Physically-Based Modeling: Mass-Spring Systems

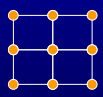
M. Alex O. Vasilescu

Mass-Spring Systems

Mass-Spring Systems

Snake implementation:

Image Processing /
Surface Reconstruction:



Mass-Spring Systems

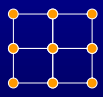
Snake implementation:

Image processing/
Surface Reconstruction:

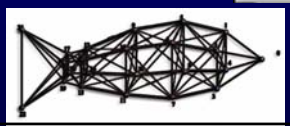
Mass-Spring Systems

Snake implementation:

Surface Reconstruction:



Bodies for creepy crawlies:



Surface Reconstruction

Data Structures

- Node i, $i=1,..,N$
  - Mass: m_i
  - Position: $\mathbf{x}_i(t)=[x_i(t),y_i(t),z_i(t)]^t$
  - Velocity: $\mathbf{v}_i(t)=d\mathbf{x}_i(t)/dt$
  - Accelaration $\mathbf{a}_i(t)=d^2\mathbf{x}_i(t)/dt^2$
  - Net nodal force: $\mathbf{f}_i^{net}(t)$

```
typedef struct node{
  float mass;
  vector position;
  vector velocity;
  vector force;
} node;
```

- Spring
  - Connects node j to node i
  - Natural length l_{ij}
  - Stiffness k_{ij}

```
typedef struct spring{
  node *n1;
  node *n2;
  double rest_length;
  double spring_constant;
} spring;
```

System Dynamics:

- Lagrange equations of motion:

$$m_i\ddot{\mathbf{x}}_i+\gamma_i\dot{\mathbf{x}}_i-\mathbf{g}_i-\mathbf{f}_i=0,$$

- γ_i is damping coefficient
- $\mathbf{f}_i$ is the external force at node i
- $\mathbf{g}_i$ total internal force on the node i due to neighboring nodes

System Dynamics:

- Lagrange equations of motion:

$$\mathbf{F}_{i,\text{total}}=-\gamma_i\dot{\mathbf{x}}_i+\mathbf{g}_i+\mathbf{f}_i$$

$$\mathbf{F}_{i,\text{total}}=m_i\ddot{\mathbf{x}}_i$$

- γ_i is damping coefficient
- $\mathbf{f}_i$ is the external force at node i
- $\mathbf{g}_i$ total internal force on the node i due to neighboring nodes

$$m_i\ddot{\mathbf{x}}_i+\gamma_i\dot{\mathbf{x}}_i-\mathbf{g}_i-\mathbf{f}_i=0,$$

Equations of Motion

Lagrange equations of motion:

$$m_i\ddot{\mathbf{x}}_i+\gamma_i\dot{\mathbf{x}}_i-\mathbf{g}_i-\mathbf{f}_i=0,$$

- γ_i is damping coeff
- $\mathbf{f}_i$ is the external force at node i
- $\mathbf{g}_i$ total force on the node i due to springs connecting node i to neighboring nodes

Explicit Euler time integration:

$$\mathbf{f}_i^{net}(t)=\mathbf{f}_i(t)-\gamma\dot{\mathbf{x}}_i(t)+\mathbf{g}_i(t)$$

$$\mathbf{a}_i(t)=\frac{\mathbf{f}_i^{net}}{m_i}$$

$$\mathbf{v}_i(t+\Delta t)=\mathbf{v}_i(t)+\Delta t\,\mathbf{a}_i(t)$$

$$\mathbf{x}_i(t+\Delta t)=\mathbf{x}_i(t)+\Delta t\,\mathbf{v}_i(t+\Delta t)$$

```
void compute_forces(int num_nodes, node *nds,
                    int num_springs, spring *sprs)
{ zeroize_forces(num_nodes, nds);
  spring_forces(num_springs, sprs);
  damping_forces(num_nodes, nds);
  external_forces(num_nodes, nds); }

void time_step(float dt, int num_nodes, node *nds,
               int num_springs, spring *sprs)
{vector accel, delta_vel, delta_pos;int i;
  compute_forces(num_nodes, nds, num_springs, sprs)
  for (i=0; i<num_nodes; i++){
    vscale((1 / nds[i].mass), nds[i].force, accel);
    vscale(dt, accel, delta_vel);
    vplus(delta_vel, nds[i].velocity, nds[i].velocity);
    vscale(dt, nds[i].velocity, delta_pos);
    vinc(delta_pos, nds[i].position);}}
```

Force Computation (Cont.)

- Damping Forces: $\gamma\dot{\mathbf{x}}_i(t)$

```
void damping_forces (int num_nodes, node *nds)
{ vector force; int i;
  for (i=0; i<num_nodes; i++){
    vscale(DAMPING, nds[i].velocity, force);
    vdec(force, nds[i].force);}}
```

- External force (very simple) :
  - springs attached to the data from each mesh node

$$f_i(t)=k_{data}\,\|\,\mathbf{data}(\mathbf{x}_i(t))-\mathbf{x}_i(t)\,\|$$

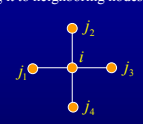
  - project the position of node i $\mathbf{x}_i(t)$ into the data, and extract the value at that location $\mathbf{data}(\mathbf{x}_i(t))$
  - calculate the force using a data spring constant k_{data}

```
void external_forces(int num_nodes, node *nds)
{ int i; node *nd; vector p; float val;
  for (i=0; i<num_nodes; i++) {
    nd = &nds[i];
    p = nd->position;
    val=image_interp(p.x, p.y, IMAGE_DATA);
    nd->force.z += IMAGE_CONST*(val - p.z);
  }}
```

Mass-Spring Forces

- Spring Forces:
  - $\mathbf{g}_i(t)$ total force on the node i due to springs connecting it to neighboring nodes $j\in N_i$

$$\mathbf{g}_i(t)=\sum_{j\in N_i}\mathbf{s}_{ij}$$



  - the force spring ij exerts on node i

$$\mathbf{s}_{ij}=k_{ij}\,e_{ij}\frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|}$$

$\mathbf{r}_{ij}=\mathbf{x}_j-\mathbf{x}_i$ node separation

$\|\mathbf{r}_{ij}\|$ actual spring length

$e_{ij}=\|\mathbf{r}_{ij}\|-l_{ij}$ spring deformation, l_{ij} natural spring length

k_{ij} is the spring constant for the spring connecting node i and node j

Force Computation

- Spring Forces:
  - $\mathbf{g}_i$ total force on the node i due to springs connecting it to neighboring nodes $j \in N_i$

$$\mathbf{g}_i(t) = \sum_{j \in N_i} \mathbf{s}_{ij}$$

  - the force spring ij exerts on node i

$$\mathbf{s}_{ij} = k_{ij} e_{ij} \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|}$$

  - where

$\mathbf{r}_{ij} = \mathbf{x}_j - \mathbf{x}_i$ is separation of nodes

$\|\mathbf{r}_{ij}\|$ is actual length of spring

$e_{ij} = \|\mathbf{r}_{ij}\| - l_{ij}$ is deformation of spring

```
void spring_forces (int num_springs, spring *sprs)
{ int i;
  for (i=0; i<num_springs; i++) spring_force(&sprs[i]);}

void spring_force (spring *s)
{ node *node1, *node2;
  double length, extension, scale_factor;
  vector direction, force;

  node1 = s->n1; node2 = s->n2;
  vminus(node2->position,node1->position, direction);

  length = vlength(direction) ;

  deformation = length - s->rest_length;
  scale_factor = ((deformation * s->spring_constant)/length);
  vscale(scale_factor, direction, force);
  vinc(force, node1->force);
  vdec(force, node2->force);}
```

Adaptive Image Reconstruction

Adaptive Meshes

- Use Lagrangian equations of motion to simulate mesh dynamics.

$$m_i \ddot{\mathbf{x}}_i + \gamma_i \dot{\mathbf{x}}_i - \mathbf{g}_i - \mathbf{f}_i = 0,$$

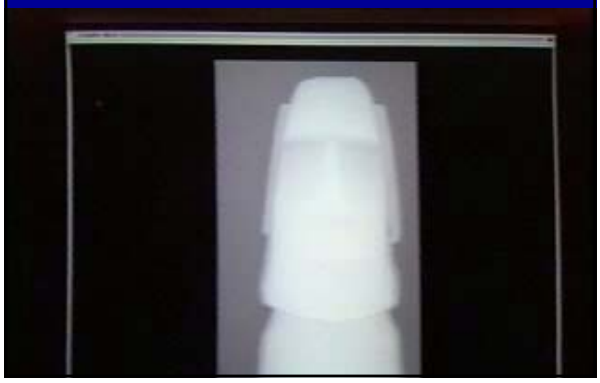
- The internal spring forces have spring "constants" that vary according to a data dependent function, such as the magnitude of the gradient.

$$\mathbf{g}_i(t) = \sum_{j \in N_i} \mathbf{s}_{ij}, \qquad \mathbf{s}_{ij} = k_{ij} e_{ij} \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|}$$

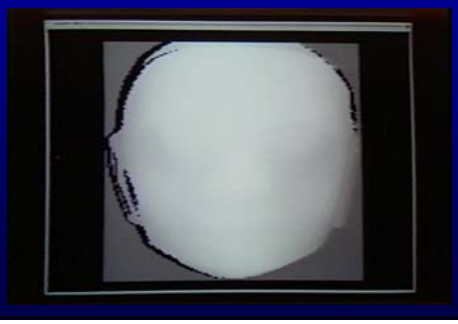
- Digital Image: $\mathbf{d}(k,l)$
- Adaptation function: $\mathbf{a}_d = G * \|\nabla \mathbf{d}\|$ $\qquad G = \text{gaussian}$
- Observation: $O_i = \mathbf{a}_d(x_i, y_i)$
- Spring "Constant": - varies according to the adaptation function

$$k_{ij} = (1-\rho)k_{\min} + \rho\, k_{\max}, \qquad \rho = .5(O_i + O_j)$$

Adaptive Surface Reconstruction



Adaptive Surface Reconstruction



SpringLens

Distributed Nonlinear Magnifications

Tobias Germer
Timo Götzelmann
Martin Spindler
Thomas Strothotte

Department of Simulation and Graphics
Otto-von-Guericke University of Magdeburg

Mass-Spring Systems Applications

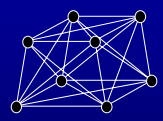
Surface Reconstruction:

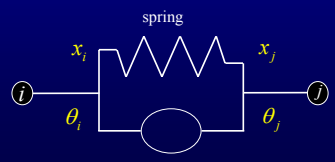
Deformable objects capable of:

- Heat conduction,
- Thermoelasticity,
- Melting and fluid-like behavior in the molten state

Cloth simulation

Heating and Melting Deformable Models

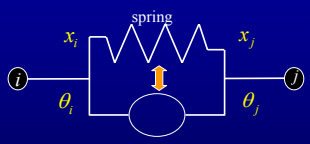




thermal conductive element

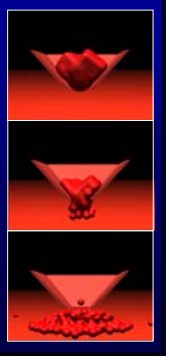
Thermoelasticity

- Spring stiffness, k_{ij} is a function of θ



thermal conductive element

$$k_{ij} = \begin{cases} k_{ij}^0 & \text{if } \theta^a \le \theta^s \\ k_{ij}^0 - \nu\left(\theta^a - \theta^s\right) & \text{if } \theta^s < \theta^a < \theta^m \\ & \nu = k_{ij} / \left(\theta^m - \theta^s\right) \\ 0 & \text{if } \theta^a \ge \theta^m \end{cases}$$



Heat/Diffusion Equation

- Diffusion of heat in materials:

$$\frac{\partial}{\partial t}(\mu\sigma\theta) - \nabla\cdot(\mathrm{C}\nabla\theta) = q$$

Heat is conducted from high temperature to low temperature .

The rate of heat conduction per unit area is proportional to the gradient of the temperature.

The amount of heat required to raise the temperature of a material theta degrees is proportional to the mass of the sample per unit volume and the proportionality factor sigma, the specific heat of the material.

Heat/Diffusion Equation

- Diffusion of heat in materials:

$$\frac{\partial}{\partial t}(\mu\sigma\theta) - \nabla\cdot(\mathbf{C}\nabla\theta) = q$$

q rate of heat generation/loss per unit volume

μ mass density, kg/m³

σ specific heat, joule/(kg·Kelvin), material property

θ temperature, Kelvin

$\mathbf{C}$ thermal conductivity matrix, material property

$\nabla = \left[\frac{\partial}{\partial u}, \frac{\partial}{\partial v}, \frac{\partial}{\partial w}\right]$ gradient operator

Heat/Diffusion Equation

- Homogeneous, isotropic material:

$$\frac{\partial}{\partial t}(\mu\sigma\theta) - c\nabla^2\theta = q$$

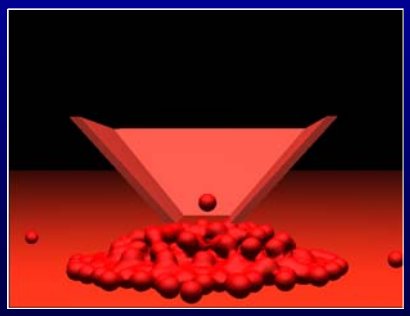
- Discretize the heat equation:

$$\mu\sigma\frac{(\theta^{t+\Delta t}-\theta^t)}{\Delta t} - c\left[\frac{\theta^t_{u+\Delta u,v,w}-2\theta^t_{u,v,w}+\theta^t_{u-\Delta u,v,w}}{\Delta u^2} + \frac{\theta^t_{u,v+\Delta v,w}-2\theta^t_{u,v,w}+\theta^t_{u,v-\Delta v,w}}{\Delta v^2} + \frac{\theta^t_{u,v,w+\Delta w}-2\theta^t_{u,v,w}+\theta^t_{u,v,w-\Delta w}}{\Delta w^2}\right] = q$$

Update θ

$$\theta^{t+\Delta t}_{u,v,w} = \theta^t_{u,v,w} + \frac{\Delta t}{\mu\sigma}c\left[\frac{\theta^t_{u+\Delta u,v,w}-2\theta^t_{u,v,w}+\theta^t_{u-\Delta u,v,w}}{\Delta u^2} + \frac{\theta^t_{u,v+\Delta v,w}-2\theta^t_{u,v,w}+\theta^t_{u,v-\Delta v,w}}{\Delta v^2} + \frac{\theta^t_{u,v,w+\Delta w}-2\theta^t_{u,v,w}+\theta^t_{u,v,w-\Delta w}}{\Delta w^2}\right] + q$$

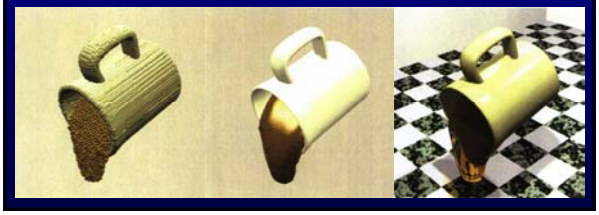
Thermoelasticity



Liquids – Particle Models

Model long range attraction and short range repulsion forces between pairs of particles according to Lennard-Jones potentials.

Forces involving inverse powers of particle separation



Discrete Fluid Model

The total force on a particle, *i*, due to all other particles

$$\mathbf{g}_i(t) = \sum_{j\neq i}\mathbf{g}_{ij}(t)$$

$$\mathbf{g}_{ij}(t) = m_i m_j(\mathbf{x}_i - \mathbf{x}_j)\left(\overbrace{-\frac{\alpha}{(r_{ij}+\varsigma)^a}}^{\text{attraction term}} + \overbrace{\frac{\beta}{(r_{ij})^b}}^{\text{repulsion term}}\right)$$

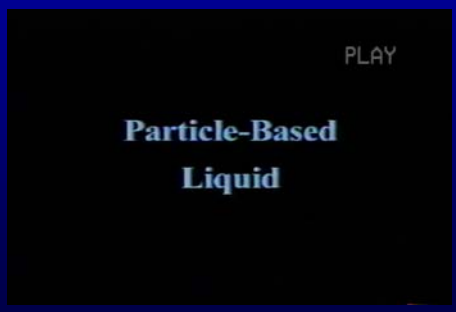
$a=2$ and $b=4$

α and β determine the strength of the attraction and repulsion forces

$r_{ij} = \|\mathbf{x}_j - \mathbf{x}_i\|$

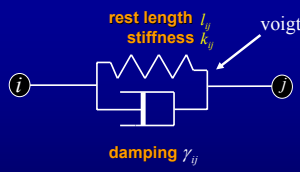
ζ minimum required separation between particles

Particle-Based Liquid



Pixar's "Geri's Game"

Viscoelastic Element



mass m_i
position $\mathbf{x}_i(t)$
velocity $\mathbf{v}_i(t) = \dot{\mathbf{x}}_i(t)$
acceleration $\mathbf{a}_i(t) = \dot{\mathbf{v}}_i(t) = \ddot{\mathbf{x}}_i(t)$

- Equations of motion

$$m_i \ddot{\mathbf{x}}_i + \gamma_i \dot{\mathbf{x}}_i + (k_{ij} e_{ij} + \gamma_{ij} \dot{e}_{ij}) \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|} = \mathbf{f}_i$$

$$m_j \ddot{\mathbf{x}}_j + \gamma_j \dot{\mathbf{x}}_j - (k_{ij} e_{ij} + \gamma_{ij} \dot{e}_{ij}) \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|} = \mathbf{f}_j$$

$$\mathbf{r}_{ij} = \mathbf{x}_j - \mathbf{x}_i$$
$$r_{ij} = \|\mathbf{r}_{ij}\|$$
$$e_{ij} = r_{ij} - l_{ij}$$

Viscoelastic Element

$$\boxed{\begin{aligned} m_i \ddot{\mathbf{x}}_i + \gamma_i \dot{\mathbf{x}}_i + c_{ij} \mathbf{r}_{ij} &= \mathbf{f}_i \\ m_j \ddot{\mathbf{x}}_j + \gamma_j \dot{\mathbf{x}}_j - c_{ij} \mathbf{r}_{ij} &= \mathbf{f}_j \end{aligned}}$$

, effective stiffness:

$$c_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \frac{k_{ij} e_{ij} + \gamma_{ij} \dot{e}_{ij}}{\|\mathbf{r}_{ij}\|}$$

$$\underbrace{\begin{bmatrix} m_i & 0 \\ 0 & m_j \end{bmatrix}}_{\mathbf{M}} \underbrace{\begin{bmatrix} \ddot{\mathbf{x}}_i \\ \ddot{\mathbf{x}}_j \end{bmatrix}}_{\ddot{\mathbf{x}}} + \underbrace{\begin{bmatrix} \gamma_i & 0 \\ 0 & \gamma_j \end{bmatrix}}_{\mathbf{G}} \underbrace{\begin{bmatrix} \dot{\mathbf{x}}_i \\ \dot{\mathbf{x}}_j \end{bmatrix}}_{\dot{\mathbf{x}}} + \underbrace{\begin{bmatrix} -c_{ij} & c_{ij} \\ c_{ij} & -c_{ij} \end{bmatrix}}_{\mathbf{K}(\mathbf{x})} \underbrace{\begin{bmatrix} \mathbf{x}_i \\ \mathbf{x}_j \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} \mathbf{f}_i \\ \mathbf{f}_j \end{bmatrix}}_{\mathbf{f}}$$

Implicit Numerical Solution

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{G}\dot{\mathbf{x}} + \mathbf{K}\mathbf{x} = \mathbf{f}$$

discretize time: $t \Rightarrow 0, \Delta t, 2\Delta t, \cdots, t, t+\Delta t, \cdots$

$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^t + \Delta t\, \dot{\mathbf{x}}^{t+\Delta t}$$
$$\dot{\mathbf{x}}^{t+\Delta t} = \dot{\mathbf{x}}^t + \Delta t\, \ddot{\mathbf{x}}^{t+\Delta t}$$
$$\Rightarrow \ddot{\mathbf{x}}^{t+\Delta t} = (\dot{\mathbf{x}}^{t+\Delta t} - \dot{\mathbf{x}}^t)/\Delta t$$

$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^t + \Delta t\, \dot{\mathbf{x}}^t$$
$$\dot{\mathbf{x}}^{t+\Delta t} = \dot{\mathbf{x}}^t + \Delta t\, \ddot{\mathbf{x}}^t$$
$$\ddot{\mathbf{x}}^{t+\Delta t}$$

$$\left(\frac{\mathbf{M}}{\Delta t}\dot{\mathbf{x}}^{t+\Delta t} - \frac{\mathbf{M}}{\Delta t}\dot{\mathbf{x}}^t\right) + \left(\mathbf{G}^t \dot{\mathbf{x}}^{t+\Delta t}\right) + \left(\mathbf{K}^t \mathbf{x}^t + \Delta t \mathbf{K}^t \dot{\mathbf{x}}^{t+\Delta t}\right) = \mathbf{f}^t$$

Implicit Numerical Solution

$$\left(\frac{\mathbf{M}}{\Delta t}\dot{\mathbf{x}}^{t+\Delta t} - \frac{\mathbf{M}}{\Delta t}\dot{\mathbf{x}}^t\right) + \left(\mathbf{G}^t \dot{\mathbf{x}}^{t+\Delta t}\right) + \left(\mathbf{K}^t \mathbf{x}^t + \Delta t \mathbf{K}^t \dot{\mathbf{x}}^{t+\Delta t}\right) = \mathbf{f}^t$$

$$\underbrace{\left(\frac{\mathbf{M}}{\Delta t} + \mathbf{G}^t + \Delta t \mathbf{K}^t\right)}_{\mathbf{A}^t} \dot{\mathbf{x}}^{t+\Delta t} = \underbrace{\mathbf{f}^t + \frac{\mathbf{M}}{\Delta t}\dot{\mathbf{x}}^t - \mathbf{K}^t \mathbf{x}^t}_{\mathbf{b}^t}$$

A, effective system matrix

b, effective load matrix

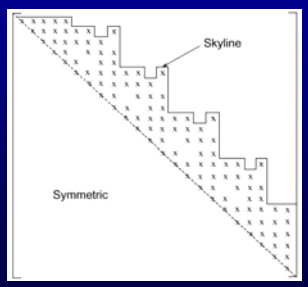
Semi-Implicit Euler

$$\begin{bmatrix} \frac{m_i}{\Delta t}+\gamma_i-c_{ij}\Delta t & c_{ij}\Delta t \\ c_{ij}\Delta t & \frac{m_j}{\Delta t}+\gamma_j-c_{ij}\Delta t \end{bmatrix} \begin{bmatrix} \dot{\mathbf{x}}_i^{t+\Delta t} \\ \dot{\mathbf{x}}_j^{t+\Delta t} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_i+\frac{m_i}{\Delta t}\dot{\mathbf{x}}_i^t-c_{ij}(\mathbf{x}_i^t-\mathbf{x}_j^t) \\ \mathbf{f}_j+\frac{m_j}{\Delta t}\dot{\mathbf{x}}_j^t+c_{ij}(\mathbf{x}_i^t-\mathbf{x}_j^t) \end{bmatrix}$$

$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^t + \Delta t\, \dot{\mathbf{x}}^{t+\Delta t}$$

Skyline Storage Schemes for System Matrix A

A is a $n^2 \times n^2$ matrix that typically requires $O(n^3)$ storage.



Solving for Nodal Velocities

- Factorize: $A^T = LDL^T$ where D = diagonal matrix, L = lower triangular

$$\underbrace{\mathbf{L}\,\mathbf{D}\,\mathbf{L}^T\ \dot{\mathbf{x}}^{t+\Delta t}}_{} = \mathbf{b}^t$$

$$\mathbf{L} \quad \mathbf{Q} \quad = \mathbf{b}^t$$

- Solve lower triangular system for Q (forward-substitution)
- Solve upper triangular system (back-substitution)

$$\mathbf{D}\mathbf{L}^T\dot{\mathbf{x}}^{t+\Delta t} = \mathbf{Q}$$

$$\mathbf{L}^T\dot{\mathbf{x}}^{t+\Delta t} = \mathbf{D}^{-1}\mathbf{Q} \quad \Rightarrow \quad \mathbf{x}^{t+\Delta t} = \mathbf{x}^t + \Delta t\, \dot{\mathbf{x}}^{t+\Delta t}$$