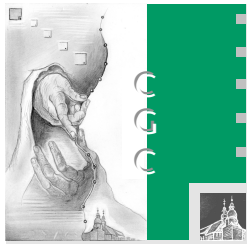
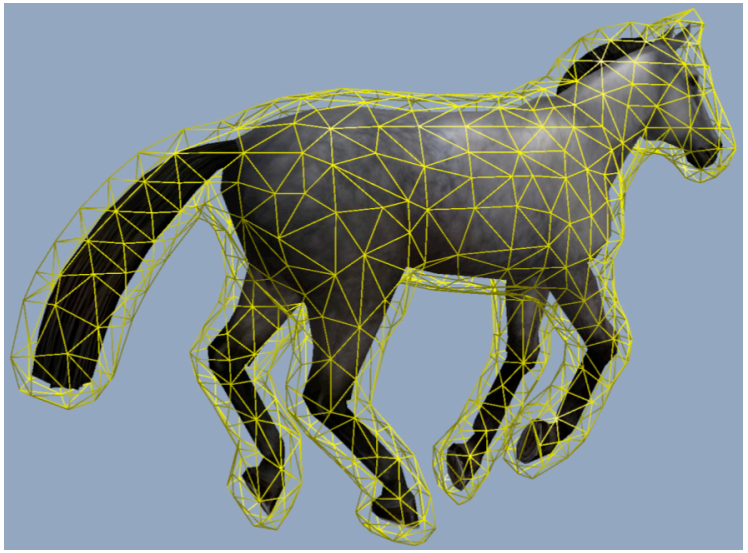


Virtuelle Realität
Feder-Masse-Systeme



G. Zachmann

Clausthal University, Germany

cg.in.tu-clausthal.de

Die Newton'schen Gesetze

1. Gesetz (Trägheitsgesetz):

> Ein kräftefreier Körper bewegt sich geradlinig mit konstanter Geschwindigkeit weiter.

- Ein Körper in Ruhe ist also nur ein Spezialfall hiervor.

2. Gesetz (Aktionsprinzip):

> Wenn eine Kraft **F** auf einen Körper mit Masse m wirkt, beschleunigt sie diesen gemäß
>
> $$\mathbf{F} = m \cdot \mathbf{a}$$

- Mit anderen Worten: Kraft und Beschleunigung sind proportional zueinander; die Proportionalitätskonstante ist m. Insbesondere haben beide dieselbe Richtung.

3. Gesetz (Reaktionsprinzip):

Wenn die Kraft **F**, die auf einen Körper wirkt, ihren Ursprung in einem anderen Körper hat, so wirkt auf diesen die entgegengesetzte Kraft **–F** .

- In der Schule lernt man : "Kraft = Gegenkraft"

4. Gesetz (Superpositionsprinzip):

Wirken auf einen Punkt (oder einen starren Körper) mehrere Kräfte $\mathbf{F}_1, \ldots, \mathbf{F}_n$, so addieren sich diese vektoriell zu einer resultierenden Kraft **F** auf:

$$\mathbf{F} = \mathbf{F}_1 + \ldots + \mathbf{F}_n \, .$$

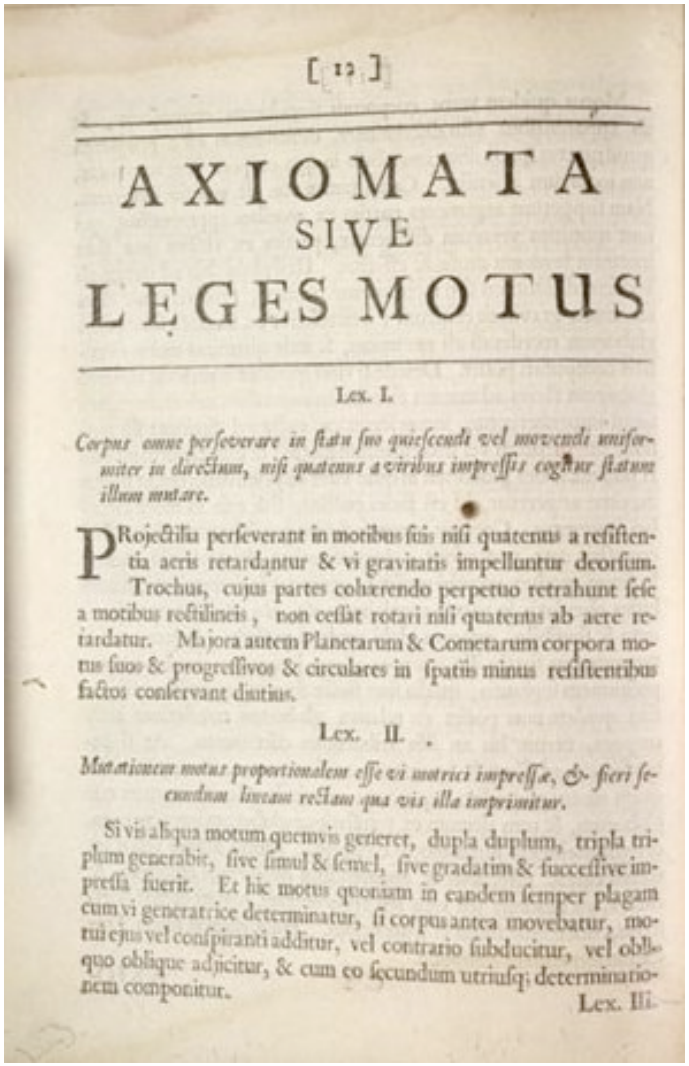
Historischer Exkurs

- Newton's Gesetze in der Originalschrift

  Principia Mathematica

  (1687):

  - *Lex I. Corpus omne perseverare in statu suo quiescendi vel movendi uniformiter in directum, nisi quatenus illud a viribus impressis cogitur statum suum mutare.*
  - *Lex II. Mutationem motus proportionalem esse vi motrici impressae, et fieri secundum lineam rectam qua vis illa imprimitur.*



[13]

AXIOMATA
SIVE
LEGES MOTUS

Lex. I.

Corpus omne perseverare in statu suo quiescendi vel movendi uniformiter in directum, nisi quatenus a viribus impressis cogitur statum illum mutare.

Projectilia perseverant in motibus suis nisi quatenus a resistentia aeris retardantur & vi gravitatis impelluntur deorsum. Trochus, cujus partes cohærendo perpetuo retrahunt sese a motibus rectilineis, non cessat rotari nisi quatenus ab aere retardatur. Majora autem Planetarum & Cometarum corpora motus suos & progressivos & circulares in spatiis minus resistentibus factos conservant diutius.

Lex. II.

Mutationem motus proportionalem esse vi motrici impressæ, & fieri secundum lineam rectam qua vis illa imprimitur.

Si vis aliqua motum quemvis generet, dupla duplum, tripla triplum generabit, sive simul & semel, sive gradatim & successive impressa fuerit. Et hic motus quoniam in eandem semper plagam cum vi generatrice determinatur, si corpus antea movebatur, motui ejus vel conspiranti additur, vel contrario subducitur, vel obliquo oblique adjicitur, & cum eo secundum utriusq; determinationem componitur.

Lex. III.

- Definition:

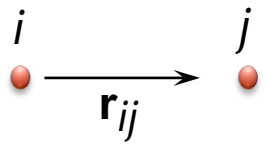
  Ein Feder-Masse-System ist ein System, bestehend aus:

  1. einer Menge von Punktmassen m_i mit Positionen $\mathbf{x}_i$ und Geschwindigkeit $\mathbf{v}_i$, i = 1...N ;
  2. einer Menge von Federn $s_{ij} = (i, j, k_s, k_d)$, die die Massen i und j verbindet, mit der Ruhelänge l_0, Federkonstante (= Steifigkeit) k_s und den Dämpfungskoeffizienten k_d

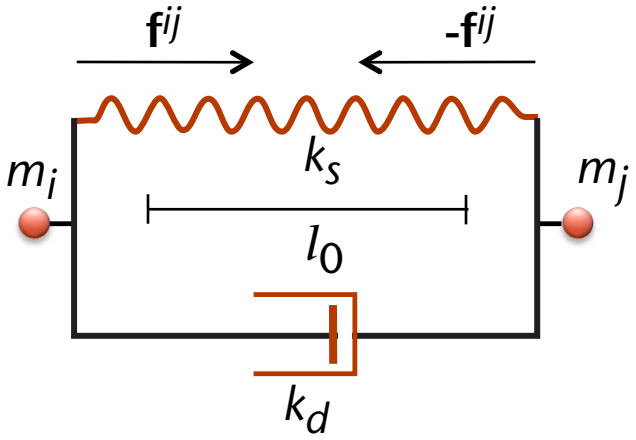
- Vorteile:
  - Sehr einfach zu programmieren
  - Ideal zum Studium verschiedener numerischer Lösungsverfahren
  - *Ubiquitous in games* (cloths, capes, ...)
- Nachteile:
  - Einige Parameter (Federkonstanten z.B.) sind nicht offensichtlich
  - Keine volumetrischen Effekte (z.B. Volumenerhaltung)

Eine einzelne Feder (mit Dämpfer)

- Gegeben: Massen m_i und m_j mit Positionen $\mathbf{x}_i$, $\mathbf{x}_j$
- Definiere $\mathbf{r}_{ij} = \dfrac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|}$



- Kräfte zwischen den beiden Massen :



1. Von der Feder:

   $$\mathbf{f}_s^{ij} = k_s \mathbf{r}_{ij}(\|\mathbf{x}_j - \mathbf{x}_i\| - l_0)$$

   wirkt auf Masse m_i in Richtung m_j

2. Durch den Dämpfer: $\mathbf{f}_d^{ij} = k_d((\mathbf{v}_j - \mathbf{v}_i)\cdot\mathbf{r}_{ij})\mathbf{r}_{ij}$
3. Zusammen : $\mathbf{f}^{ij} = \mathbf{f}_s^{ij} + \mathbf{f}_d^{ij}$
4. Kraft auf m_j : $\mathbf{f}^{ji} = -\mathbf{f}^{ij}$

- Aus (4) → der Impuls bleibt erhalten!

- Alternative Federkraft:

$$\mathbf{f}_s^{ij} = k_s \, \mathbf{r}_{ij} \, \frac{|\mathbf{x}_j - \mathbf{x}_i| - l_0}{l_0}$$

Simulation einer einzelnen Feder

- Aus Newton'schen Gesetzen folgt: $\ddot{\mathbf{x}} = \frac{1}{m}\mathbf{f}$
- DGL der Ordnung 2 umwandeln in DGLs erster Ordnung:

$$\dot{\mathbf{v}}(t) = \frac{1}{m}\mathbf{f}(t)$$

$$\dot{\mathbf{x}}(t) = \mathbf{v}(t)$$

- Anfangswerte: $\mathbf{v}(t_0) = \mathbf{v}_0\ ,\quad \mathbf{x}(t_0) = \mathbf{x}_0$
- "Simulation" = "Integration von DGLs über die Zeit"
- Taylor-Expansion liefert:

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t\, \dot{\mathbf{x}}(t) + O(\Delta t^2)$$

- Analog: $\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \Delta t\, \dot{\mathbf{v}}(t)$

→ Dieses Integrationsverfahren heißt **explizite Euler-Integration**

Der Algorithmus

```
forall particles i :
  initialize x_i, v_i, m_i

loop:
  forall particles i :
```

$$\mathbf{f}_i \leftarrow \mathbf{f}^g + \mathbf{f}_i^{coll} + \sum_{j,(i,j)\in S} \mathbf{f}(\mathbf{x}_i, \mathbf{v}_i, \mathbf{x}_j, \mathbf{v}_j)$$

```
  forall particles i :
```

$$\mathbf{v}_i \mathrel{+}= \Delta t \cdot \frac{\mathbf{f}_i}{m_i}$$

$$\mathbf{x}_i \mathrel{+}= \Delta t \cdot \mathbf{v}_i$$

```
  render system every n-th time
```

$\mathbf{f}^g$ = Gravitationskraft

$\mathbf{f}^{coll}$ = Rückstoßkraft aus Kollision (z.B. mit Hindernis)

- Weitere Integrations-Schemata:
  - Midpoint
  - ...
- Vorteil: einfach zu implementieren, schnelle Ausführung pro Zeitschritt
- Nachteil: nur für sehr kleine Schrittweiten stabil
  - Typ.weise $\Delta t \approx 10^{-4} \ldots 10^{-3}$ sec!
  - Bei großen Schrittweiten wird zusätzliche Energie im System erzeugt, und schließlich explodiert es ☺
  - Beispiel: Overshooting bei einfacher Feder
- Weiterer Nachteil: Fehler akkumulieren sich schnell (bei Euler)

Beispiel für die Instabilität des Euler-Integrationsverfahrens

- Betrachte die DGL

$$\dot{x}(t) = -kx(t)$$

- Die exakte Lösung:

$$x(t) = x_0\, e^{-kt}$$

- Das Euler-Verfahren liefert:

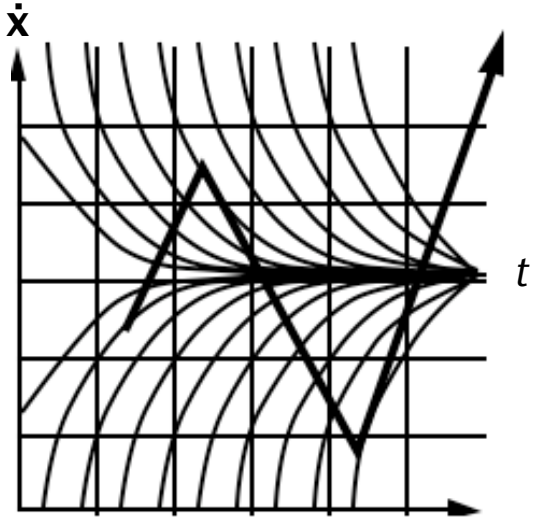
$$x^{t+1} = x^t + \Delta t(-kx^t)$$

- Falls $\Delta t > \frac{1}{k}$:

$$x^{t+1} = x^t \underbrace{(1 - k\Delta t)}_{<0}$$

$\Rightarrow$ x^t oszilliert um 0,
geht aber (hoffentlich) gegen 0

- Falls $\Delta t > \frac{2}{k}$ $\Rightarrow$ $x^t \to \infty$!



Veranschaulichung der Fehlerakkumulation

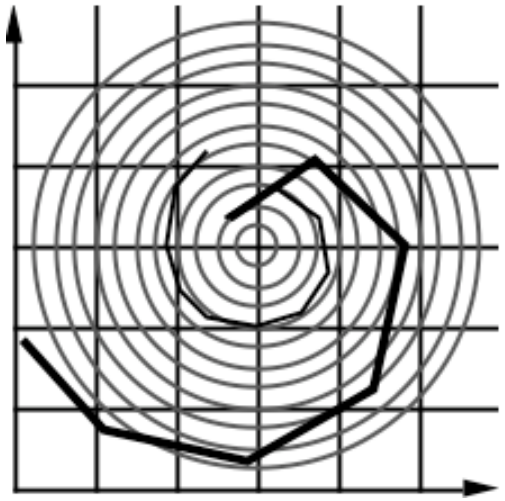
- Betrachte die DGL:

$$\dot{\mathbf{x}}(t) = \begin{pmatrix} -x_1 \\ x_2 \end{pmatrix}$$

- Exakte Lösung:

$$\mathbf{x}(t) = \begin{pmatrix} r\cos(t+\phi) \\ r\sin(t+\phi) \end{pmatrix}$$

- Euler läuft in Spiralen nach außen, egal wie klein die Schrittweite gewählt wird!

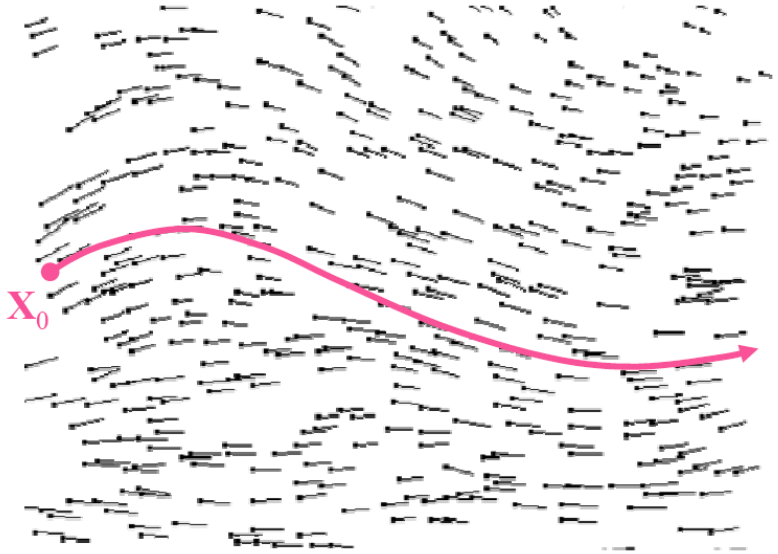


Veranschaulichung von DGLs

- Die allgemeine Form einer DGL (hier):

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t)$$

- Visualisiere **f** als Vektorfeld:



  - Achtung: dieses Vektorfeld kann sich mit *t* ändern!

- Lösung eines Anfangswertproblems = Pfad durch dieses Feld

- Runge-Kutta 2-ter Ordnung:
  - Idee: approximiere $\mathbf{f}(\mathbf{x}(t), t)$ durch quadratische Funktion, die an der Stelle $\mathbf{x}(t)$, $\mathbf{x}(t+\frac{1}{2}\Delta t)$ und $\mathbf{x}(t+\Delta t)$ definiert ist
  - Der Integrator:

$$\mathbf{a}_1 = \mathbf{v}^t \qquad \mathbf{a}_2 = \frac{1}{m}\mathbf{f}(\mathbf{x}^t, \mathbf{v}^t)$$

$$\mathbf{b}_1 = \mathbf{v}^t + \tfrac{1}{2}\Delta t\mathbf{a}_2 \qquad \mathbf{b}_2 = \frac{1}{m}\mathbf{f}\left(\mathbf{x}^t + \frac{1}{2}\Delta t\mathbf{a}_1, \mathbf{v}^t + \frac{1}{2}\Delta t\mathbf{a}_2\right)$$

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \Delta t\mathbf{b}_1 \qquad \mathbf{v}^{t+1} = \mathbf{v}^t + \Delta t\mathbf{b}_2$$

- Runge-Kutta 4-ter Ordnung:
  - Der Standard-Integrator unter den expliziten Integrations-Schemata
  - Benötigt 4 Funktionsauswertungen (Kräfte berechnen) pro Zeitschritt
  - Konvergenzordnung: $e(\Delta t) = O(\Delta t^4)$

Verlet-Integration

- Alternative Methode zur Steigung der Konvergenzordnung: verwende Werte aus der Vergangenheit
- Verlet: verwendet $\mathbf{x}(t - \Delta t)$
- Herleitung:
  - Taylor-Reihe in beide Zeitrichtungen entwickeln:

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t\dot{\mathbf{x}}(t) + \frac{1}{2}\Delta t^2\ddot{\mathbf{x}}(t) + \frac{1}{6}\Delta t^3\dddot{\mathbf{x}}(t) + O(\Delta t^4)$$

$$\mathbf{x}(t-\Delta t) = \mathbf{x}(t) - \Delta t\dot{\mathbf{x}}(t) + \frac{1}{2}\Delta t^2\ddot{\mathbf{x}}(t) - \frac{1}{6}\Delta t^3\dddot{\mathbf{x}}(t) + O(\Delta t^4)$$

- Addieren:

$$\mathbf{x}(t+\Delta t)+\mathbf{x}(t-\Delta t)=2\mathbf{x}(t)+\Delta t^2\,\ddot{\mathbf{x}}(t)+O(\Delta t^4)$$

$$\mathbf{x}(t+\Delta t)=2\mathbf{x}(t)-\mathbf{x}(t-\Delta t)+\Delta t^2\,\ddot{\mathbf{x}}(t)+O(\Delta t^4)$$

- Initialisierung:

$$\mathbf{x}(\Delta t)=\mathbf{x}(0)+\Delta t\mathbf{v}(0)+\frac{1}{2}\Delta t^2\left(\frac{1}{m}\mathbf{f}(\mathbf{x}(0),\mathbf{v}(0))\right)$$

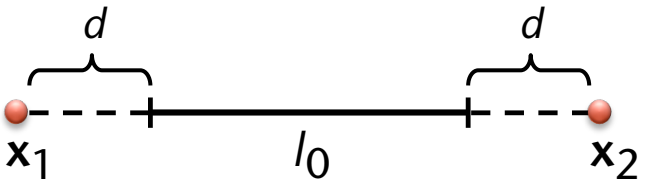
Constraints

- Großer Vorteil gegenüber Euler & Runge-Kutta:
  man kann sehr leicht Constraints behandeln
- Definition: Constraint = Einschränkung der Position eines oder mehrer Massenpunkte
- Beispiele:
  1. Ein Massenpunkt darf nicht in ein Hindernis eindringen
  2. Der Abstand zwischen zwei Massenpunkten soll konstant sein, oder ≤ bestimmter Abstand

- Verfahren anhand eines Beispiels:
  - Constraint:

$$\|\mathbf{x}_1 - \mathbf{x}_2\| \stackrel{!}{=} l_0$$

1. Führe einen Verlet-Integrationsschritt durch → $\tilde{\mathbf{x}}^{t+1}$
2. Enforce Constraint:



$$\mathbf{x}_1^{t+1} = \tilde{\mathbf{x}}_1^{t+1} + \frac{1}{2}\mathbf{r}_{12} \cdot \left(\|\tilde{\mathbf{x}}_2^{t+1} - \tilde{\mathbf{x}}_1^{t+1}\| - l_0 \right)$$

$$\mathbf{x}_2^{t+1} = \tilde{\mathbf{x}}_2^{t+1} - \underbrace{\frac{1}{2}\mathbf{r}_{12} \cdot \left(\|\tilde{\mathbf{x}}_2^{t+1} - \tilde{\mathbf{x}}_1^{t+1}\| - l_0 \right)}_{d}$$

- Problem, falls mehrere Constraints denselben Massepunkt einschränken sollen
  - Verwende Contraint-Algorithmen

Implizite Integration

- Bisherige Schemata sind nur *conditionally stable*
  - D.h.: Nur stabil für Δt in einen bestimmten Bereich
  - Dieser Bereich hängt von der Steifigkeit der Federn ab
- Ziel: *unconditionally stable*
- Eine Möglichkeit: implizite Euler-Integration

explizit

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \Delta t \mathbf{v}_i^t$$

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \Delta t \frac{1}{m} \mathbf{f}(\mathbf{x}^t)$$

implizit

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \Delta t \mathbf{v}_i^{t+1}$$

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \Delta t \frac{1}{m} \mathbf{f}(\mathbf{x}^{t+1})$$

- Wir haben jetzt ein System von nicht-linearen, algebraischen Gleichungen, mit $\mathbf{x}^{t+1}$ und $\mathbf{v}^{t+1}$ als Unbekannte auf beiden Seiten
  → implizite Integration

Lösungsmethode

- Schreibe das gesamte Feder-Masse-System mit Vektoren:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} = \begin{pmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{21} \\ x_{22} \\ \vdots \\ x_{n3} \end{pmatrix} \quad \mathbf{v} = \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n \end{pmatrix} = \begin{pmatrix} v_{11} \\ v_{12} \\ v_{13} \\ v_{21} \\ v_{22} \\ \vdots \\ v_{n3} \end{pmatrix} \quad \mathbf{f}(\mathbf{x}) = \begin{pmatrix} \mathbf{f}_1(x) \\ \vdots \\ \mathbf{f}_n(x) \end{pmatrix}$$

$$\mathbf{f}_i = \begin{pmatrix} f_{i1}(x) \\ f_{i2}(x) \\ f_{i3}(x) \end{pmatrix} \quad M_{3n\times 3n} = \begin{pmatrix} m_1 & & & & & & & & \\ & m_1 & & & & & & & \\ & & m_1 & & & & & & \\ & & & m_2 & & & & & \\ & & & & m_2 & & & & \\ & & & & & \ddots & & & \\ & & & & & & m_n & & \\ & & & & & & & m_n & \\ & & & & & & & & m_n \end{pmatrix}$$

- Schreibe die vielen impliziten Gleichungen als ein großes Gleichungssystem um :

$$M\mathbf{v}^{t+1} = M\mathbf{v}^{t} + \Delta t\mathbf{f}(\mathbf{x}^{t+1}) \quad (1)$$

$$\mathbf{x}^{t+1} = \mathbf{x}^{t} + \Delta t\,\mathbf{v}^{t+1} \quad (2)$$

- Einsetzen von (2) in (1) ergibt:

$$M\mathbf{v}^{t+1} = M\mathbf{v}^{t} + \Delta t\,\mathbf{f}(\,\mathbf{x}^{t} + \Delta t\mathbf{v}^{t+1}\,)$$

- Die Taylor-Reihe für $\mathbf{f}$ ist:

$$\mathbf{f}(\mathbf{x}^{t} + \Delta t\,\mathbf{v}^{t+1}) = \mathbf{f}(\,\mathbf{x}^{t}) + \frac{\partial}{\partial \mathbf{x}}\,\mathbf{f}(\,\mathbf{x}^{t}) \cdot (\Delta t\,\mathbf{v}^{t+1}) + O((\,\Delta t\,\mathbf{v}^{t+1})^2)$$

- Einsetzen:

$$
\begin{aligned}
M\,\mathbf{v}^{t+1} &= M\,\mathbf{v}^{t} + \Delta t\left(\mathbf{f}(\mathbf{x}^{t}) + \underbrace{\frac{\partial}{\partial x}\mathbf{f}(x^{t})}_{K}\cdot\left(\Delta t\,\mathbf{v}^{t+1}\right)\right) \\
&= M\,\mathbf{v}^{t} + \Delta t\,\mathbf{f}(\mathbf{x}^{t}) + \Delta t^{2}\,K\,\mathbf{v}^{t+1}
\end{aligned}
$$

- K ist die Jacobi-Matrix (= Ableitung):

$$
K = \begin{pmatrix}
\frac{\partial}{\partial x_{11}} f_{11} & \frac{\partial}{\partial x_{12}} f_{11} & \cdots & \frac{\partial}{\partial x_{n3}} f_{11} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial}{\partial x_{11}} f_{n3} & \cdots & \cdots & \frac{\partial}{\partial x_{n3}} f_{n3}
\end{pmatrix}
$$

  - Heißt tangent stiffnes matrix
    - (Die normale Steifigkeitsmatrix wird im Gleichgewichtszustand ausgewertet; hier aber an einer beliebigen, aktuellen Position des Systems.

- Terme umordnen:

$$\left(M - \Delta t^{\,2}\, K\right) \mathbf{v}^{t+1} = M\, \mathbf{v}^{t} + \Delta t\, \mathbf{f}(\mathbf{x}^{t})$$

- Das ist von der Form:

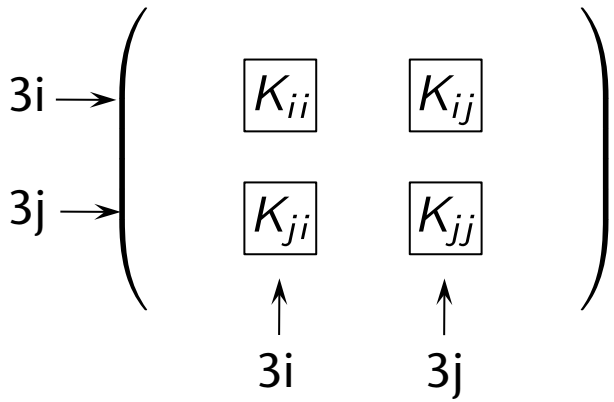
$$A\, \mathbf{v}^{t+1} = \mathbf{b}$$

$$\text{mit} \quad A \in \mathbb{R}^{3n \times 3n}, \quad b \in \mathbb{R}^{3n}$$

- Löse dieses LGS mittels einer iterativen Methode
  - Denn: A ändert sich in jedem Frame

Berechnung der Steifigkeitsmatrix

- Die Anatomie der Matrix K :
  - Eine Feder (i, j) addiert folgende vier 3x3 Untermatrizen zu K :



  - Die Matrix K_{ij} entsteht durch die Ableitung von $\mathbf{f}_i$ nach $\mathbf{x}_j = (x_{j1}, x_{j2}, x_{j3})$:

$$K_{ij} = \begin{pmatrix} \frac{\partial}{\partial x_{j1}} f_{i1} & \frac{\partial}{\partial x_{j2}} f_{i1} & \frac{\partial}{\partial x_{j3}} f_{i1} \\ \vdots & & \vdots \\ \frac{\partial}{\partial x_{j1}} f_{i3} & \cdots & \frac{\partial}{\partial x_{j3}} f_{i3} \end{pmatrix}$$

  - Betrachte im folgenden nur f^s (Federkraft)

- Zunächst K_{ii}:

$$
\begin{aligned}
K_{ii} &= \frac{\partial}{\partial \mathbf{x}_i} f_i(\mathbf{x}_i, \mathbf{x}_j) \\
&= k_s \frac{\partial}{\partial \mathbf{x}_i} \left((\mathbf{x}_j - \mathbf{x}_i) - l_0 \frac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|} \right) \\
&= k_s \left(-I - l_0 \frac{-I \cdot \|\mathbf{x}_j - \mathbf{x}_i\| - (\mathbf{x}_j - \mathbf{x}_i) \cdot 2 \frac{(\mathbf{x}_j - \mathbf{x}_i)^\top}{\|\mathbf{x}_j - \mathbf{x}_i\|}}{\|\mathbf{x}_j - \mathbf{x}_i\|^2} \right) \\
&= k_s \left(-I + l_0 \frac{1}{\|\mathbf{x}_j - \mathbf{x}_i\|} I + \frac{2 l_0}{\|\mathbf{x}_j - \mathbf{x}_i\|^3} (\mathbf{x}_j - \mathbf{x}_i)(\mathbf{x}_j - \mathbf{x}_i)^\mathsf{T} \right)
\end{aligned}
$$

- Aus einigen Symmetrien folgt:
  - $K_{ij} = \frac{\partial}{\partial \mathbf{x}_j} f_i(\mathbf{x}_i, \mathbf{x}_j) = -K_{ii}$
  - $K_{jj} = \frac{\partial}{\partial x_j} f_j(\mathbf{x}_i, \mathbf{x}_j) = \frac{\partial}{\partial \mathbf{x}_j}(-\mathbf{f}_i(\mathbf{x}_i, \mathbf{x}_j)) = K_{ii}$
  - $K_{ji} = K_{ij}$

- Zur Erinnerung:
  - $\left(\frac{f}{g}\right)' = \frac{f'g - fg'}{g^2}$
  - $\frac{\partial}{\partial \mathbf{x}} \|\mathbf{x}\| = \frac{\partial}{\partial \mathbf{x}} \left(\sqrt{x_1^2 + x_2^2 + x_3^2} \right) = 2 \frac{\mathbf{x}^T}{\|\mathbf{x}\|}$

Lösungsverfahren

- Setze $K = 0$
- Für jede Feder (i,j) berechne K_{ii}, K_{ij}, K_{ji}, K_{jj} und akkumuliere zu K an den richtigen Stellen
- Berechne $\mathbf{b} = M\,\mathbf{v}^t + \Delta t\, f(\mathbf{x}^t)$
- Löse das LGS $\rightarrow$ $\mathbf{v}^{t+1}$
- Berechne $\mathbf{x}^{t+1} = \mathbf{x}^t + \Delta t\,\mathbf{v}^{t+1}$

Vor- und Nachteile

- Explizite Integration:

  + Sehr einfach zu programmieren

  - kleine Schrittweite nötig

  - Steife Federn funktionieren nicht gut

  - Kräfte werden nur um eine Feder pro Schritt propagiert

- Implizite Integration:

  + Unconditionally stable

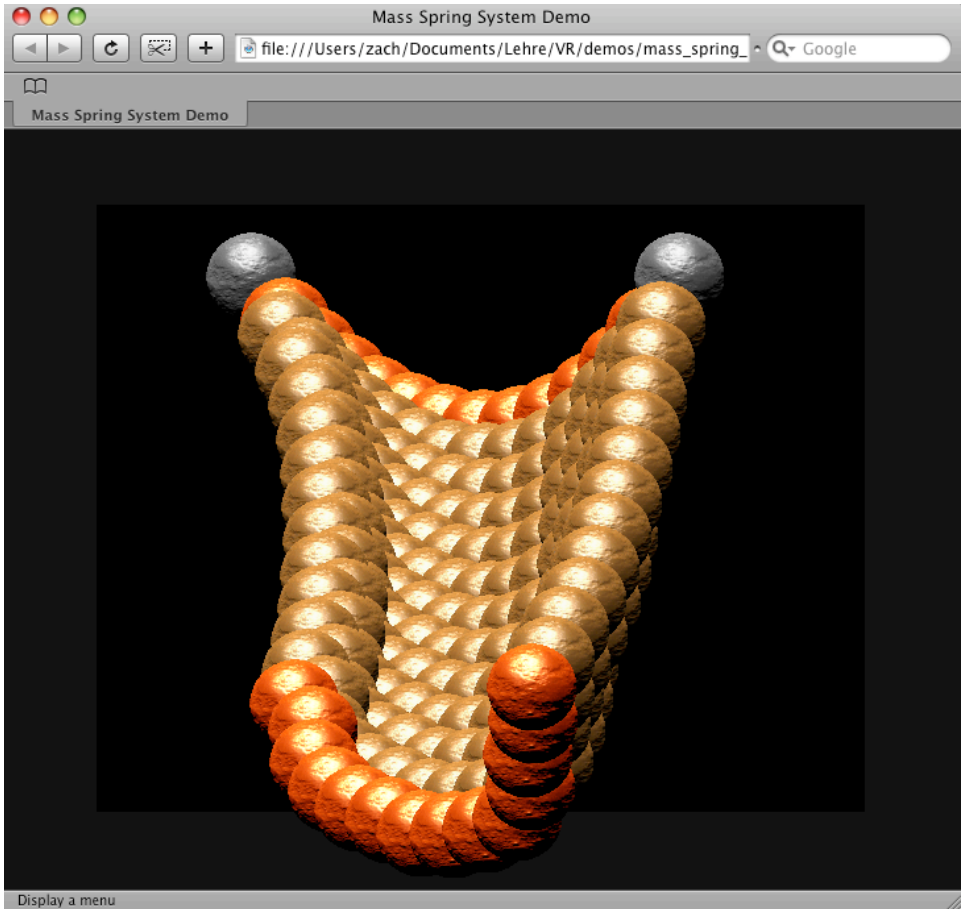
  + Steife Federn werden besser handhabbar

  + Globaler Solver → Kräfte werden schon bei einem Simulationsschritt durch das ganze System propagiert

  - Große Schrittweiten nötig, da ein Schritt sehr teuer (und in Wahrheit schon aus vielen Einzelschritten besteht)

  - Unerwünschte Dämpfung durch das Integrationsverfahren selbst

Demo



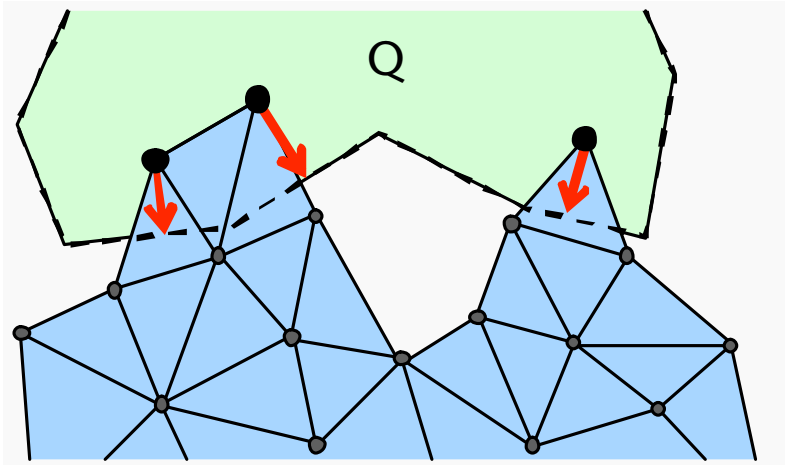
http://www.dhteumeuleu.com/dhtml/v-grid.html

Kollisionserkennung

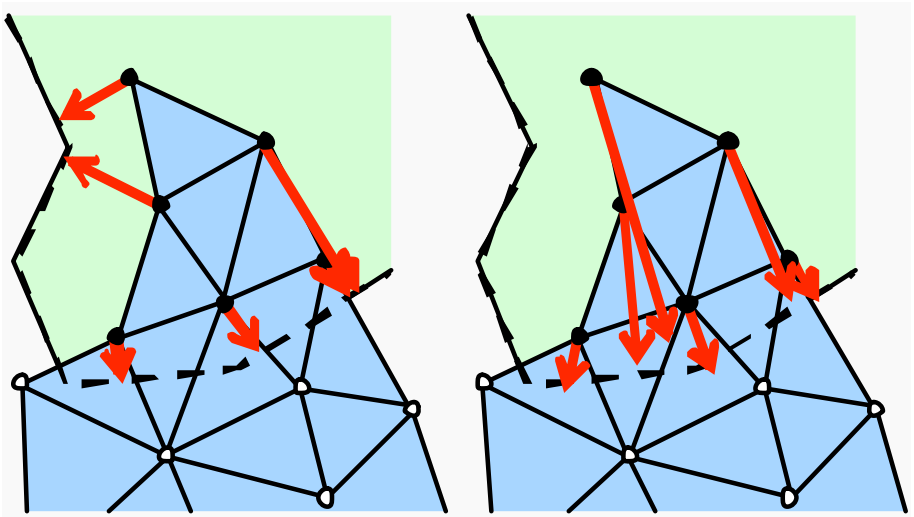
- Sortiere die Tetraeder in ein 3D Gitter (Hash-Tabelle!) ein
- Bei Kollision in der Hash-Tabelle:
  - Führe exakten Schnitttest zwischen 2 Tetraedern durch

Kollisionsantwort (collision response)

- Aufgabenstellung: gegeben zwei kollidierende Objekte P und Q (Tetraeder-Meshes) — welches ist die Rückstellkraft (penalty force)?
- Naiver Ansatz:
  - Berechne für jeden eingedrungen Massepunkt von P die kleinste Distanz zur Oberfläche von Q
    → Kraft (Betrag + Richtung)
  - Problem:
    - unplausible Kräfte (implausibel?)
    - "Durchtunneln" (s. a. Force-Feedback-Kapitel)

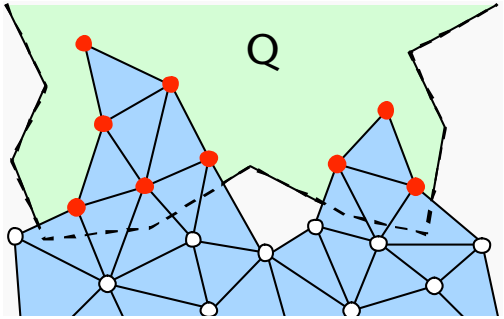


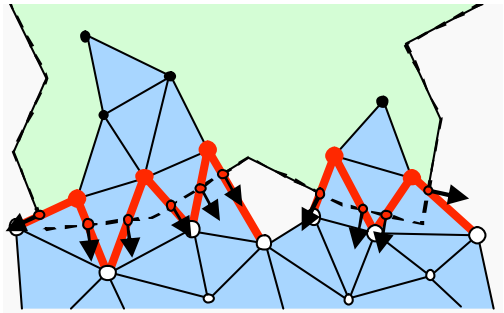
- Beispiele:



Konsistente Penalty Forces

1. Phase: identifiziere alle eindringenden Punkte von P
2. Phase: bestimme alle schneidenden Kanten von P
   - Berechne zu jeder solchen Schnittkante den exakten Schnittpunkt $\mathbf{x}_i$
   - Berechne zu jedem Schnittpunkt eine Normale $\mathbf{n}_i$
     - Z.B. mittels baryzentrischer Interpolation der Vertexnormalen von Q





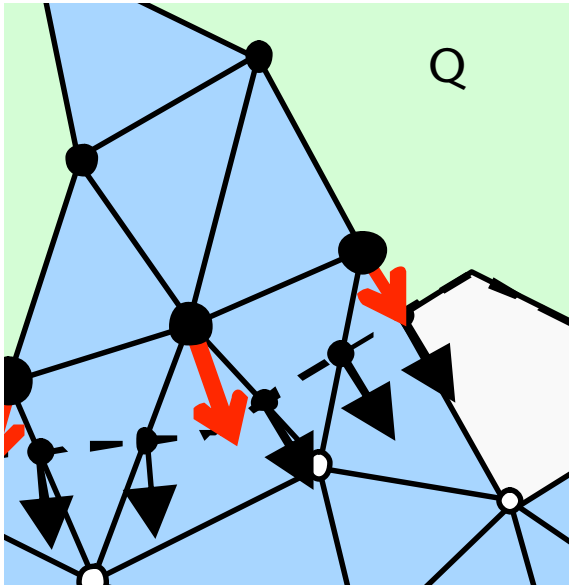
3. Phase: berechne ungefähre Kraft für "Randpunkte"

- Randpunkt = eingedrungener Punkt **p** inzident zu einer Schnittkante
- Beobachtung: ein Randpunkt kann zu mehreren Schnittkanten inzident sein
- Eindringtiefe = gewichtete Summe

$$d(\mathbf{p}) = \frac{\sum_{i=1}^{k} \omega(\mathbf{x}_i, \mathbf{p})\,(\mathbf{x}_i - \mathbf{p})\cdot\mathbf{n}_i}{\sum_{i=1}^{k} \omega(\mathbf{x}_i, \mathbf{p})}$$

wobei $d(\mathbf{p})$ = approx. Eindringtiefe von Massepunkt **p**, $\mathbf{x}_i$ = Schnittpunkt der Schnittkante inzident zu **p**, $\mathbf{n}_i$ = Normale zur Oberfläche von Q im Schnittpunkt $\mathbf{x}_i$,

und $\omega(\mathbf{x}_i, \mathbf{p}) = \frac{1}{\|\mathbf{x}_i - \mathbf{p}\|}$



- Richtung der Kraft für Randpunkte:

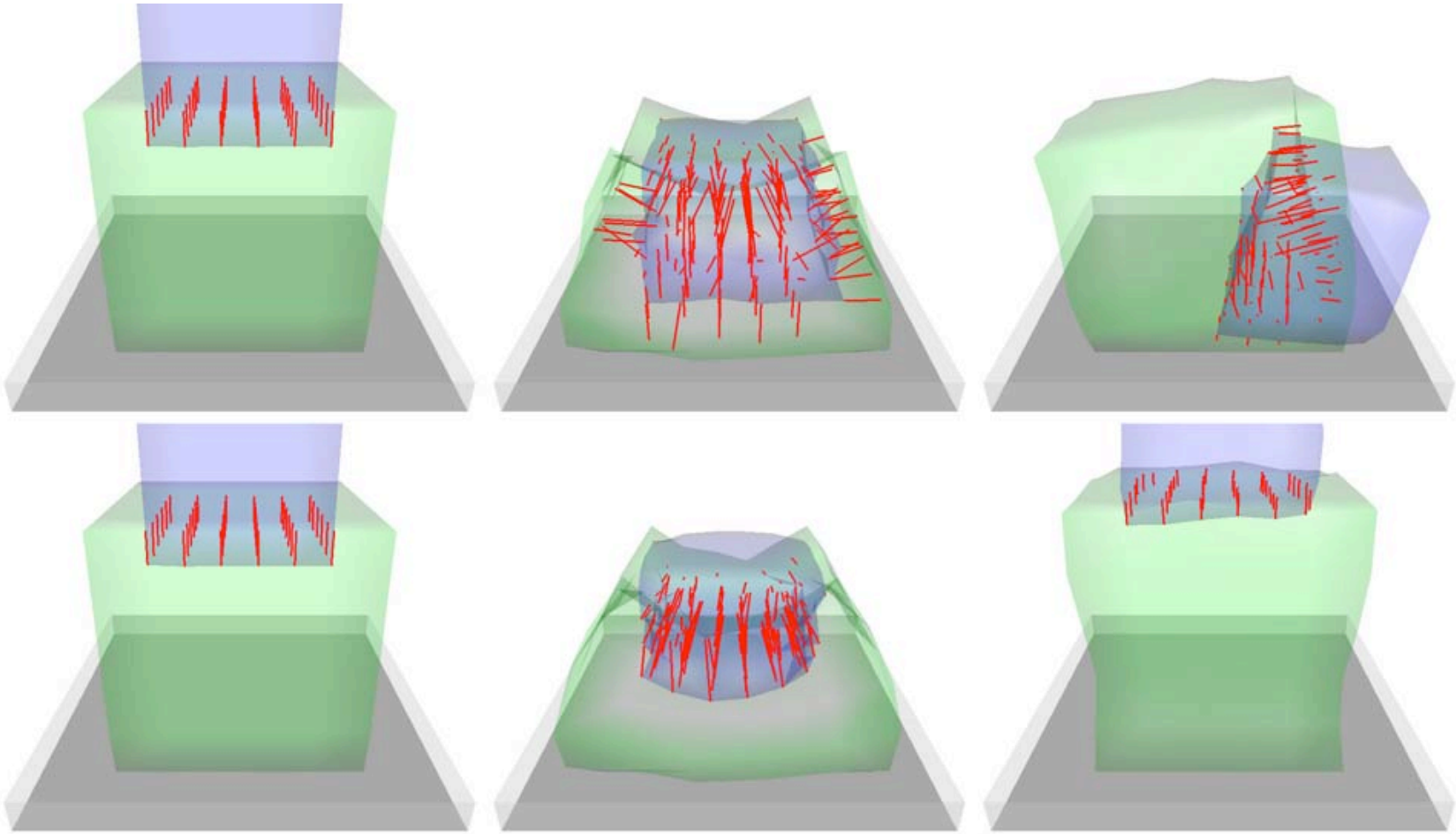
$$\hat{\mathbf{r}}(\mathbf{p}) = \frac{\sum_{i=1}^{k} \omega(\mathbf{x}_i, \mathbf{p})\, \mathbf{n}_i}{\sum_{i=1}^{k} \omega(\mathbf{x}_i, \mathbf{p})} \qquad \mathbf{r}(\mathbf{p}) = \hat{\mathbf{r}}(\mathbf{p})^0$$

4. Phase: Propagation mittels breadth-first traversal durch das Tetraeder-Mesh

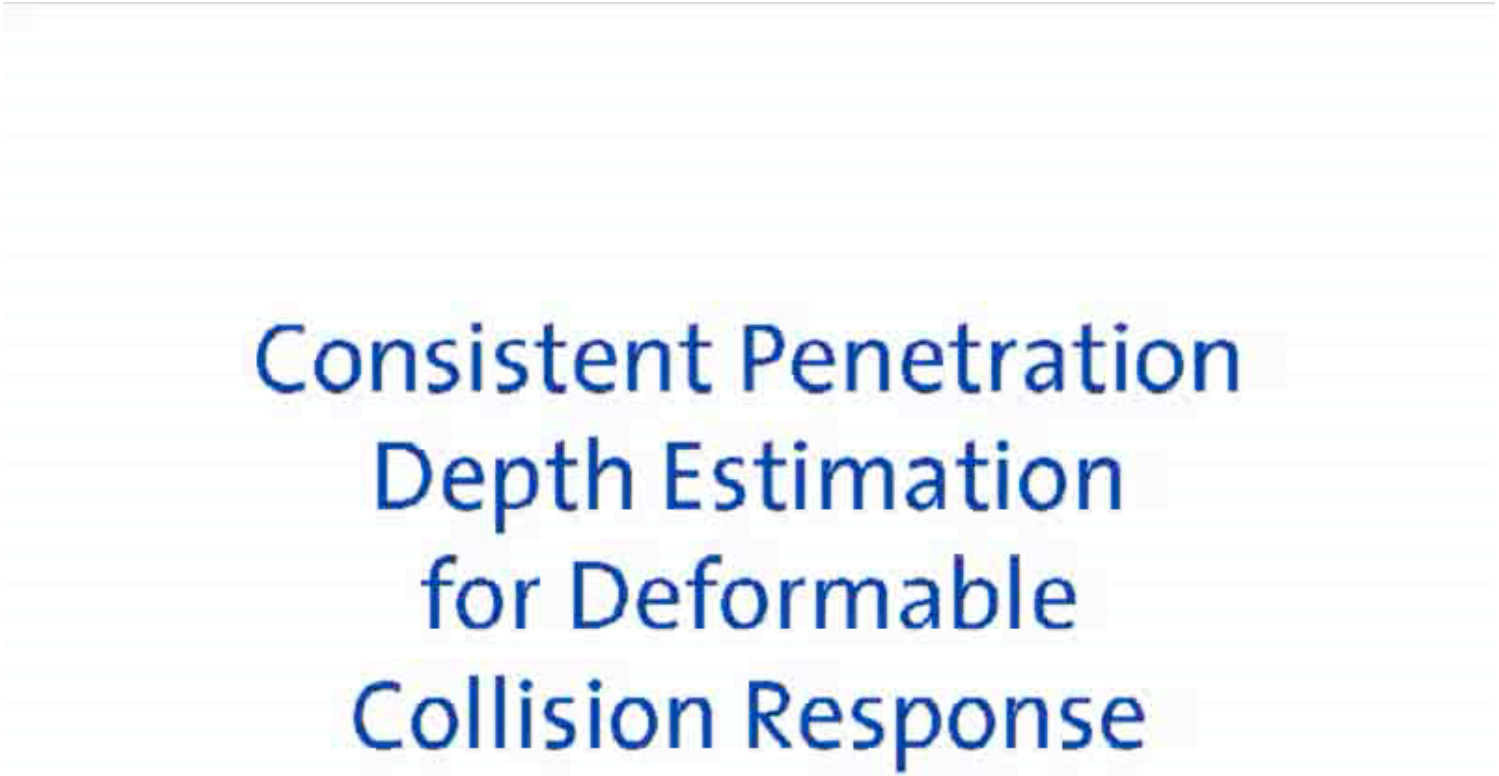
$$d(\mathbf{p}) = \frac{\sum_{i=1}^{k} \omega(\mathbf{p}_i, \mathbf{p})\big((\mathbf{p}_i - \mathbf{p}) \cdot \mathbf{r}_i + d(\mathbf{p}_i)\big)}{\sum_{i=1}^{k} \omega(\mathbf{x}_i, \mathbf{p})}$$

wobei $\mathbf{p}_i$ = schon besuchter eindringender Punkt von P, $\mathbf{p}$ = noch nicht besuchter Punkt, $\mathbf{r}_i$ = Richtung der geschätzten penalty force in Punkt $\mathbf{p}_i$.

Viualisierung



Video



http://cg.informatik.uni-freiburg.de