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:

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

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

- 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** (w.r.t.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:

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

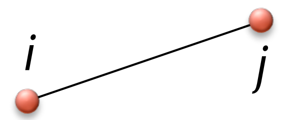
$$\text{mit} \quad A \in \mathbb{R}^{3n \times 3n}, \quad b \in \mathbb{R}^{3n}$$

- 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 :

$$\begin{array}{r c}
3i \rightarrow & \multirow{2}{*}{} \\
\end{array}
\begin{array}{r}
3i \rightarrow \\[1em]
3j \rightarrow
\end{array}
\begin{pmatrix}
\boxed{K_{ii}} & \boxed{K_{ij}} \\[1em]
\boxed{K_{ji}} & \boxed{K_{jj}}
\end{pmatrix}
\quad
\begin{array}{cc}
\uparrow & \uparrow \\
3i & 3j
\end{array}$$



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

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

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

- First of all, compute K_{ii}:

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

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

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

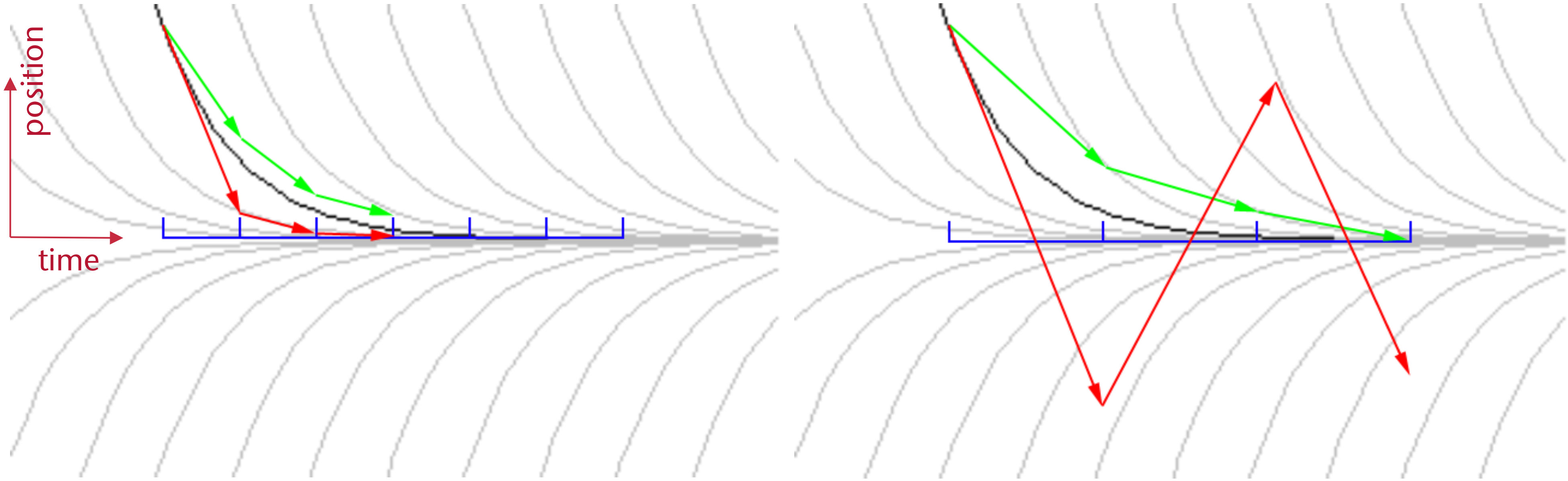
Overall Solution Algorithm

- 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

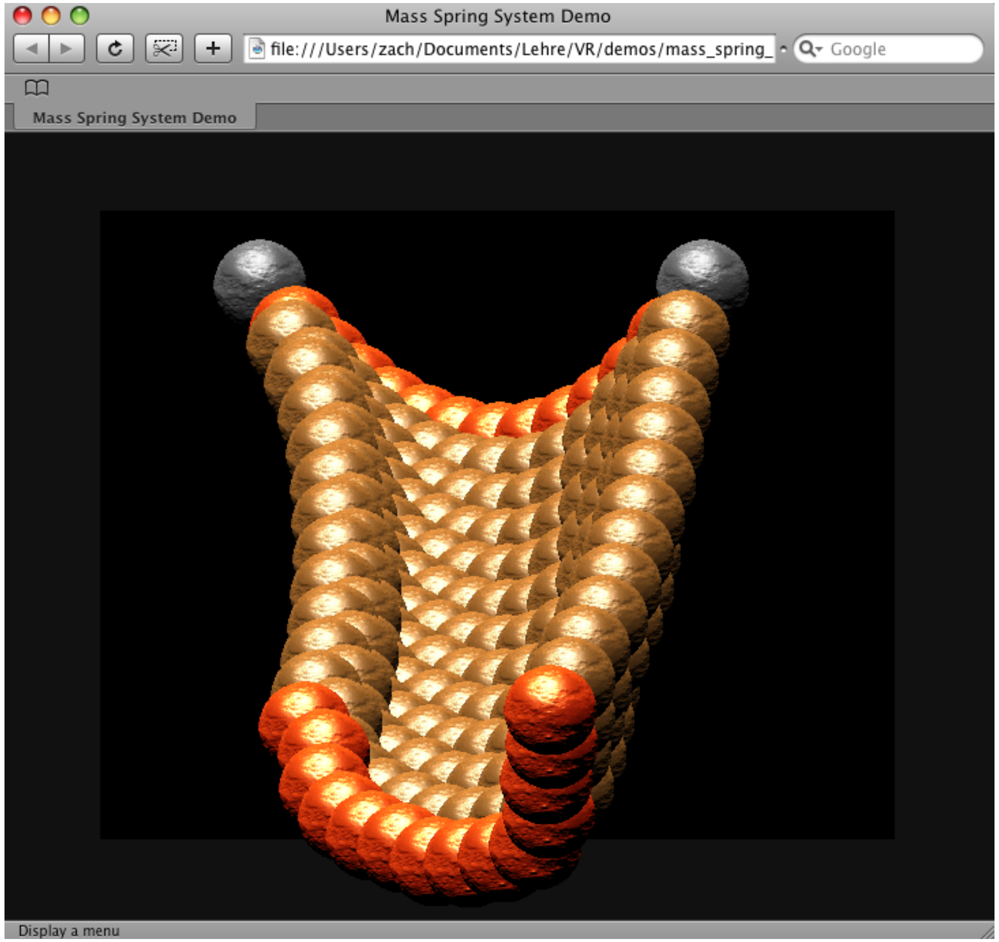
- Explicit integration:
  - \+ Very easy to implement
  - \- Small step sizes needed
  - \- Stiff springs don't work very well
  - \- Forces are propagated only by one spring per time step
- Implicit Integration:
  - \+ Unconditionally stable
  - \+ Stiff springs work better
  - \+ Globale solver → forces 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 behindly, 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

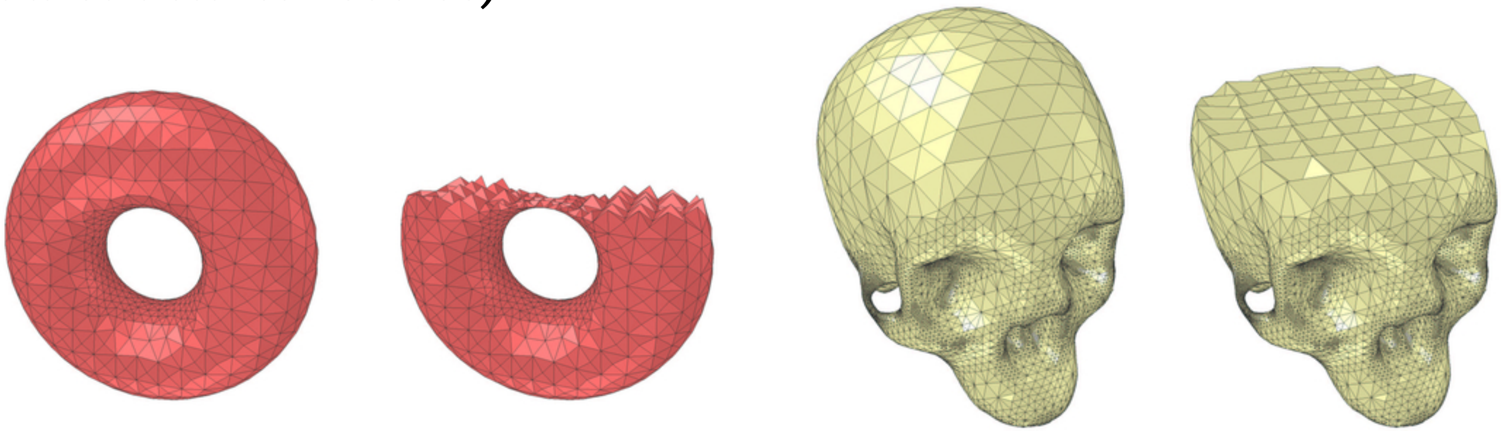


http://www.dhteumeuleu.com/dhtml/v-grid.html

Mesh Creation for Volumetric Objects

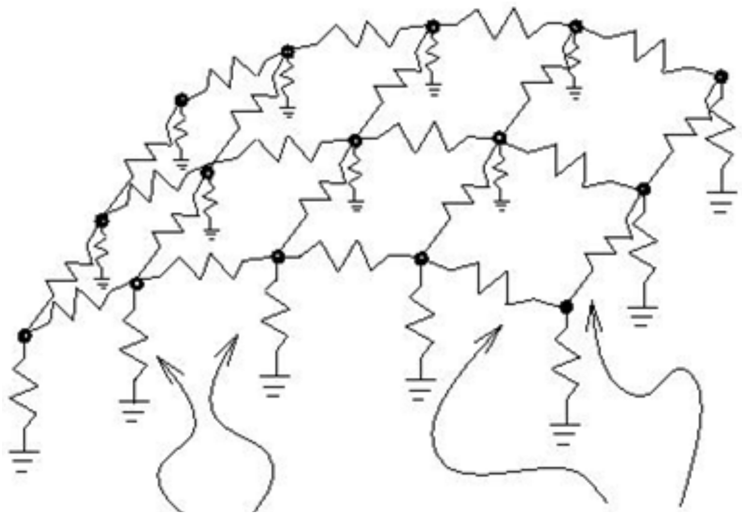
- How to create a mass-spring system for a volumetric model?
- Direct conversion of 3D (surface) geometry into spring-mass system does not yield good results:
  - Geometry has too high a complexity
  - Degenerate polygons
- Better (and still simple) idea:
  - Create a tetrahedron mesh out of the geometry (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

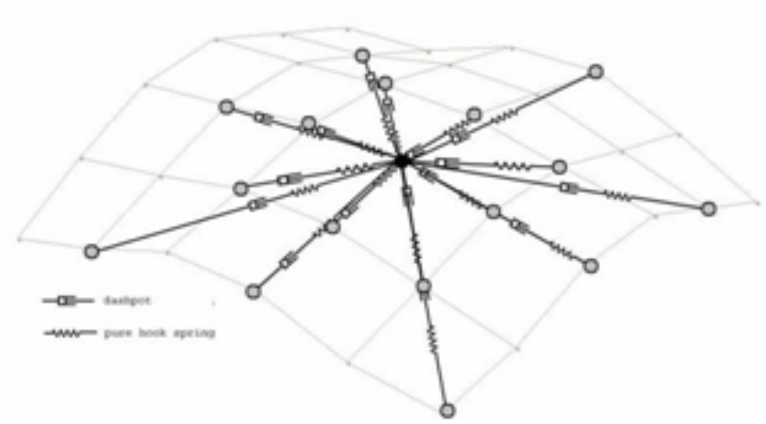
- Generation of the tetrahedron mesh (simple method):
  - Distribute a number of points uniformly (perhaps randomly) in the interior of the geometry (so called "Steiner points")
  - Dito for a sheet/band above the surface
  - Connect the points by Delaunay triangulation (see my "Geometric Data structures"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



- Anchor the surface meshes within the tetraeder mesh:
  - Represent each vertex of the surface mesh by baryzentric combination of ist surrounding tetraeder mesh

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



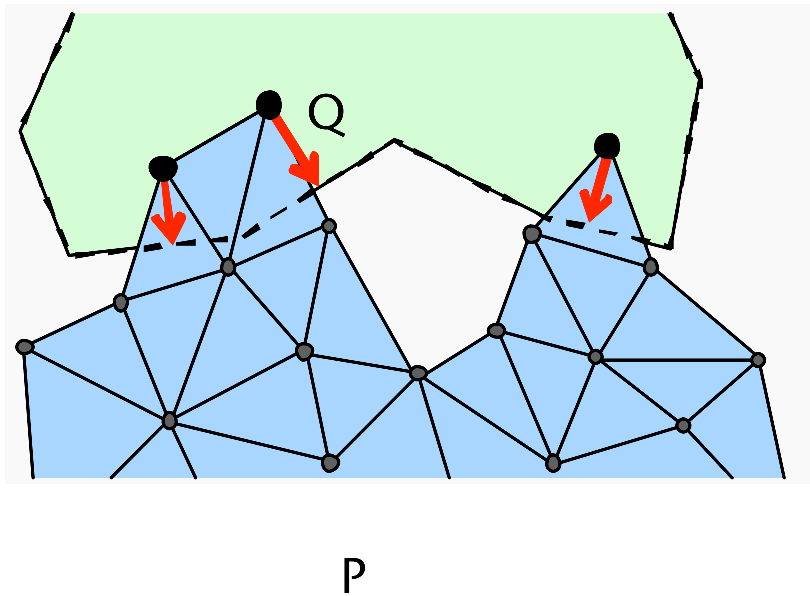


Collision Detection

- 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

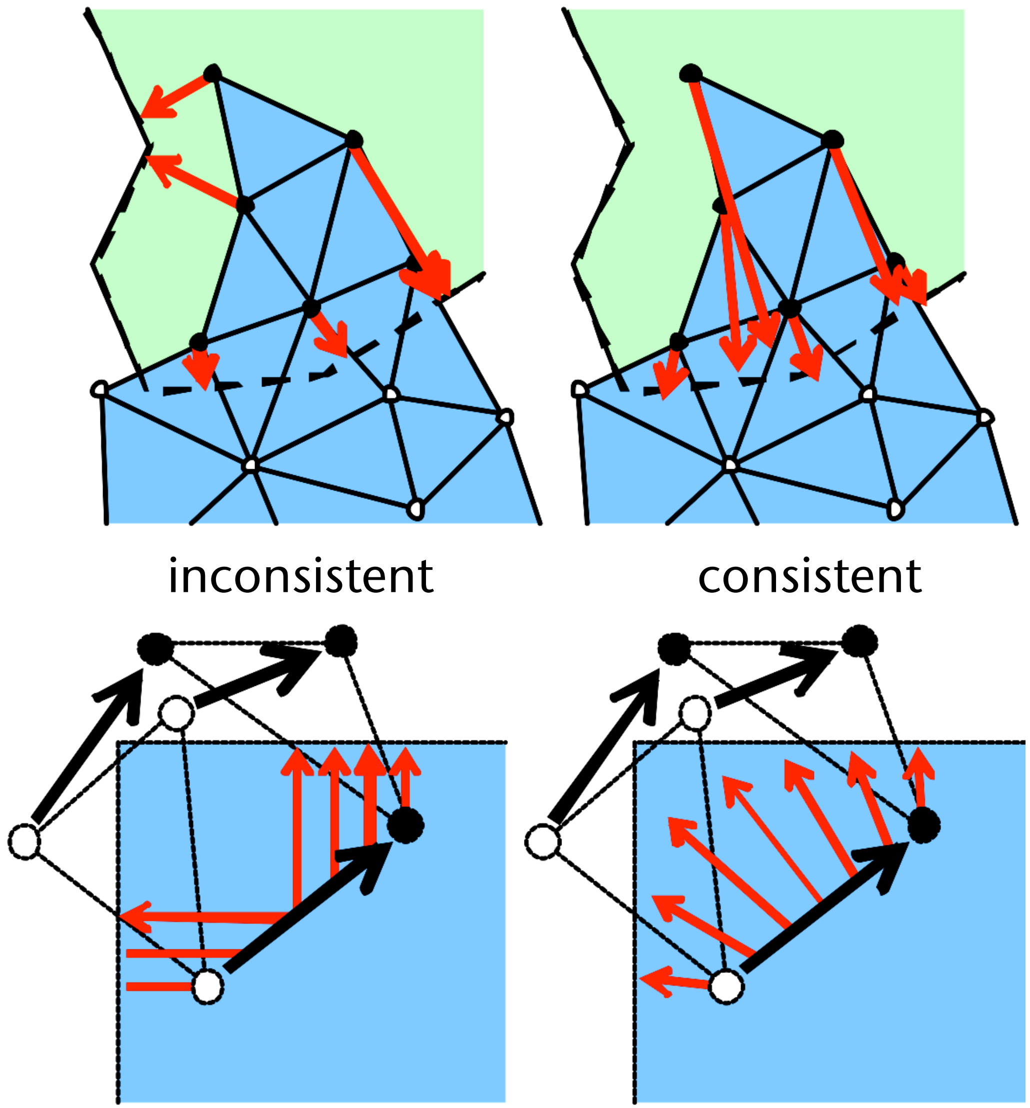
Collision Response

- 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)



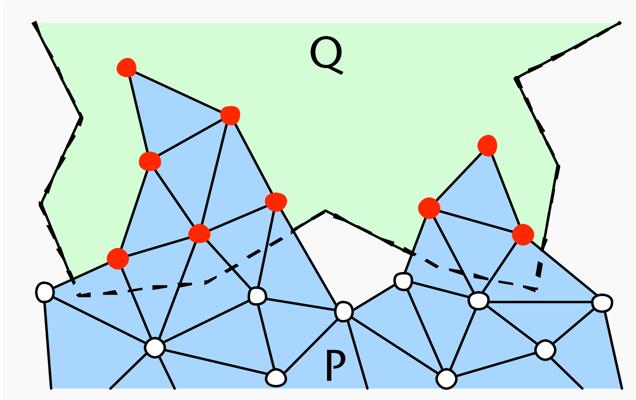
- Problem:
  - Implausible forces
  - "Tunneling" (s. a. the chapter on force-feedback)

- Examples:

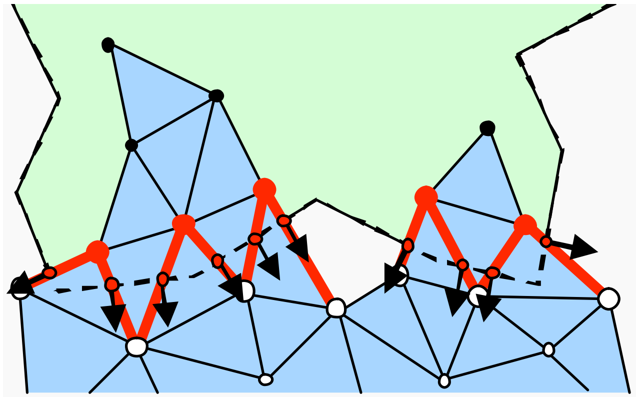


Consistente Penalty Forces

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



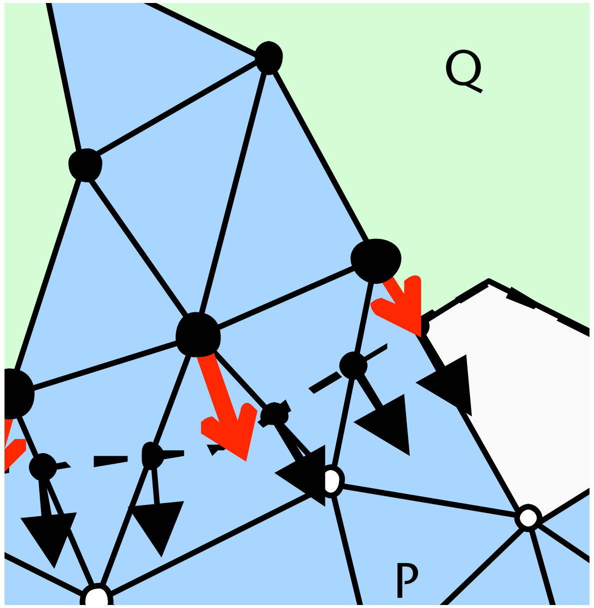
3. Phase: compute the approximate force for border points

- 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:

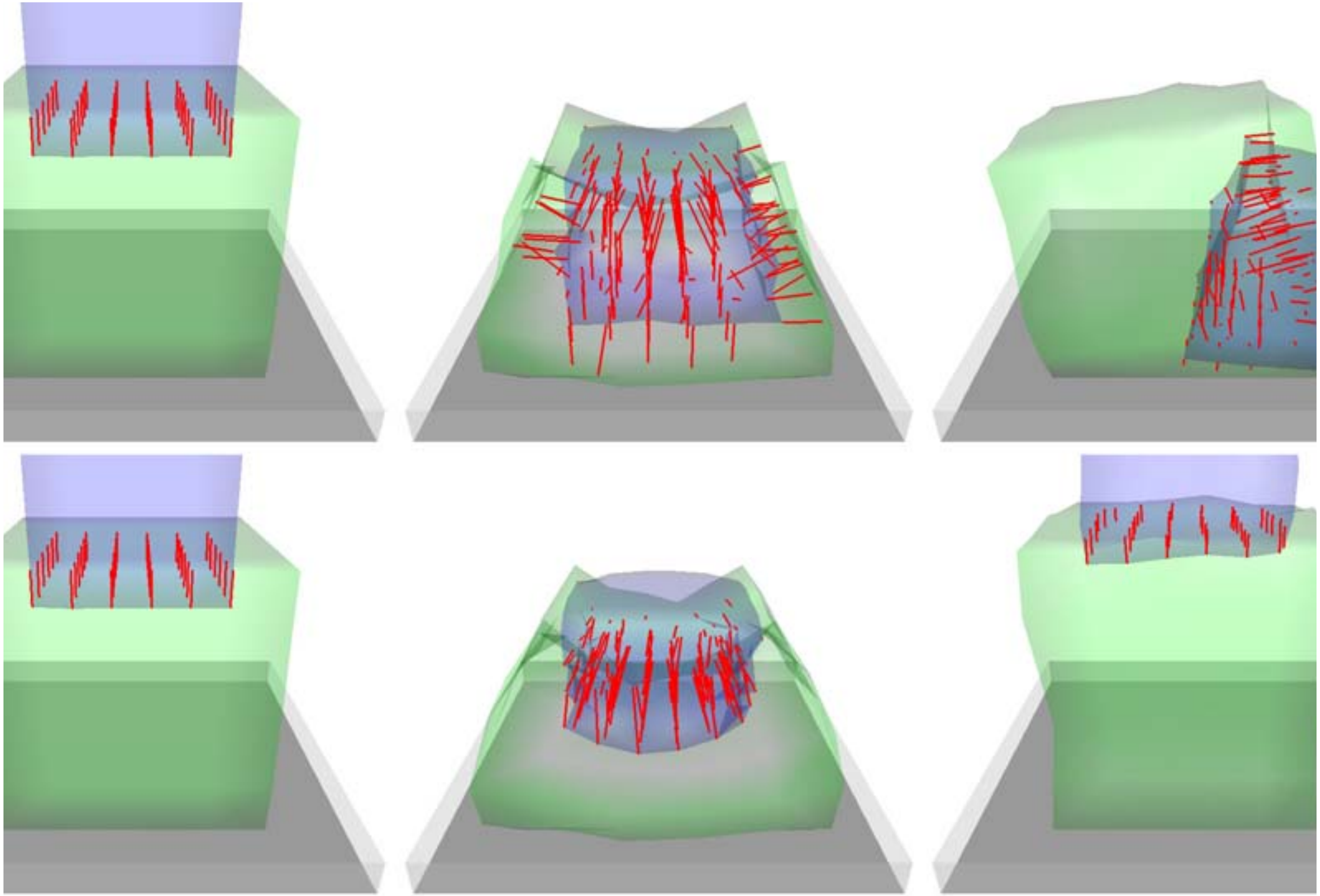
$$\mathbf{r}(\mathbf{p}) = \frac{\sum_{i=1}^{k} \omega(\mathbf{x}_i, \mathbf{p})\, \mathbf{n}_i}{\sum_{i=1}^{k} \omega(\mathbf{x}_i, \mathbf{p})}$$

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

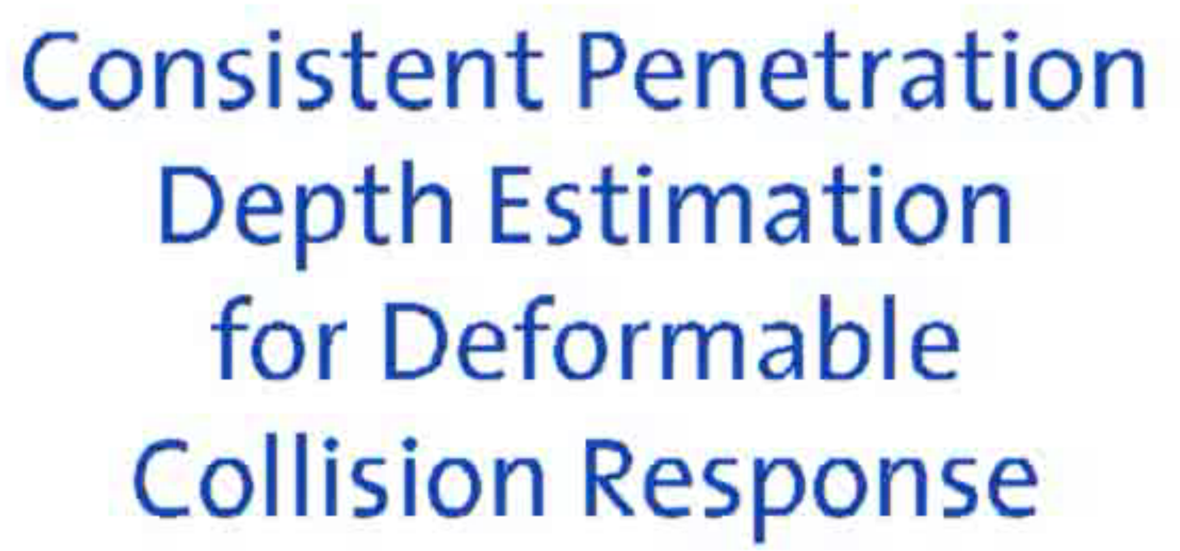
$$d(\mathbf{p}) = \frac{\sum_{i=1}^{k} \omega(\mathbf{p}_i, \mathbf{p})\big((\mathbf{p}_i - \mathbf{p})\cdot\mathbf{r}_i + d(\mathbf{p}_i)\big)}{\sum_{i=1}^{k} \omega(\mathbf{x}_i, \mathbf{p})}$$

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