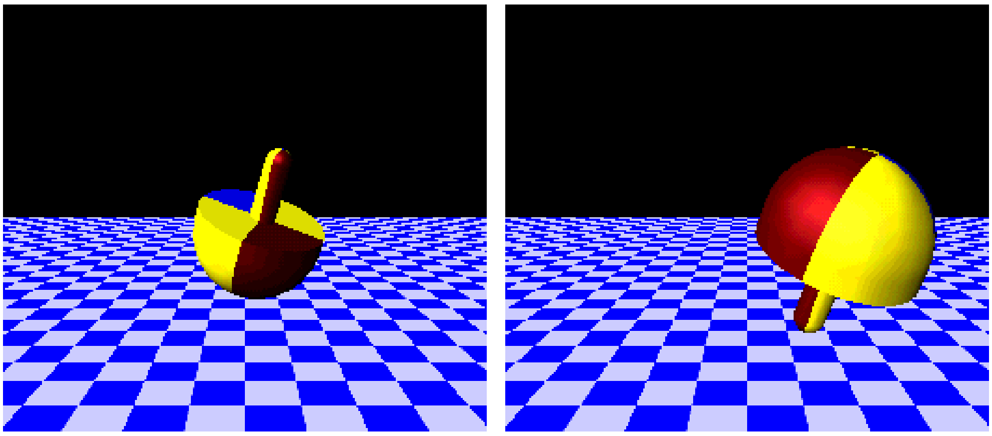
# Impulsbasierte Dynamiksimulation starrer Körper unter Verwendung von Hüllkörperhierarchien

Christian Lennerz
lennerz@cs.uni-sb.de

12. November 2002



Christian Lennerz

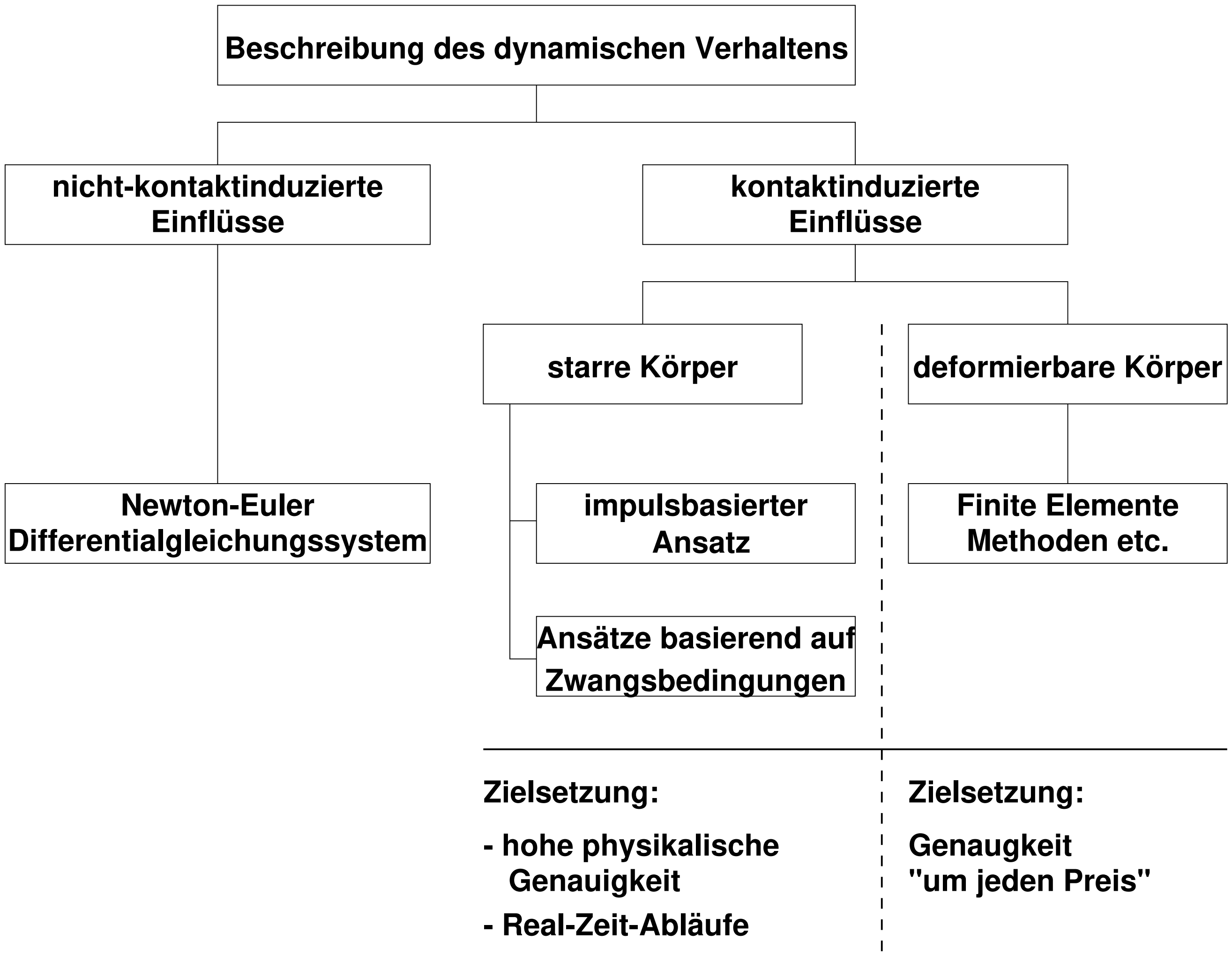
# Motivation der Aufgabenstellung

**Ausgangspunkt:**

- Wunsch das dynamische Verhalten von Körpern automatisiert voraussagen zu können
- Wunsch das Ergebnis einer Parametervariation sofort sehen zu können

**Herausforderung:**
Entwicklung eines Systems mit folgenden Eigenschaften:

- möglichst exakte Voraussage des dynamischen Verhaltens
- Antwortzeiten, die Interaktivität erlauben

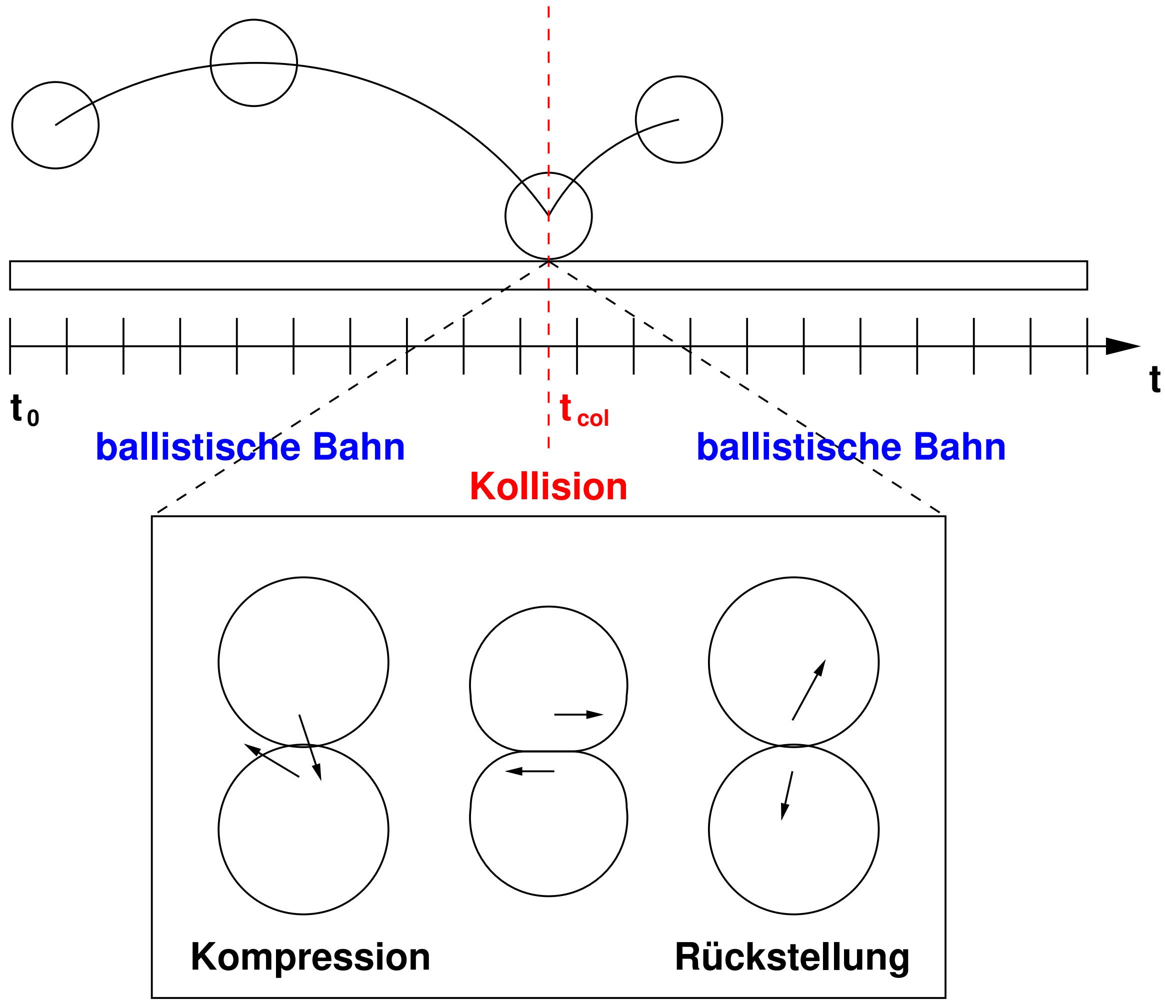

Beschreibung des dynamischen Verhaltens
nicht-kontaktinduzierte Einflüsse
kontaktinduzierte Einflüsse
starre Körper
deformierbare Körper
Newton-Euler Differentialgleichungssystem
impulsbasierter Ansatz
Ansätze basierend auf Zwangsbedingungen
Finite Elemente Methoden etc.
Zielsetzung:
- hohe physikalische Genauigkeit
- Real-Zeit-Abläufe
Zielsetzung:
Genaugkeit "um jeden Preis"

# Der impulsbasierte Ansatz

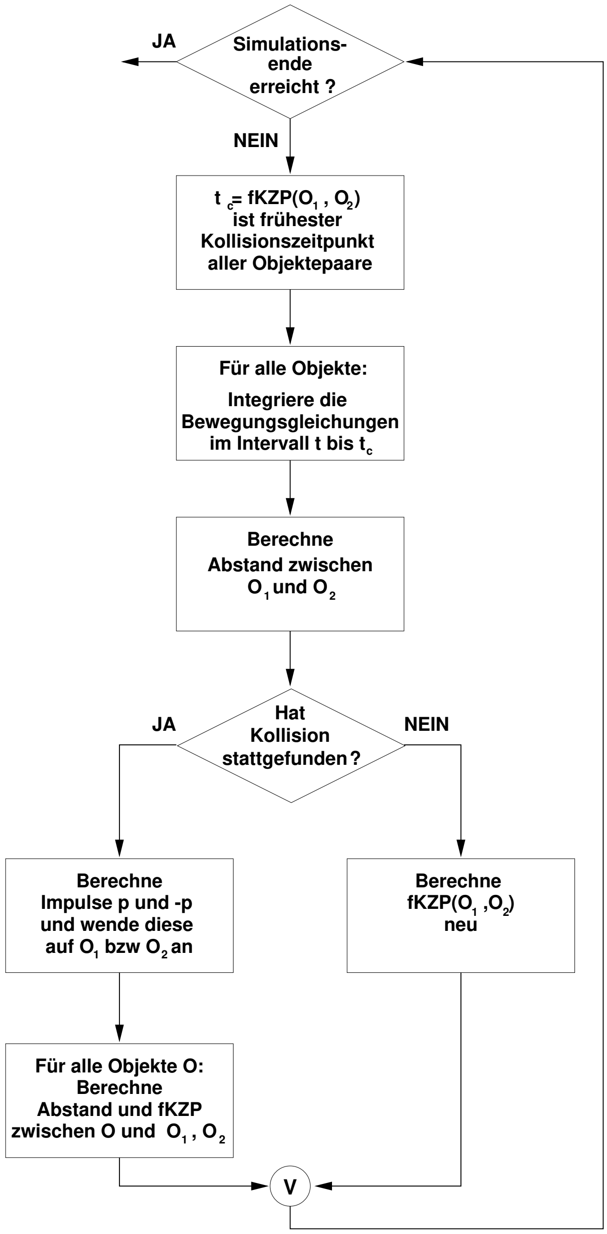
## Paradigma der impulsbasierten Simulation

- Alle Kontakte werden als Kollisionen im Kontaktpunkt behandelt.
- Zwischen den Kollisionszeitpunkten bewegen sich die Körper auf ballistischen Bahnen (nur Gravitation als externer Einfluß).
- Permanenter Kontakt wird als eine Folge von Mikrokollisionen aufgefaßt.

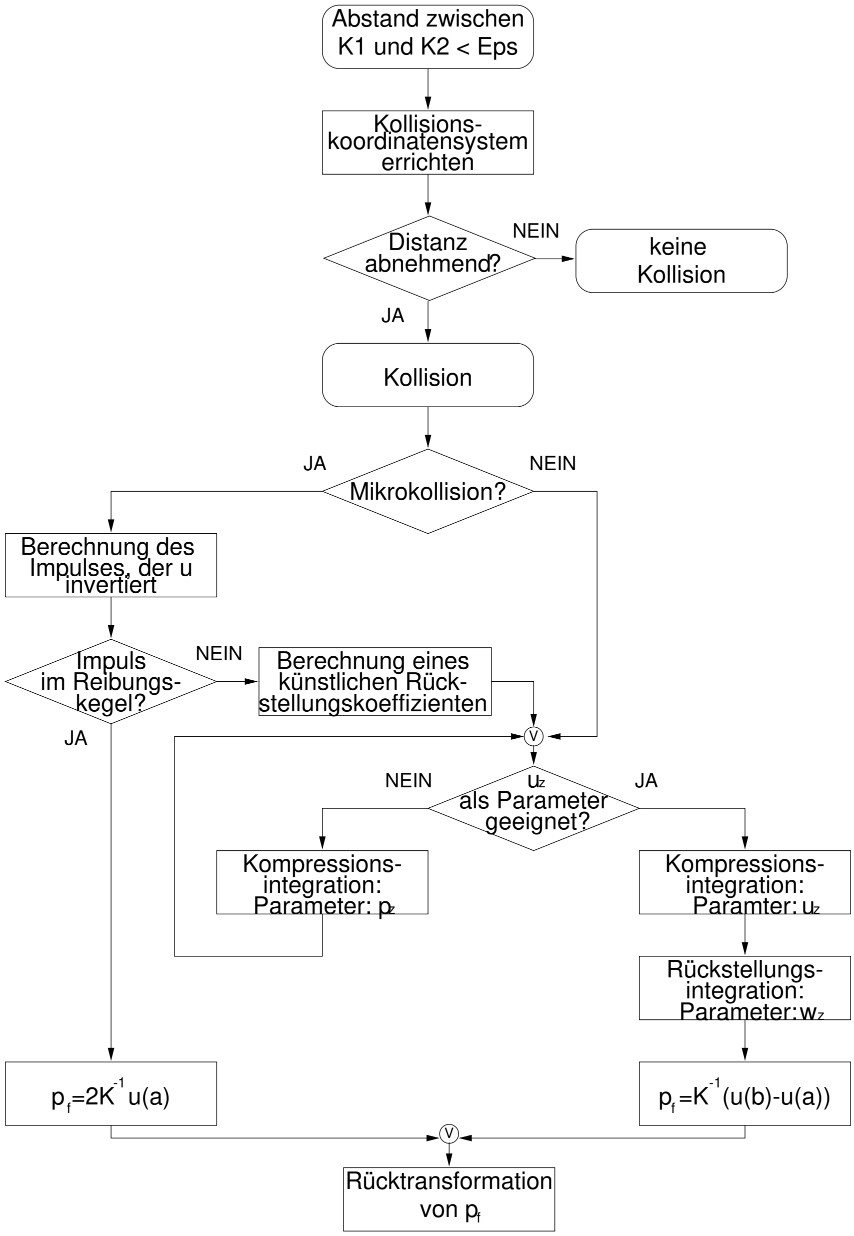
**Makroskopisches Verhalten ergibt sich durch exakte Behandlung der auftretenden Kollisionen (mikroskopisch)**


t
t0
tcol
ballistische Bahn
ballistische Bahn
Kollision
Kompression
Rückstellung

# Sinnvolles Zusammenspiel der Komponenten



# Übersicht über das Kollisionsauflösungssystem

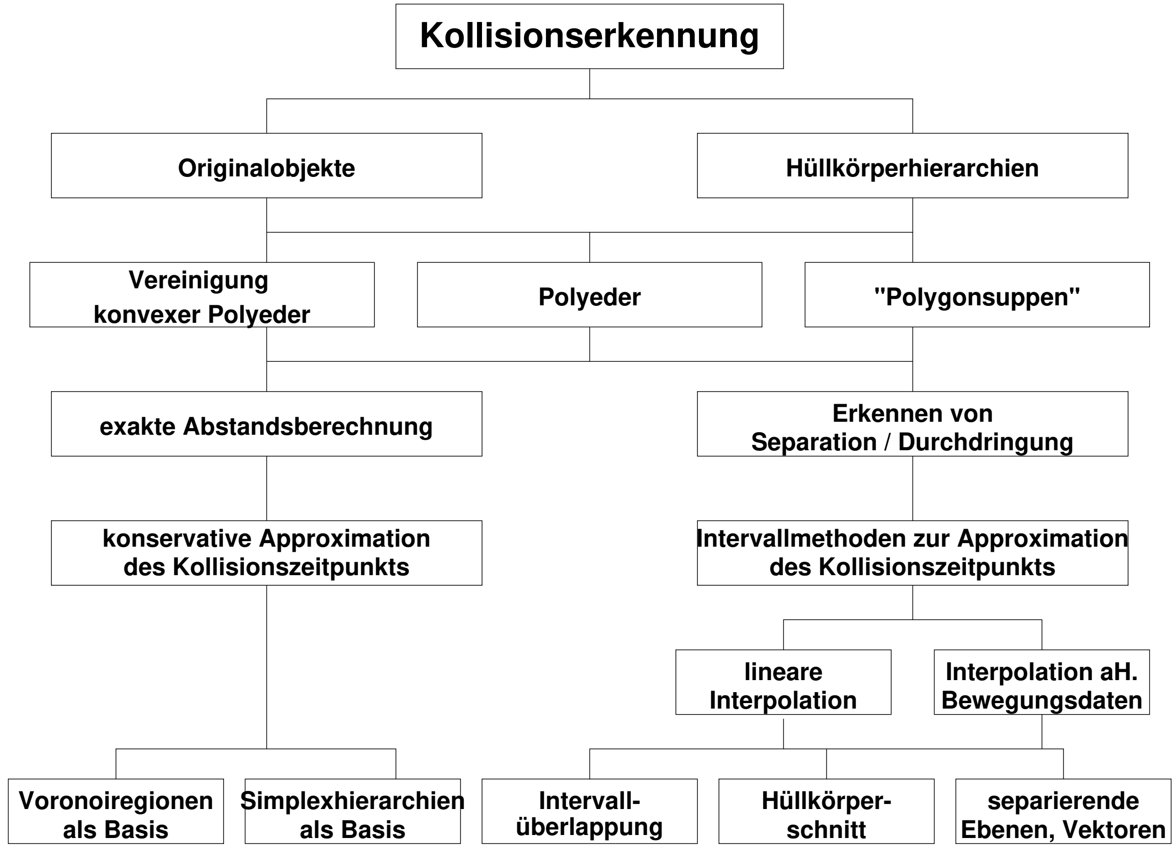


# Komplexität des Kollisionserkennungsproblems

1. *konvex* $\leftrightarrow$ *konvex*: $O((\log n)^2)$
   Dopkin/Kirkpatrick [83], [85], [90]

2. *konvex* $\leftrightarrow$ *nicht-konvex*: $O(n \log n)$
   Dopkin/Herschberger/Kirkpatrick/Suri [93]
   Doprindt/Mehlhorn/Yvinec [93]
   Schömer [94]

3. *nicht-konvex* $\leftrightarrow$ *nicht-konvex*: $subquadratisch$
   Schömer/Thiel [96]


Kollisionserkennung
Originalobjekte
Hüllkörperhierarchien
Vereinigung konvexer Polyeder
Polyeder
"Polygonsuppen"
exakte Abstandsberechnung
Erkennen von Separation / Durchdringung
konservative Approximation des Kollisionszeitpunkts
Intervallmethoden zur Approximation des Kollisionszeitpunkts
lineare Interpolation
Interpolation aH. Bewegungsdaten
Voronoiregionen als Basis
Simplexhierarchien als Basis
Intervall-überlappung
Hüllkörper-schnitt
separierende Ebenen, Vektoren

# Anforderungen an einen Kollisionserkennungsansatz im Rahmen der impulsbasierten Dynamiksimulation

1. allgemeine Objektrepräsentation

2. exakte Abstandsberechnung

3. Skalierbarkeit

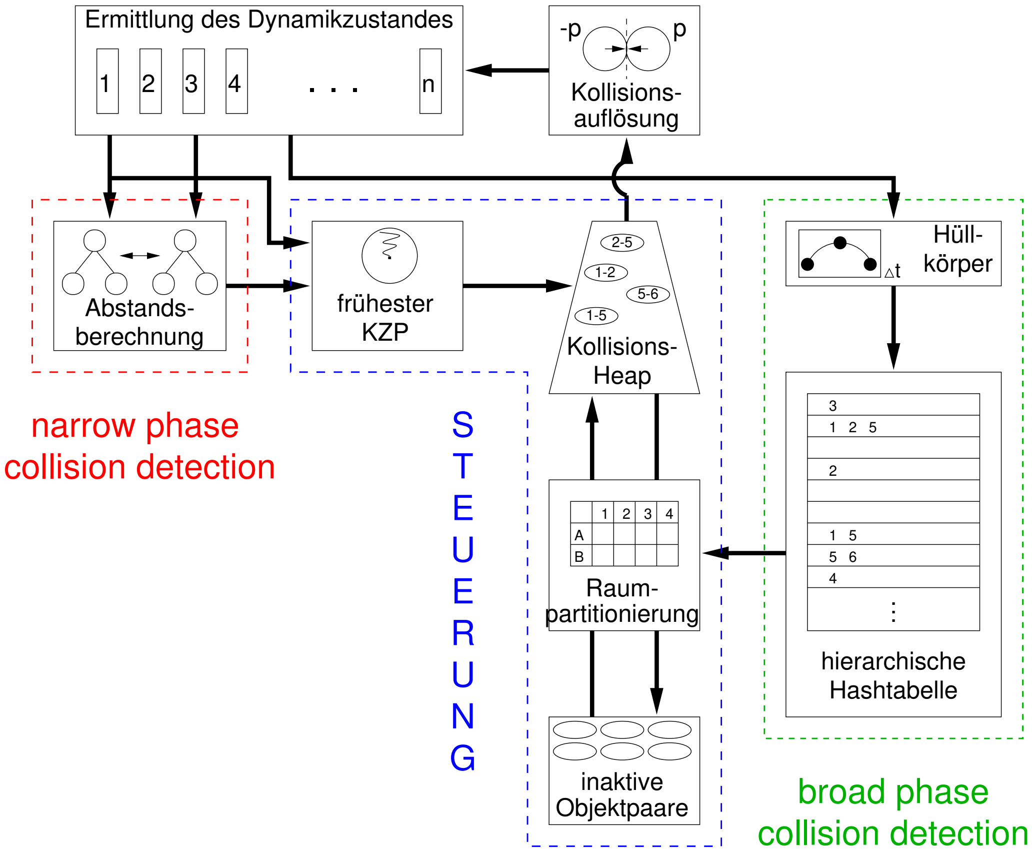
4. Ausnutzung temporärer Kohärenz

5. geringer Speicherplatzverbrauch

6. gute empirische Laufzeit

7. numerische Stabilität

# Übersicht über das Kollisionserkennungssystem



# Abstandsberechnung zwischen Polyedern

## Eigenschaften, der betrachteten Punktmengen

- zusammenhängend
- beschränkt
- realtive Lage im Zeitablauf konstant

## Explizite Modellierung der Körper mittels *Boundary Representation*

erfaßt *Topologie* und *Geometrie* des Randes $\partial P$ eines Polyeders $P$ durch Angabe der

- kartesischen Koordinaten der Eckpunkte ($V$)
- Inzidenzen zwischen Eckpunkten ($V$), Kanten ($E$) und Flächen ($F$)

## Definition. [minimaler Abstand zwischen zwei Punktmengen]

$$\delta(X_1, X_2) := \inf\{|x_1 - x_2| \mid x_i \in X_i, X_i \subset \mathbb{R}^3\}$$

**Lemma.** *Für zwei disjunkte starre Körper $X_1$ und $X_2$ gilt:*

$$\delta(\overline{X}_1, \overline{X}_2) = \delta(\partial X_1, \partial X_2)$$

**Beobachtung.** *Sind $P_1$, $P_2$ disjunkte kompakte Polyeder und $F_1$, $F_2$ die Mengen ihrer Begrenzungsflächen, dann gilt:*

$$\begin{aligned} \delta(P_1, P_2) = \delta(\partial P_1, \partial P_2) &= \delta(F_1, F_2) \\ &= \min\{\delta(f_1, f_2) \mid f_1 \in F_1, f_2 \in F_2\} \\ &= \min\{\delta(V_1, F_2), \delta(F_1, V_2), \delta(E_1, E_2)\} \end{aligned}$$

# Naiver Algorithmus zur Abstandsberchnung zwischen Polyedern

$\textsc{minDist}(f_1, f_2)$

$d \leftarrow \infty$<br>
**for each** $v_1$ $of$ $f_1$<br>
**do** $d \leftarrow \min\{d, \textsc{minDist}(v_1, f_2)\}$<br>
**for each** $v_2$ $of$ $f_2$<br>
**do** $d \leftarrow \min\{d, \textsc{minDist}(v_2, f_1)\}$<br>
**for each** $e_1$ $of$ $f_1$<br>
**do for each** $e_2$ $of$ $f_2$<br>
    **do** $d \leftarrow \min\{d, \textsc{minDist}(e_1, e_2)\}$<br>
**return** $d$

$\textsc{minDist}(P_1, P_2)$

$d \leftarrow \infty$<br>
**for each** $f_1$ $of$ $\partial P_1$<br>
**do for each** $f_2$ $of$ $\partial P_2$<br>
    **do** $d \leftarrow \min\{d, \textsc{minDist}(f_1, f_2)\}$<br>
**return** $d$

# Das Branch-and-Bound-Paradigma zur Lösung von Minimierungsproblemen

*Geg:* eine endliche Menge $X$, $f : X \to \mathbb{R}$

*Ges:* $z = \min\{f(x) \mid x \in X\}$

$\textsc{findMin}(f, Y \subset X)$

**if** $Y = \emptyset$
  **then return**
**if** $Y = \{y\}$
  **then** $curMin \leftarrow \min\{curMin, f(y)\}$
  **else** $\mu_Y \leftarrow \textsc{lowBound}(Y)$
    **if** $curMin < \mu_Y$
      **then return**
      **else** $\textsc{partition}(Y, Y_1, \cdots, Y_k)$
        **for each** $i \in \{1, \cdots, k\}$
        **do** $\textsc{findMin}(f, Y_i)$

$\textsc{genericB\&B}(f, X)$

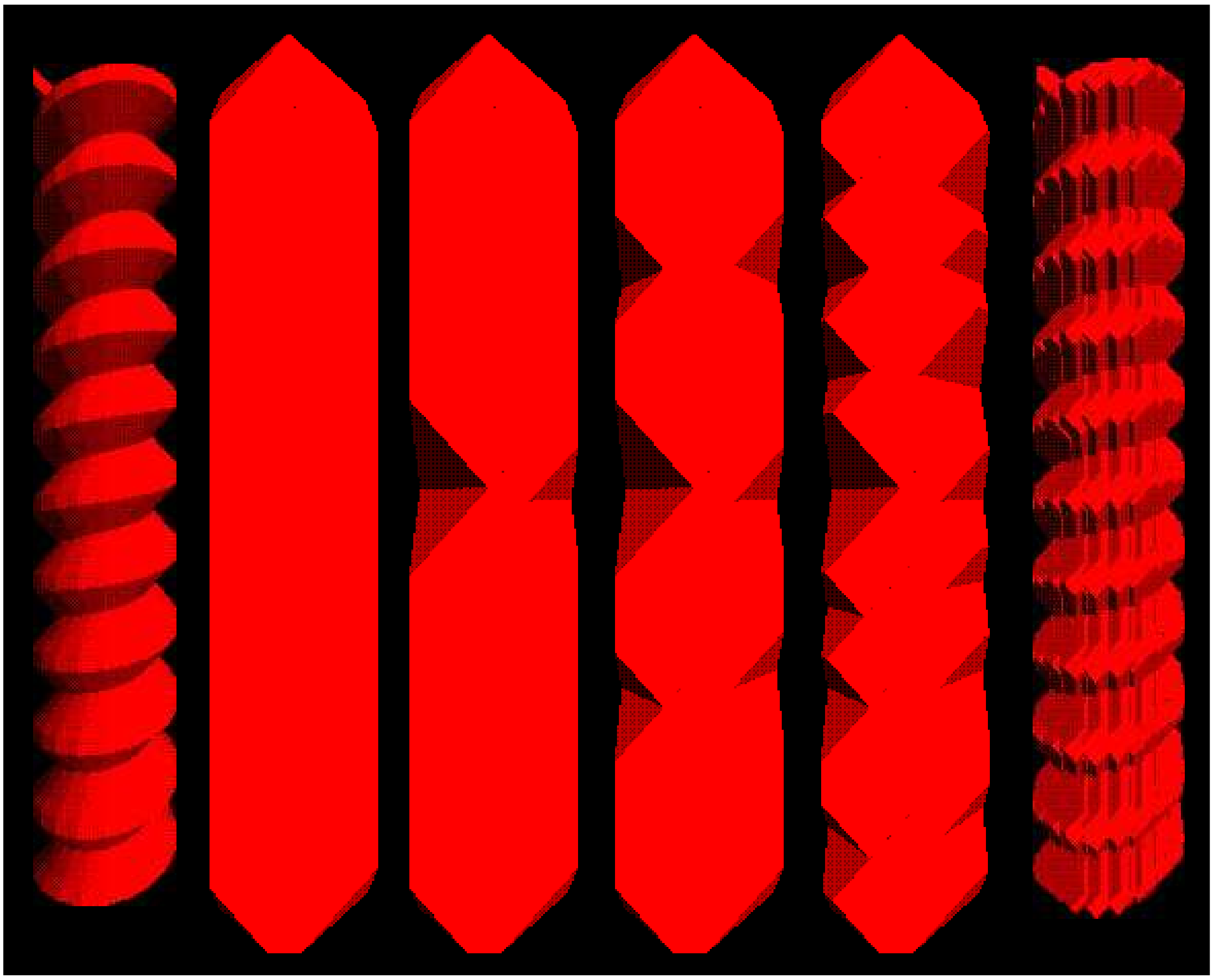
$curMin \leftarrow \infty$
$\textsc{findMin}(f, X)$
**return** $curMin$

# Instantiierung des generischen Verfahrens mit obiger Problemstellung

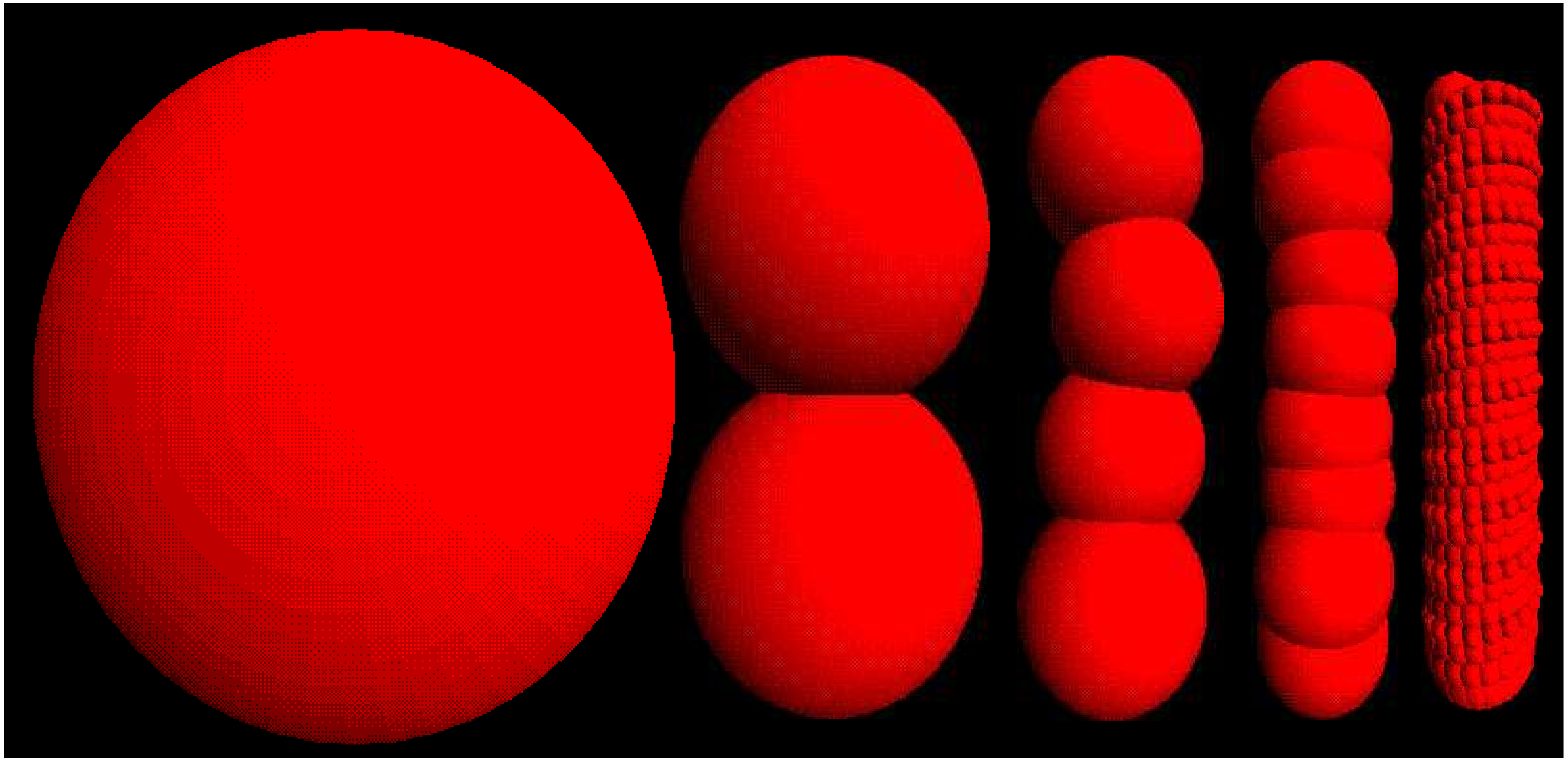
- $X = \{\{f_1, f_2\} \mid f_1 \in F_1, f_2 \in F_2\}$
- $f : X \to \mathbb{R}$
  $(f_1, f_2) \longmapsto \delta(f_1, f_2)$
- untere Schranke: Hüllkörperabstand
- Partitionierung: Vorverarbeitung (kein Einfluß auf Laufzeit der Abstandsanfragen)
  - Anzahl der Partitionen $k \equiv$ Grad der Hierarchie:
    **Wir wählen** $k = 2$
  - Top-Down- versus Bottom-Up-Konstruktion
    **Wir wählen Top-Down Vorhensweise.**
  - Aufteilungsstrategien
    * Optimiere vorgegebene Zielfunktion (Eckstein) oder
    * Wende Heuristiken an

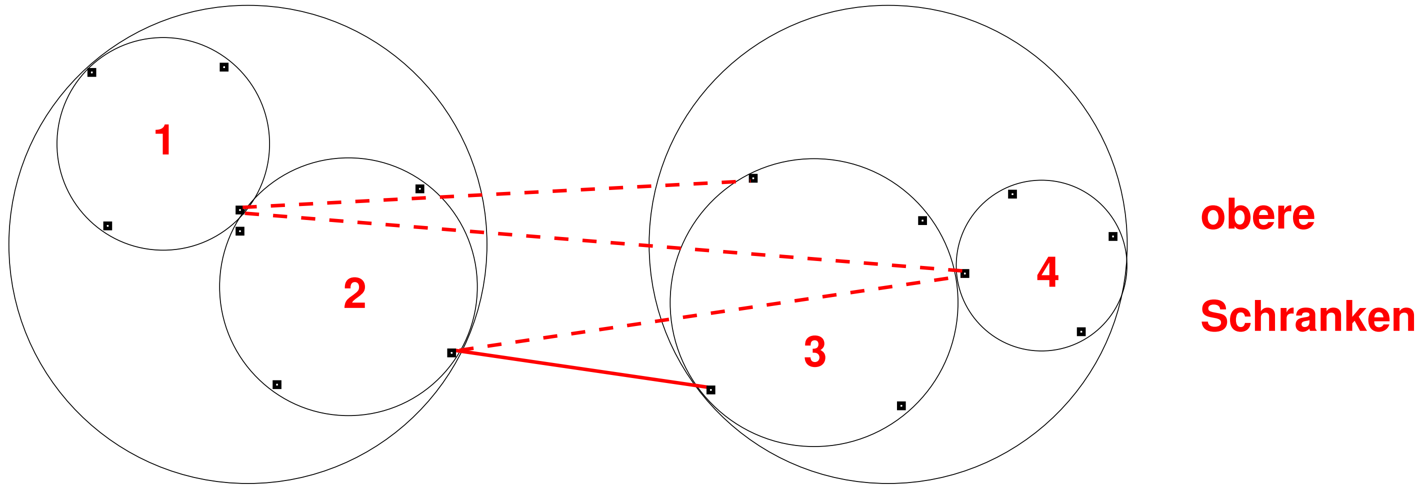
    **Wir wählen heuristische Strategien.**

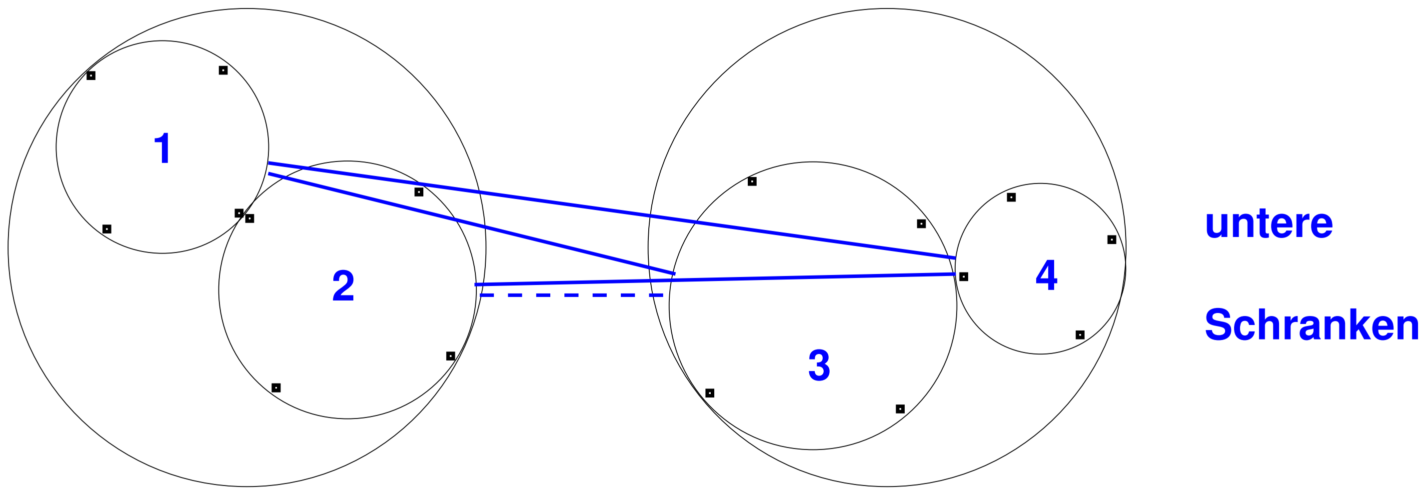
# $FDH_{14}$-Hierarchien



# Kugelhierarchien




1
2
3
4
obere
Schranken


1
2
3
4
untere
Schranken

# Ein erster B&B-Algorithmus

```
NODEDIST(p1, p2)
if p1 is not a leaf
   then if p2 is not a leaf
           then for each child v1 of p1
                   do v1.TRANSFORM(f)
                      for each child v2 of p2
                      do d ← LOWBOUND(v1, v2)
                         if d < ub
                            then NODEDIST(v1, v2)
           else for each child v1 of p1
                   do v1.TRANSFORM(f)
                      d ← LOWBOUND(v1, p2)
                      if d < ub
                         then NODEDIST(v1, p2)
   else if p2 is not a leaf
           then for each child v2 of p2
                   do d ← LOWBOUND(p1, v2)
                      if d < ub
                         then NODEDIST(p1, v2)
           else ub ← min{ub, LEAFDIST(p1, p2)}
```

```
nodeDist(p1, p2)
init(d);
ub ← min{ub, repDist(p1, p2)}
if p1 is not a leaf
  then if p2 is not a leaf
         then for i ← 1 to p1.NumberOfChildren
                do v1 ← p1.Child[i]
                   v1.Transform(f)
                   for j ← 1 to p2.NumberOfChildren
                   do v2 ← p2.Child[j]
                      d[i][j] ← lowBound(v1, v2)
         else j ← 1
              for i ← 1 to p1.NumberOfChildren
              do v1 ← Child[i]
                 v1 ← Transform(f)
                 d[i][j] ← lowBound(v1, p2)
  else if p2 is not a leaf
         then i ← 1
              for j ← 1 to p2.NumberOfChildren
              do v2 ← p2.Child[j]
                 d[i][j] ← lowBound(p1, v2)
         else ub ← min{ub, leafDist(p1, p2)}
              return
for n ← 1 to i + j
do (k, l) ← minIndex(d, i, j)
   if d[k][l] ≥ ub
     then return
   d[k][l] ← ∞
   minDist(p1.Child[k], p2.Child[l])
```

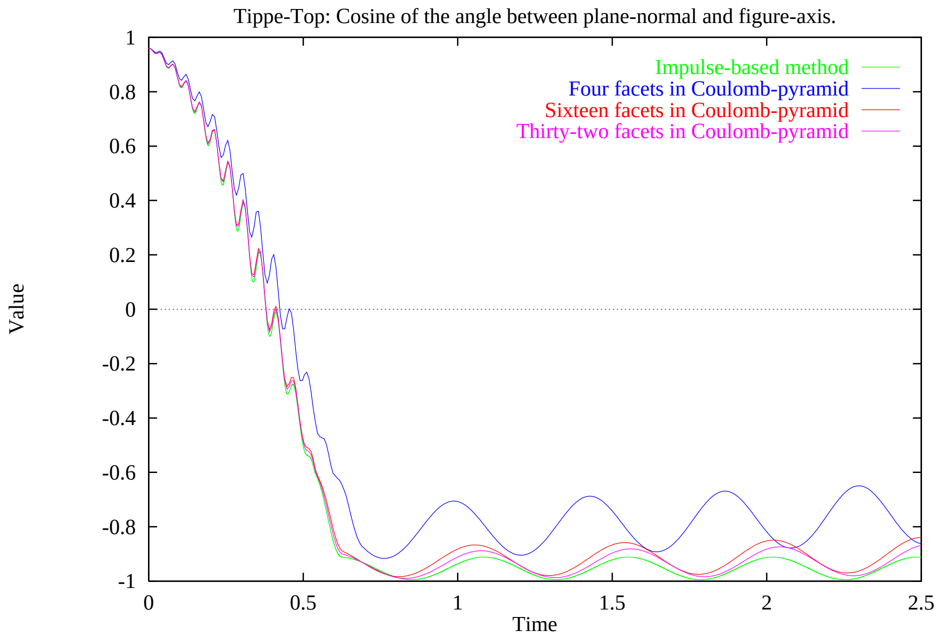
$\textsc{nodeDist}(r_1, r_2)$

$d \leftarrow \textsc{lowBound}(r_1, r_2)$<br>
$lb \leftarrow d$<br>
$sortSeq.\textsc{insert}(d, (r_1, r_2))$<br>
**while** $sortSeq$ is not empty and $lb < ub$<br>
**do** $(p_1, p_2) \leftarrow sortSeq.\textsc{delMin}()$<br>
    $ub \leftarrow \min\{ub, \textsc{repDist}(p_1, p_2)\}$<br>
    **while** $sortSeq.\textsc{max}() > ub$<br>
    **do** $sortSeq.\textsc{delMax}()$<br>
    **if** $p_1$ is a leaf and $p_2$ is a leaf<br>
      **then** $fd \leftarrow \min\{fd, \textsc{leafDist}(p_1, p_2)\}$<br>
          $ub \leftarrow \min\{ub, fd\}$<br>
      **else** $\textsc{insertChildren}(sortSeq, p1, p2)$<br>
    **if** $sortSeq$ is not empty<br>
      **then** $d \leftarrow \min\{fd, sortSeq.\textsc{minKey}()\}$<br>
          $lb \leftarrow \max\{lb, d\}$

```
insertChildren(S, p1, p2))
if p_1 is not a leaf
   then if p_2 is not a leaf
           then for each child v_1 of p_1
                   do v_1.Transform(f)
                      for each child v_2 of p_2
                      do d ← lowBound(v_1, v_2)
                         if d ≤ ub
                            then S.insert(d, v_1, v_2)
           else for each child v_1 of p_1
                   do v_1.Transform(f)
                      d ← lowBound(v_1, p_2)
                      if d ≤ ub
                         then S.insert(d, v_1, p_2)
   else if p_2 is not a leaf
           then for each child v_2 of p_2
                   do d ← lowBound(p_1, v_2)
                      if d ≤ ub
                         then S.insert(d, p_1, v_2)
```

# Resultate

## Validierung:




## Laufzeitmessung („worst case“):

| Algorithmus | # Elementartests |
|---|---|
| naiv | $> 1000000$ |
| B&B | 10833 |
| B&B + Reps | 6736 |
| B&B + greedy | 866 |
| B&B + sortSeq | 769 |