkreise in jedem Segment + "Gehirn" (neuronales Netz)



## Codierung durch Graphen:

- "äußerer Graph": Anordnung der Segmente (Glieder) und Gelenke
- "innerer Graph": Verschaltung der Sensoren, Neuronen und Effektoren

# Evolution

## Ein Genotyp wird in folgenden Schritten mutiert:

1. Interne Parameter der Knoten werden z. T. verändert.
2. Ein neuer Knoten wird hinzugefügt.
3. Parameter der Kanten werden z. T. verändert; evtl. zeigt die Kante anschließend auf einen anderen Knoten.
4. Neue Kanten werden evtl. hinzugefügt.
5. Nicht (mehr) befestigte Knoten werden entfernt

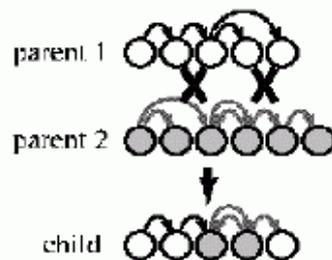
## Das Verfahren besteht aus zwei Phasen:

Zuerst wird der äußere Graph bearbeitet, dann der innere.

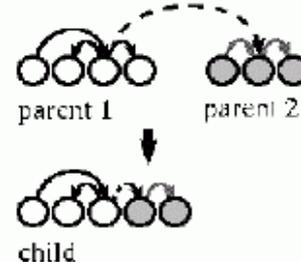
## Methoden:

1. [40%] Kopieren des Genotyps; dabei treten **Mutationen** auf
2. [30%] **Crossover** (evtl. mit Mutationen)
3. [30%] **Grafting** (evtl. mit Mutationen)

a. Crossovers:



b. Grafting:



Der beschriebene genetische Algorithmus wurde angewendet, um Lebewesen zu generieren, die eine der folgende Aufgaben besonders gut beherrschen:

**1. Schwimmen**

**2. Fortbewegen an Land**

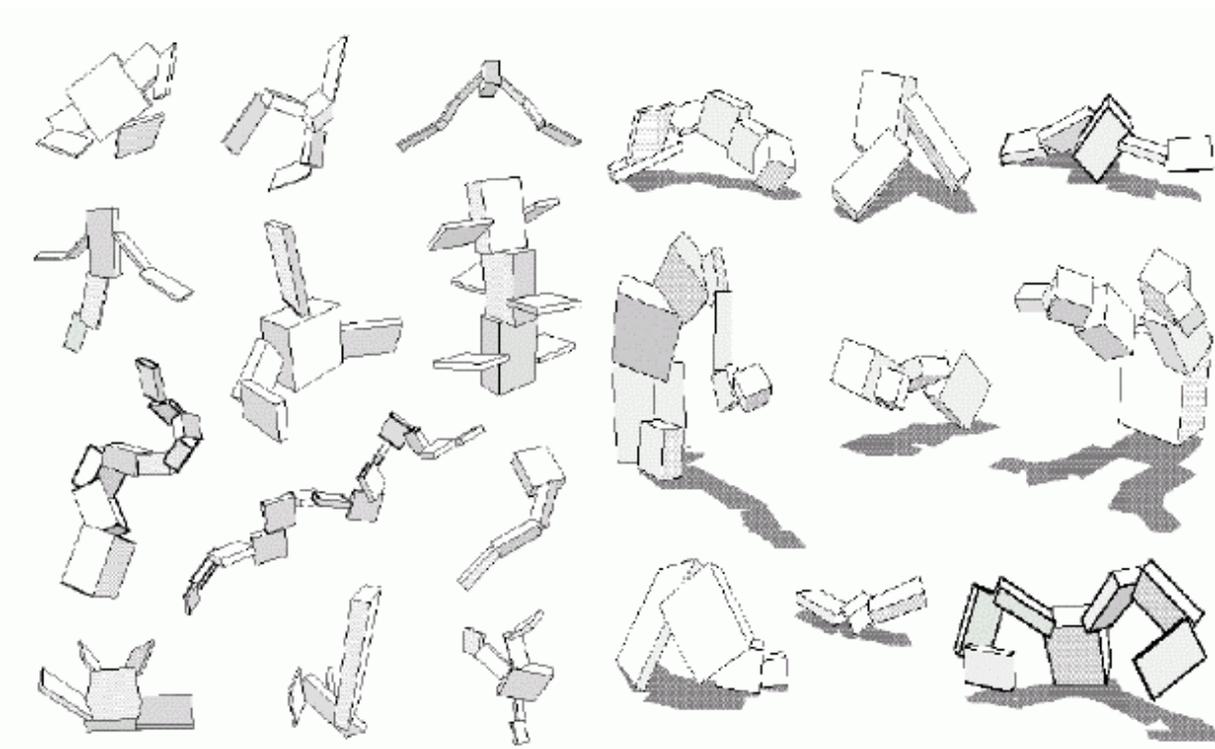
**3. Springen**

**4. Folgen einer Lichtquelle**

**5. Sich im **Wettkampf** gegen ein anderes Individuum durchsetzen**

### **Zur Implementierung:**

- Populationsgröße: 300
- jeweils 50 - 100 Generationen pro Evolution
- jeweils max. 10 sec. pro Individuum und Generation zur Qualitätsbewertung
- **Laufzeit:** 3 - 4 Std. pro Evolution mit 100 Generationen auf einer CM-5 mit 32 Prozessoren



*Aufgreifen des Grundprinzips und Umsetzung in ein Spiel:*

Stephen (Steve) Grand 1996, "Creatures" (Cyberlife Inc.)

<http://www.creatures.co.uk/>

