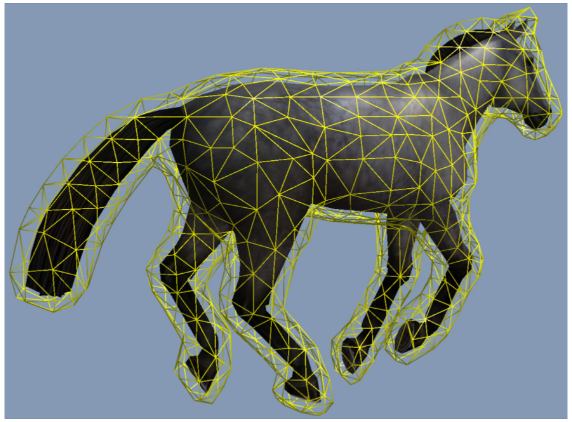
Virtual Reality & Physically-Based Simulation

Mass-Spring-Systems



G. Zachmann

University of Bremen, Germany

cgvr.cs.uni-bremen.de

Newton's Laws

1. Law (law of inertia):

> A body, which no forces act upon, continues to move with constant velocity.

- A resting body is just a special case of this law.

2. Law (law of action):

> If a force **F** acts on a body with mass *m* , then the body accelerates, and its acceleration is given by
> $$\mathbf{F} = m \cdot \mathbf{a}$$

- In other words: force and acceleration are proportional to each other; (the proportionality factor happens to be *m)*. In aprticular, both force and acceleration have the same direction.

3. Law (law of reaction):

If a force **F**, that acts on a body, is extended to another body, Then the opposite force –**F** acts on that other body.

- In school, you learn: "action= reaction"

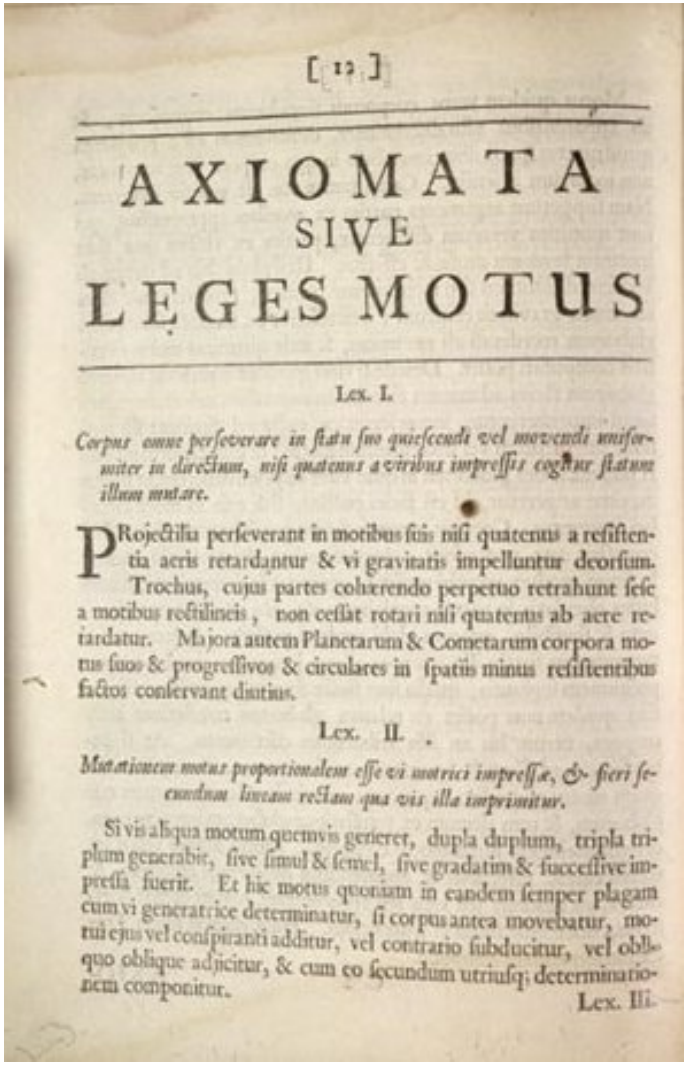
4. Law (law of superposition):

If a number offorces $\mathbf{F}_1, \ldots, \mathbf{F}_n$ act on a point or body, then they can be accumulated by vector addition yielding one resulting force:

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

Historical Digression

- Newton published these laws in his original book
  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.*



[12]

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.

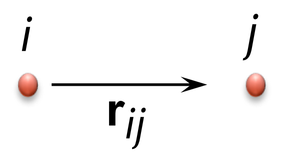
Si vis aliqua motum quemvis generet, dupla duplum, tripla triplum generabit, ſive ſimul & ſemel, ſive gradatim & ſucceſſive impreſſa fuerit. Et hic motus quoniam in eandem ſemper plagam cum vi generatrice determinatur, ſi corpus antea movebatur, motui ejus vel conſpiranti additur, vel contrario ſubducitur,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

Lex. III.

- Definition:
  A mass-spring system is a system consisting of:
  1. A set of point masses m_i with positions $\mathbf{x}_i$ and velocities $\mathbf{v}_i$, i = 1…N ;
  2. A set of springs $s_{ij} = (i, j, k_s, k_d)$, where s_{ij} connects masses *i* und *j,* with rest length l_0 , spring constant k_s (= stiffness) and the damping coeffizient k_d
- Advantages:
  - Very easy to program
  - Ideally suited to study different kinds of solving methods
  - *Ubiquitous in games* (cloths, capes, sometimes also for deformable objects)
- Disadvantages:
  - Some parameters (in particular the spring constants) are not obvious, i.e., difficult to derive
  - No built-in volumetric effects (e.g., preservation of volume)

A Single Spring (Plus Damper)

- Given: masses m_i and m_j with positions $\mathbf{x}_i$, $\mathbf{x}_j$
- Let $\mathbf{r}_{ij} = \dfrac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|}$

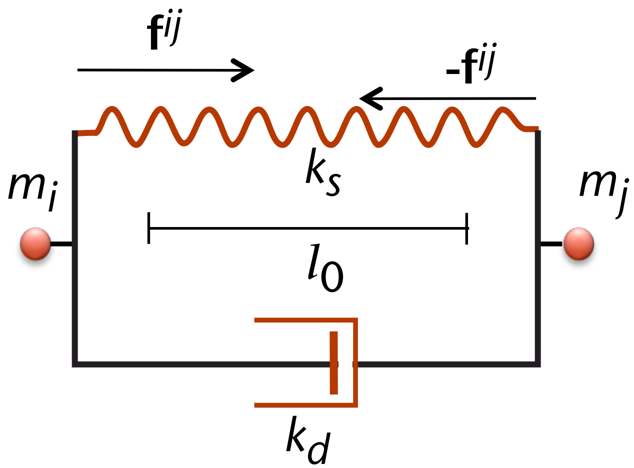


- The force between particles i and j :

  1. Force extended by spring (Hooke's law):

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

     acts on mass m_i in direction of m_j



  2. Force extended by damper : $\mathbf{f}_d^{ij} = k_d \left((\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{r}_{ij} \right) \mathbf{r}_{ij}$

  3. Sum of forces : $\mathbf{f}^{ij} = \mathbf{f}_s^{ij} + \mathbf{f}_d^{ij}$

  4. Force on m_j : $\mathbf{f}^{ji} = -\mathbf{f}^{ij}$

- Notice: from (4) follows that the momentum is conserved
  - I.e., the kinetic energy is conserved
  - Momentum = velocity × mass = $\mathbf{v} \cdot m$
- Note on terminology:
  - German "Kraftstoß" = English "Impulse" = force × time
  - German "Impuls" = English "momentum" = force × mass
- Alternative Federkraft: $\mathbf{f}_s^{ij} = k_s \mathbf{r}_{ij} \frac{\|\mathbf{x}_j - \mathbf{x}_i\| - l_0}{l_0}$
- A spring-damper element in reality:

Simulation of a Single Spring

- From Newton's law, we have: $\ddot{\mathbf{x}} = \frac{1}{m}\mathbf{f}$
- Convert differential equation (DE) of order 2 into DE of order 1:

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

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

- Initial values (boundary values): $\mathbf{v}(t_0) = \mathbf{v}_0\ ,\quad \mathbf{x}(t_0) = \mathbf{x}_0$
- "Simulation" = "Integration of DE's over time"
- By Taylor expansion we get:

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

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

→ This integration scheme is called **explicit Euler integration**

The Algorithm

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

loop forever:
   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
```

$\mathbf{f}^g$ = gravitational force

$\mathbf{f}^{coll}$ = penalty force exerted by collision (e.g., with obstacles)

- Advantages:
  - Can be implemented very easily
  - Fast execution per time step
- Disadvantages:
  - Stable only for very small time steps
    - Typically $\Delta t \approx 10^{-4} \ldots 10^{-3}$ sec!
  - With large time steps, additional energy is generated "out of thin air", until the system explodes ☺
  - Example: overshooting when simulating a single spring
  - Errors accumulate quickly

Example for the Instability of Euler Integration

- Consider the diferential equation

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

- The exact solution:

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

- Euler integration does this:

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

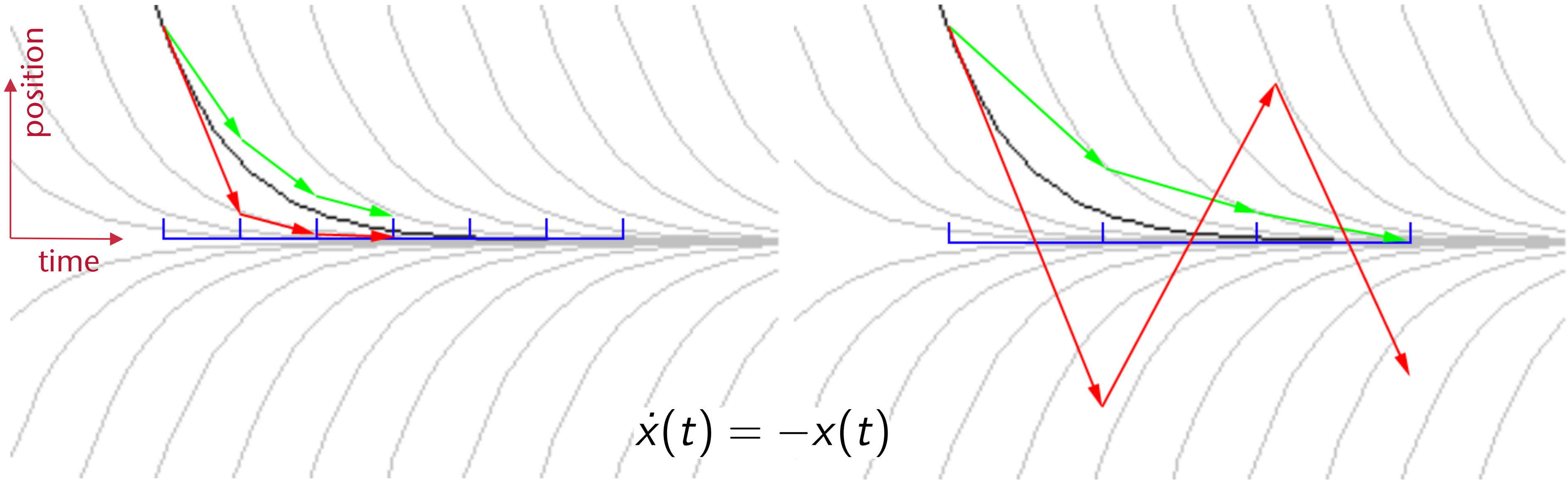
- Case $\Delta t > \frac{1}{k}$:

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

$\Rightarrow$ x^t oscillates about 0, but approaches 0 (hopefully)

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

- Visualization:



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

- Terminology: if k is large → the DE is called *"stiff "*
  - The stiffer the DE, the smaller Δt has to be

Visualization of Error Accumulation

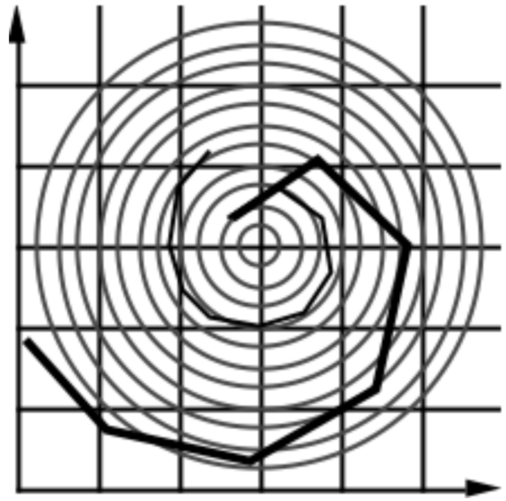
- Consider this DE:

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

- Exact solution:

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

- The solution by Euler integration moves in spirals outward, no matter how small Δt!
- Conclusion: Euler integration accumulates errors, no matter how small Δt!

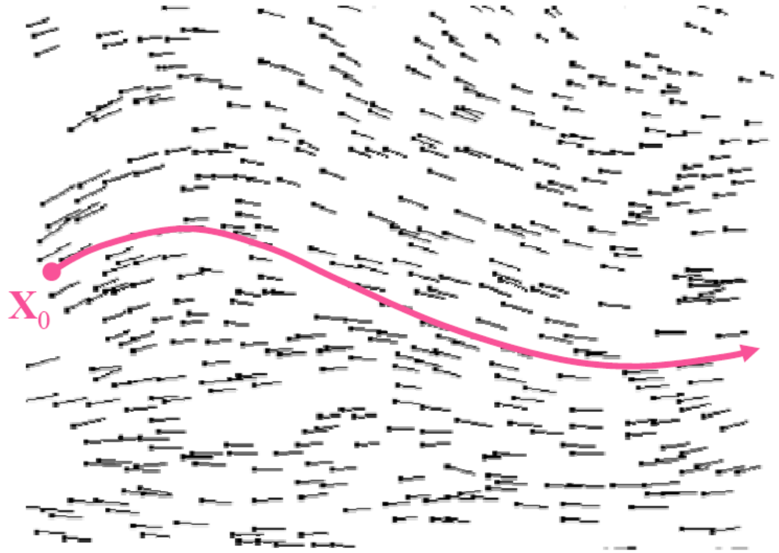


Visualization of Differential Equations

- The general form of a DE:

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

- Visualization of **f** as a vector field:



  - Notice: this vector field can vary over time!

- Solution of a boundary value problem = path through this field

- Runge-Kutta of order 2:
  - Idea: approximate **f**(**x**(t), t) by a quadratic function that is defined at positions **x**(t), **x**(t+ ½Δt) and **v**(t)
  - The integrator (w/o proof):

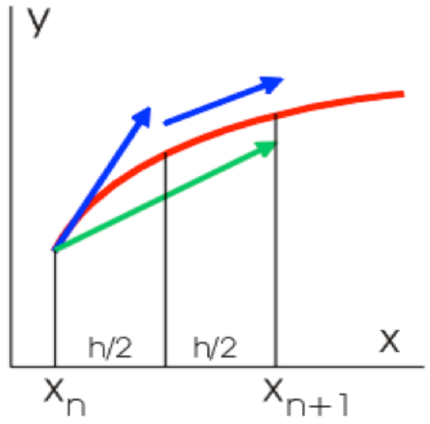
$$\mathbf{a}_1 = \mathbf{v}^t \qquad \mathbf{a}_2 = \frac{1}{m}\mathbf{f}(\mathbf{x}^t, \mathbf{v}^t)$$

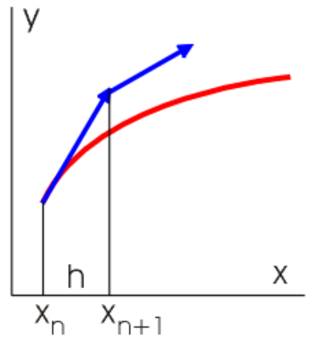
$$\mathbf{b}_1 = \mathbf{v}^t + \frac{1}{2}\Delta t \mathbf{a}_2 \qquad \mathbf{b}_2 = \frac{1}{m}\mathbf{f}\left(\mathbf{x}^t + \frac{1}{2}\Delta t \mathbf{a}_1, \mathbf{v}^t + \frac{1}{2}\Delta t \mathbf{a}_2\right)$$

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

- Runge-Kutta of order 4:
  - The standard integrator among the explicit integration schemata
  - Needs 4 function evaluations (i.e., force computations) per time step
  - Order of convergence is: $e(\Delta t) = O(\Delta t^4)$

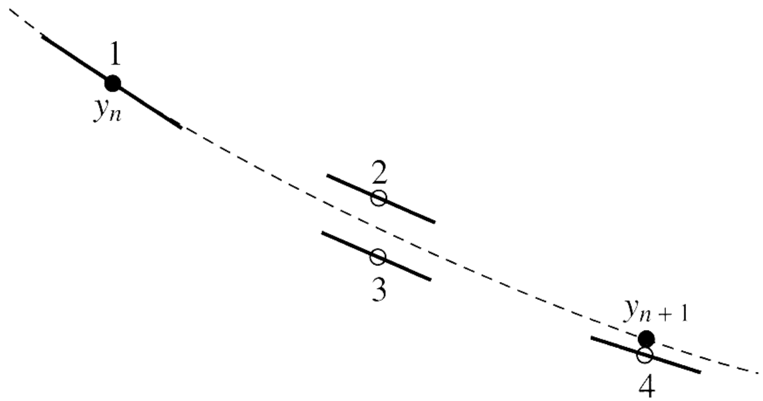
- Runge-Kutta of order 2:





Euler

- Runge-Kutta of order 4:



Verlet Integration

- A general, alternative method to increase the order of convergence: utilizes values from history
- Verlet: utilize $\mathbf{x}(t - \Delta t)$
- Derivation:
  - Develop the taylor series in both time directions:

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

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

- Add both:

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

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

- Initialization:

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

- Remark: the velocity does not occur any more!
  (at least, not explicitely)

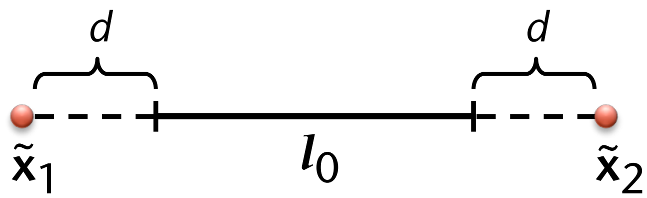
Constraints

- Big advantage of Verlet over Euler & Runge-Kutta: it is very easy to handle constraints
- Definition: constraint = some condition on the position of one or more mass points
- Examples:
  1. A point must not penetrate an obstacle
  2. The distance between two points must be constant, or distance must be $\leq$ some specific distance

- Examples:
  - Consider the constraint:

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

1. Perform one Verlet integration step $\rightarrow \quad \tilde{\mathbf{x}}^{t+1}$
2. Enforce the constraint:



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

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

- Problem: if several constraints are to constrain the same mass point, we need to employ constraint algorithms

Implicit Integration

- All integration schemes are only *conditionally stable*
  - I.e.: they are only stable for a specific range for Δt
  - This range depends on the stiffness of the springs
- Goal: *unconditionally stability*
- One option: implicit Euler integration

explicit

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

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

implicit

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

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

- Now we've got a system of non-linear, algebraic equations, with $\mathbf{x}^{t+1}$ and $\mathbf{v}^{t+1}$ as unknowns on both sides → implicit integration

Solution Method

- Write the whole spring-mass system with vectors:

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

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

- Write all the implicit equations as one big system of equations :

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

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

- Plug (2) into (1) :

$$M\mathbf{v}^{t+1} = M\mathbf{v}^{t} + \Delta t\, \mathbf{f}(\, \mathbf{x}^{t} + \Delta t\mathbf{v}^{t+1}\,) \quad (3)$$

- Expand **f** as Taylor series:

$$\mathbf{f}(\mathbf{x}^{t} + \Delta t\, \mathbf{v}^{t+1}) = \mathbf{f}(\,\mathbf{x}^{t}) + \frac{\partial}{\partial \mathbf{x}}\mathbf{f}(\,\mathbf{x}^{t}) \cdot (\Delta t\, \mathbf{v}^{t+1}) + O((\,\Delta t\, \mathbf{v}^{t+1})^2) \quad (4)$$

- Plug (4) into (3):

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

- K is the Jacobi-Matrix, i.e., the derivative of **f** (wrt. **x**):

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

  - K is called the tangent stiffness matrix
    - (The normal stiffness matrix is evaluated at the equilibrium of the system: here the matrix is evaluated at an arbitrary "position" of the system in phase space, hence the name "*tangent* ...")

- Reorder terms :

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

- Now, this has the form:

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

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

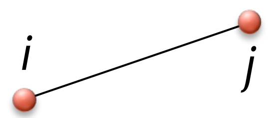
- Solve this system of linear equations with any of the iterative solvers
- Don't use a non-iterative solver, because
  - A changes with every frame (simulation step)
  - We can "warm start" the iterative solver with the solution as of last frame

Computation of the Stiffness Matrix

- First, understand the anatomy of matrix K :
  - A spring (i,j) adds the following four 3x3 block matrices to K :

$$
\begin{array}{r}
3i \longrightarrow \\
3j \longrightarrow
\end{array}
\begin{pmatrix}
 & \boxed{K_{ii}} & & \boxed{K_{ij}} & \\
 & \boxed{K_{ji}} & & \boxed{K_{jj}} &
\end{pmatrix}
$$

$$
\begin{array}{cc}
\uparrow & \uparrow \\
3i & 3j
\end{array}
$$



  - Matrix K_{ij} arises from the derivation of $\mathbf{f}_i = (f_{i1}, f_{i2}, f_{i3})$ wrt. $\mathbf{x}_j = (x_{j1}, x_{j2}, x_{j3})$:

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

  - In the following, consider only f^s (spring force)

- First of all, compute K_{ii}:

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

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

- From some symmetries, we can analogously derive:
    - $K_{ij} = \frac{\partial}{\partial \mathbf{x}_j} f_i(\mathbf{x}_i, \mathbf{x}_j) = -K_{ii}$
    - $K_{jj} = \frac{\partial}{\partial x_j} f_j(\mathbf{x}_i, \mathbf{x}_j) = \frac{\partial}{\partial \mathbf{x}_j}(-\mathbf{f}_i(\mathbf{x}_i, \mathbf{x}_j)) = K_{ii}$
    - $K_{ji} = K_{ij}$

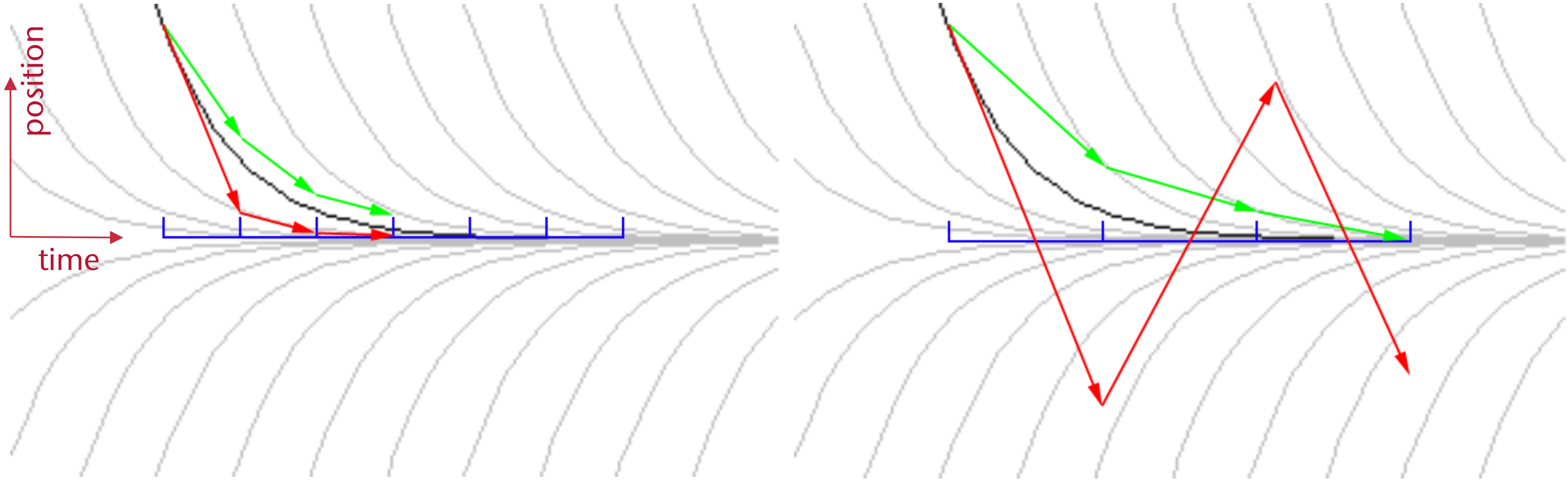
Overall Solution Algorithm

- Initialize $K = 0$
- For each spring (i,j) compute $K_{ii}, K_{ij}, K_{ji}, K_{jj}$ and accumulate it to K at the right places
- Compute $\mathbf{b} = M\,\mathbf{v}^t + \Delta t\, f(\mathbf{x}^t)$
- Solve the linear equation system $A\mathbf{v}^{t+1} = \mathbf{b} \;\rightarrow\; \mathbf{v}^{t+1}$
- Compute $\mathbf{x}^{t+1} = \mathbf{x}^t + \Delta t\,\mathbf{v}^{t+1}$

Advantages and Disadvantages

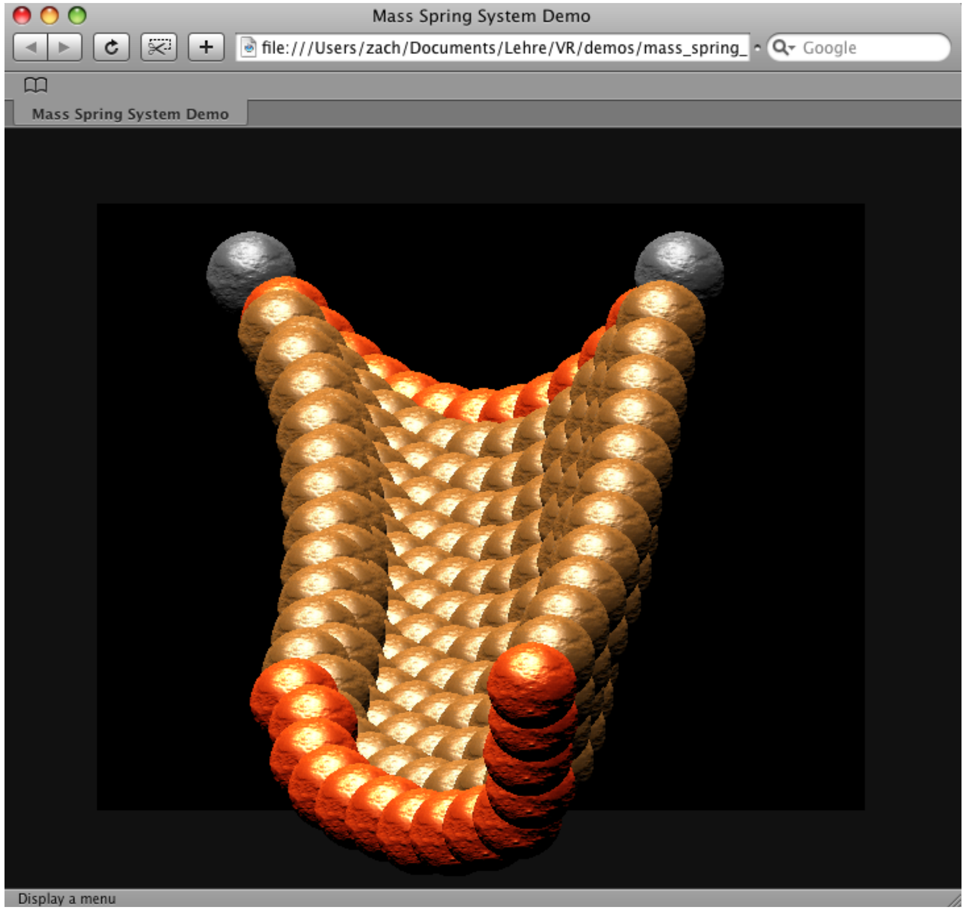
- Explicit integration:
  + Very easy to implement
  - Small step sizes needed
  - Stiff springs don't work very well
  - Forces are propagated only by one spring per time step
- Implicit Integration:
  + Unconditionally stable
  + Stiff springs work better
  + Globale solver → forces are being propagated throughut the whole pring-mass system with one time step
  - Large stime steps are needed, because one step is much more expensive (if real-time is needed)
  - The integration scheme introduces damping by itself (might be unwanted)

- Visualization of: $\dot{x}(t) = -x(t)$



- Informal Descripiton:
  - Explicit jumps forward blindly, based on current information
  - Implicit jumps backward and tries to find a future position such that the backwards jump arrives exactly at the current point (in phase space)

Demo

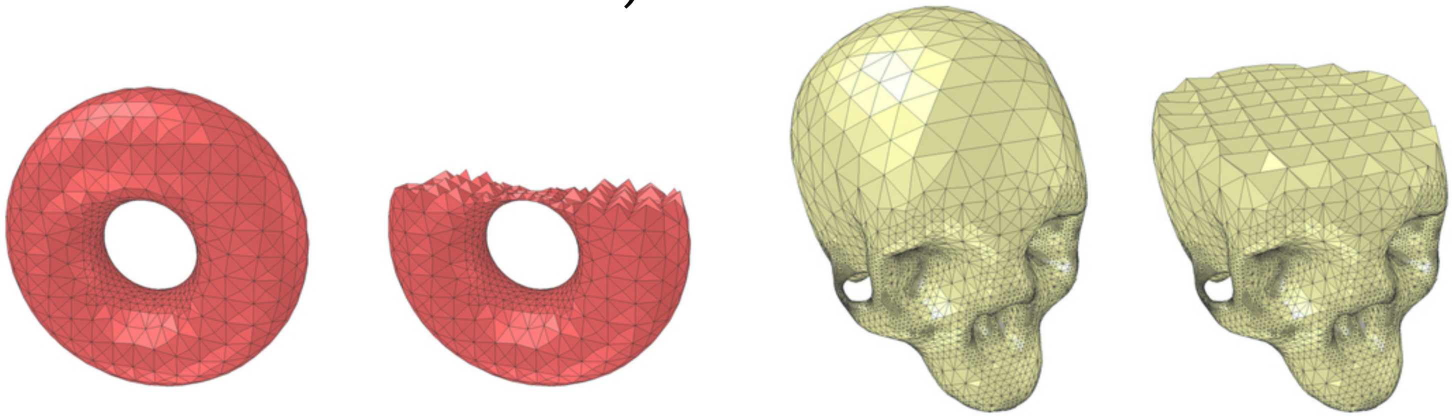


http://www.dhteumeuleu.com/dhtml/v-grid.html

Mesh Creation for Volumetric Objects

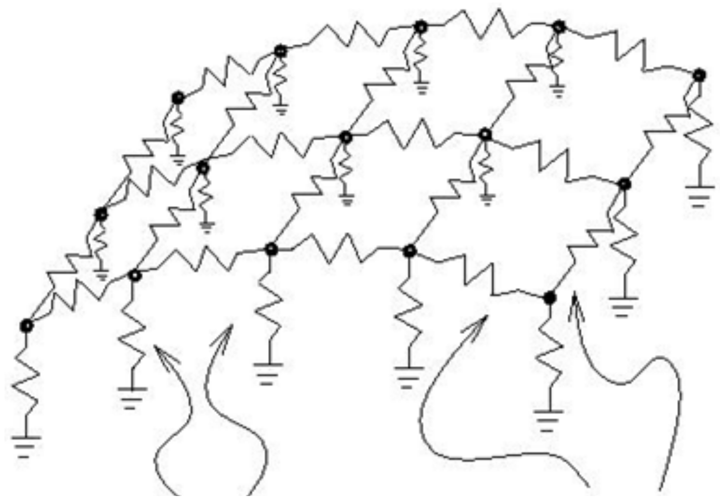
- How to create a mass-spring system for a volumetric model?
- Direct conversion of 3D (surface) geometry into spring-mass system does not yield good results:
  - Geometry has too high a complexity
  - Degenerate polygons
- Better (and still simple) idea:
  - Create a tetrahedron mesh out of the geometry (somehow)
  - Each vertex (node) of the tetrahedron mesh becomes a mass point, each edge a spring
  - Distribute the masses of the tetraeder (= density × volume) equally among the mass point

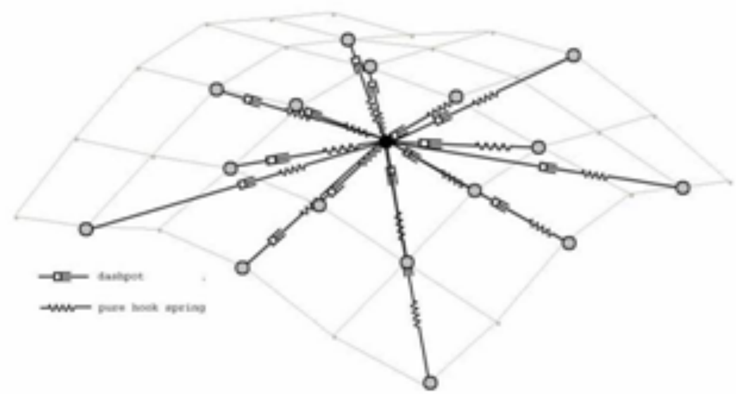
- Generation of the tetrahedron mesh (simple method):
  - Distribute a number of points uniformly (perhaps randomly) in the interior of the geometry (so called "Steiner points")
  - Dito for a sheet/band above the surface
  - Connect the points by Delaunay triangulation (see my "Geometric Data Structures for CG" course)



- Anchor the surface meshes within the tetraeder mesh:
  - Represent each vertex of the surface mesh by the *barycentric combination* of its surrounding tetrahedron vertices

- In addition (optionally):
  - Anchor the outer mass points (of the tetrahedron mesh) at (imaginary) walls
  - Introduce diagonal "struts" (Streben)



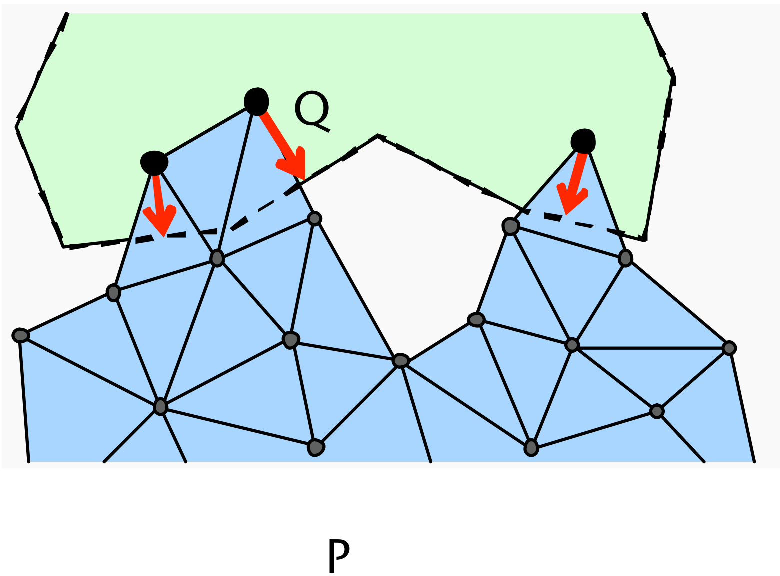


Collision Detection

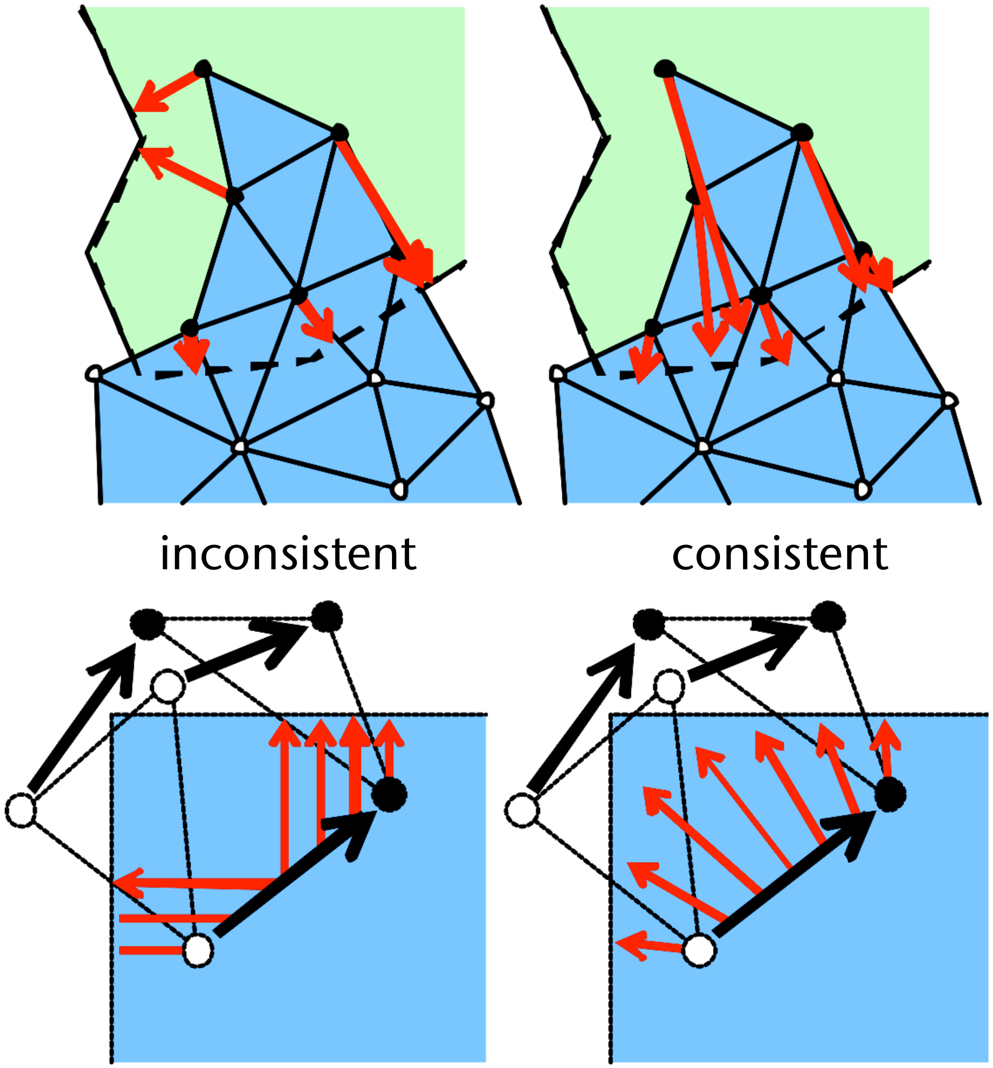
- Put all tetrahedra in a 3D grid (use a hash table!)
- In case of a collision in the hash table:
  - Compute exact intersection between the 2 involved tetrahedra

Collision Response

- Task: objects P and Q (= tetrahedral meshes) collide — what is the penalty force?
- Naïve approach:
  - For each mass point of P that has penetrated, compute its closest distance from the surface of Q → force (amount + direction)
- Problem:
  - Implausible forces
  - "Tunneling" (s. a. the chapter on force-feedback)

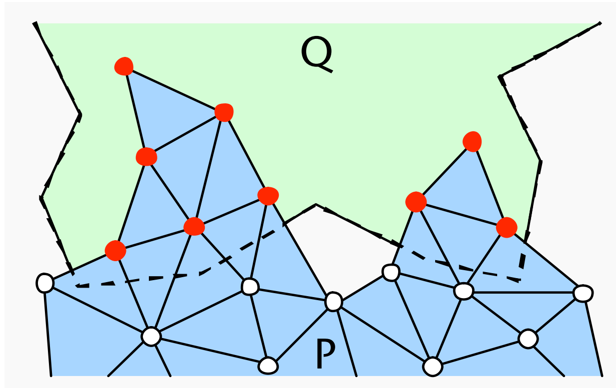


- Examples:

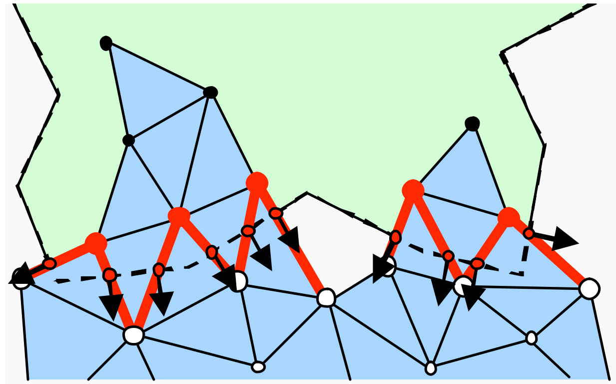


Consistente Penalty Forces

1. Phase: identify all points of P that penetrate Q

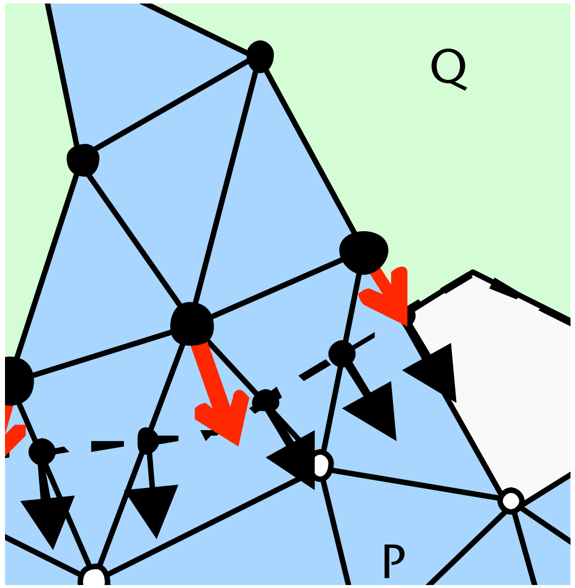


2. Phase: determine all edges of P that intersect the surface of Q
   - For each such edge, compute the exact intersection point $\mathbf{x}_i$
   - For each intersection point, compute a normal $\mathbf{n}_i$
     - E.g., by barycentric interpolation of the vertex normals of Q



3. Phase: compute the approximate force for border points

- Border point = a point **p** that penetrates Q and is incident to an intersecting edge
- Observation: a border point can be incident to several intersecting edges
- Set the penetration depth for point p to



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

where $d(\mathbf{p})$ = approx. penetration depth of mass point **p**, $\mathbf{x}_i$ = point of the intersection of an edge incident to **p** with surface Q, $\mathbf{n}_i$ = normal to surface of Q at point $\mathbf{x}_i$,

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

- Direction of the penalty force on border points:

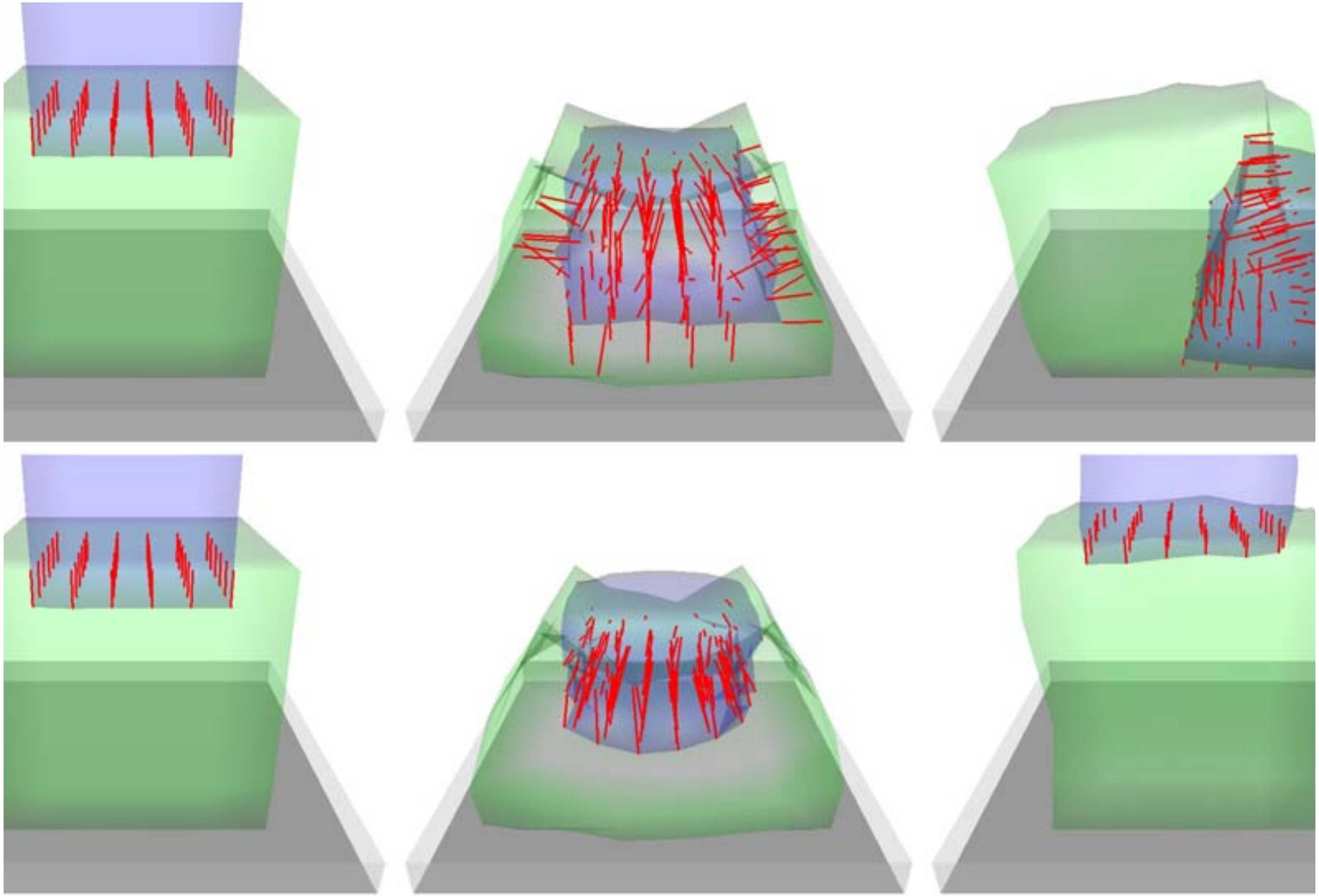
$$\mathbf{r}(\mathbf{p}) = \frac{\sum_{i=1}^{k} \omega(\mathbf{x}_i, \mathbf{p})\, \mathbf{n}_i}{\sum_{i=1}^{k} \omega(\mathbf{x}_i, \mathbf{p})}$$

4. Phase: propagate forces by way of breadth-first traversal through the tetrahedron mesh

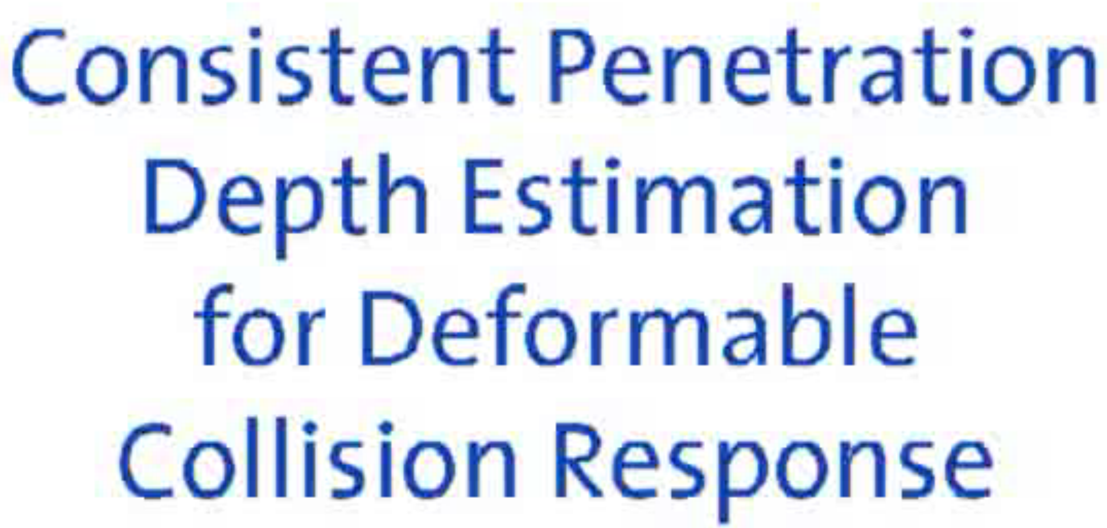
$$d(\mathbf{p}) = \frac{\sum_{i=1}^{k} \omega(\mathbf{p}_i, \mathbf{p})\big((\mathbf{p}_i - \mathbf{p})\cdot\mathbf{r}_i + d(\mathbf{p}_i)\big)}{\sum_{i=1}^{k} \omega(\mathbf{x}_i, \mathbf{p})}$$

where $\mathbf{p}_i$ = points of P that have been visited already, $\mathbf{p}$ = point not yet visited, $\mathbf{r}_i$ = direction of the estimated penalty force in point $\mathbf{p}_i$.

Visualization



Video



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