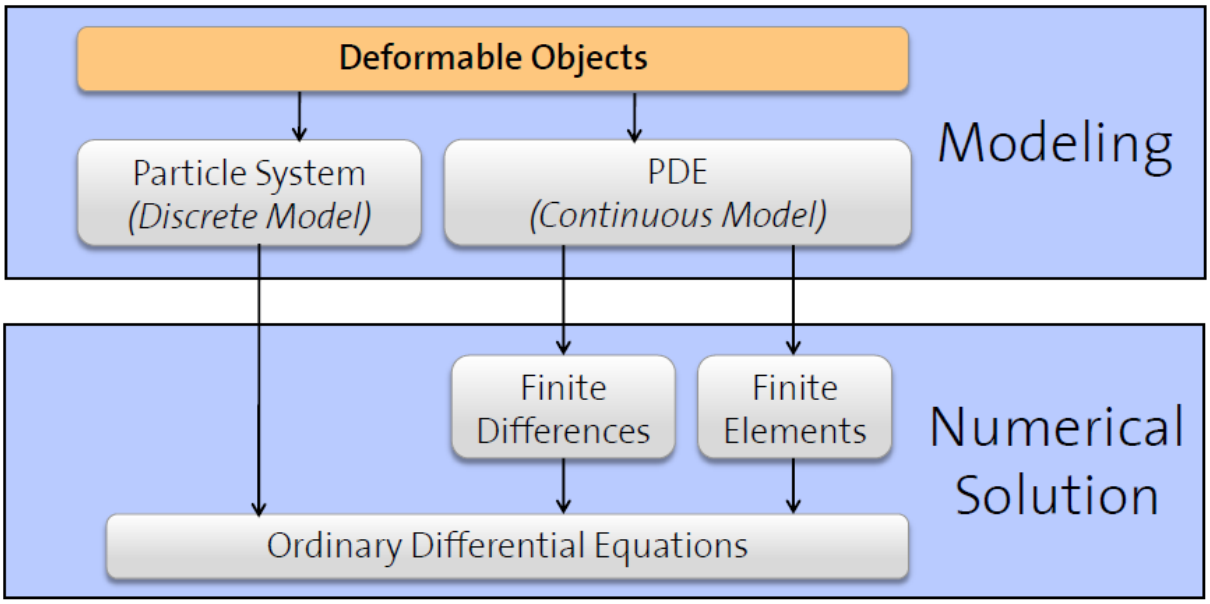
Summary Physically-Based simulation in computer graphics

Physical simulation roadmap



Particle System

Mass-Spring systems

Mass points

- Sample objects (uniformly) with mass points
- Each **mass point** has properties
  - Mass m_i
  - Position $\boldsymbol{x_i}(t)$
  - Velocity $\boldsymbol{v_i}(t)$

Forces

- **Forces** = external forces + internal forces
- Internal forces: **Elastic springs**
  - 1D: $F = -k(l - L)$
  - 3D: $\boldsymbol{F_i} = -k\left(\left|\left|\boldsymbol{x_i} - \boldsymbol{x_j}\right|\right| - L\right)\frac{\boldsymbol{x_i} - \boldsymbol{x_j}}{\left|\left|\boldsymbol{x_i} - \boldsymbol{x_j}\right|\right|}$

    where k is the spring constant, l is the elongated length and L is the original length. $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ are the end positions of the spring, i.e. $\left|\left|\boldsymbol{x_i} - \boldsymbol{x_j}\right|\right| = l$
  - Total spring force at mass point 0 with A_0 being the set of adjacent mass points.

$$F_0 = -\sum_{i \in A_0} k_i\left(\left|\left|\boldsymbol{x_i} - \boldsymbol{x_0}\right|\right| - L_i\right)\frac{\boldsymbol{x_i} - \boldsymbol{x_0}}{\left|\left|\boldsymbol{x_i} - \boldsymbol{x_0}\right|\right|}$$

- Internal forces: **Dissipation / Point damping**
  - $\boldsymbol{F}^{pd}(t) = -\gamma \cdot \boldsymbol{v}(t)$

Equations of motion

- For each mass point: $m_i \frac{d^2 \boldsymbol{x_i}(t)}{dt^2} + \gamma \cdot \frac{d\boldsymbol{x_i}(t)}{dt} = \boldsymbol{F}_i(t)$
- Coupled first order problem: $\frac{d\boldsymbol{y}(t)}{dt} = \boldsymbol{y}(t)$, where

$$\boldsymbol{y}(t) = \begin{pmatrix} \boldsymbol{x_i}(t) \\ \boldsymbol{v_i}(t) \end{pmatrix}, \frac{dy(t)}{dt} = f(t, \boldsymbol{y}(t)) = \begin{pmatrix} \boldsymbol{v_i}(t) \\ \frac{\boldsymbol{F_i}(t) - \gamma \cdot \boldsymbol{v_i}(t)}{m_i} \end{pmatrix}$$

First-order numerical integration of ODE's

Local error (single step): $\left| (y_n + \int_{t_n}^{t_{n+1}} y'(t)dt) \; - y^{n+1} \right|$

Global error (accumulated): $|y_i - y(t_i)|$

Accuracy: Method is accurate of order p, if the local error is $O(h^{p+1})$. The accuracy can be determined by comparing the update formula with the Taylor expansion.

General Runge-Kutta methods

- Given the **IVP**: $y'(t) = f\big(t, y(t)\big), y(0) = y_0$
- **s-step RK solution**: $y_{n+1} = y_n + h \sum_{j=1}^{s} b_j k_j$
- Intermediate steps: $k_j = f(t_n + hc_j, y_n + h \sum_{l=1}^{s} a_{jl} k_l)$, $j = 1, \dots, s$
- The defining coefficients a,b and c are conveniently depicted in the Butcher scheme

$$\begin{pmatrix} \boldsymbol{c} & A \\ & \boldsymbol{b^T} \end{pmatrix}$$

Euler method

- Butcher scheme: $\begin{pmatrix} 0 & \\ & 1 \end{pmatrix}$
- Update formula: $\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + hf(t_n, \boldsymbol{y}_n)$
- EOM: $\begin{pmatrix} \boldsymbol{x_i}(t) \\ \boldsymbol{v_i}(t) \end{pmatrix}^{n+1} = \begin{pmatrix} \boldsymbol{x_i}(t) \\ \boldsymbol{v_i}(t) \end{pmatrix}^{n} + h \begin{pmatrix} \boldsymbol{v_i}(t) \\ \frac{\boldsymbol{F_i}(t) - \gamma \cdot \boldsymbol{v_i}(t)}{m_i} \end{pmatrix}$
- Accuracy: Order 1, $O(h^2)$ error per step

Heun's method

- Butcher scheme: $\begin{pmatrix} 0 & & \\ 1 & 1 & \\ & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$
- Idea: $y(t+h) = y(t) + \frac{h}{2}\big(y'(t) + y'(t+h)\big) + O(h^3)$. For $y'(t+h)$, a first order approximation is enough, since there is h in front of the parenthesis that makes it a second order approximation in the end.
- Update formula:
  - $k_0 = f(t_n, y_n)$
  - $\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{h}{2} k_0 + \frac{h}{2} f(t_n + h, y_n + hk_0)$
- Accuracy: Order 2, $O(h^3)$ error per step

Midpoint rule

- Butcher scheme: $\begin{pmatrix} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ & 0 & 1 \end{pmatrix}$
- Accuracy: Order 2

Backwards Euler

- Butcher scheme: $\begin{pmatrix} 1 & 1 \\ & 1 \end{pmatrix}$
- Update formula: $y_{k+1} = y_k + hf(t_{k+1}, y_{k+1})$
- Generally involves solving a set of nonlinear equations
- Accuracy: Order 1

Semi-implicit Euler

- Same as backwards Euler, but approximate $f(t_{k+1}, y_{k+1})$ using Taylor approximation.
- Details, see slides "Semi-implicit Euler"
- Emerging LSE has a sparse matrix A
    - Solvable with Krylov-subspace methods ((P)CG, Jacobi, Gauss-Seidel, Cholesky decomposition)

Higher-Order numerical integration

Verlet

- $x(t+h) = 2x(t) - x(t-h) + h^2 a(t) + O(h^4)$
- +: Accuracy: Order 2
- +: Only one force evaluation
- -: A posteriori approximation of velocities
- -: Two-step method problematic at discontinuities

Leapfrog

- $v\left(t + \frac{h}{2}\right) = v\left(t - \frac{h}{2}\right) + h \cdot a(t)$
  $x(t+h) = x(t) + h \cdot v\left(t + \frac{h}{2}\right)$
- +: Accuracy: 2nd order
- +: Only one force evaluation
- -: a(t) must not depend on v(t) => no damping

Symplectic Euler

- $v(t+h) = v(t) + h \cdot a(t)$
  $x(t+h) = x(t) + h \cdot v(t+h)$
- -: Accuracy: 1st order
- +: Good stability for oscillatory motion
- +: Good conservation of momentum and energy

Stability

- Apply integration scheme to test equation, e.g. $y' = \lambda y$
- Von Neumann stability check, i.e. assume solution to be of form $y(t) = e^{i\lambda t}$
- CFL-criterion: $\frac{a\Delta t}{\Delta x} < m$, where m depends on the difference stencil. For most schemes, it is 1.

Practical issues of mass-spring systems

Types of springs

- Structural springs: Stretching
- Diagonal springs: Shearing
- Interleaved springs: Bending

Problems

- Material behaviour strongly dependent on spring network
- Spring couple different deformation modes: Bending and shear springs both respond to stretch.
- How to distinguish between stretches and shear in triangular meshes?

Method of constraints

- Strictly enforce conditions (not just via force imposing)
- Restrict motion (with only mass-spring motion is unlimited)
- A constraint $C(x_1, \dots, x_n)$: $\mathbb{R}^n \to \mathbb{R}$ is satisfied if and only if $C = 0$

Fluid simulation

Eulerian (grid-based) viewpoint

Navier Stokes Equation

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = -\frac{1}{\rho}\nabla p + \nu \, \Delta \, u + g$$

With: $\nabla \cdot u = 0$ (Continuity equation)

Discretisation

Separate time-dependent NSE part.

$$\begin{aligned} \frac{\partial u}{\partial t} &= RHS \\ \frac{u^{n+1} - u^n}{\Delta t} &\approx RHS \\ u^{n+1} &\approx u^n + \Delta t \cdot RHS \end{aligned}$$

Discretise RHS parts and insert them later in the RHS term.

Advection

$(u \cdot \nabla)u$: Use Semi-Lagrangian advection

Diffusion

$$\nu \, \Delta \, u \approx \frac{\nu}{\Delta x \Delta y}(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}$$

Gravity

$$g = g$$

Embarrassingly easy.

Pressure equation and divergence-freedom

Pressure is defined through the continuity equation. Therefore,

$$\begin{aligned} \nabla \cdot \left(u' - \Delta t \frac{1}{\rho}\nabla p\right) &= 0 \\ \Leftrightarrow \nabla \cdot \nabla p &= \frac{\rho}{\Delta \mathrm{t}} \nabla \cdot u' \end{aligned}$$

Advection, diffusion and gravity (and all other forces) have already been applied to u'.

The discretisation for the two-dimensional gradient is straight-forward.

$$\nabla f|_{i,j} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} \approx \frac{f_{i+1,j} - f_{i-1,j}}{2\Delta x} + \frac{f_{i,j+1} - f_{i,j-1}}{2\Delta y}$$

Caution: If solved on a staggered grid, the operator changes slightly.

$$\nabla f|_{i+\frac{1}{2},j+\frac{1}{2}} \approx \frac{f_{i+1,j} - f_{i,j}}{\Delta x} + \frac{f_{i,j+1} - f_{i,j}}{\Delta y}$$

Lagrangian (particle-based) viewpoint

Navier-Stokes equation

$$\rho \frac{\partial u}{\partial t} + \cancel{\rho(u \cdot \nabla)u} = -\nabla p + \mu \Delta u + \rho g$$

- Force density formulation. Note: $\rho\nu = \mu$
- No advective term (since particle is directly followed)
- Mass conservation guaranteed (no particle is lost), but neither are incompressibility nor divergence-freedom.

Discretisation

- Particles represent a certain fluid volume, they are not molecules!
- Properties vary continuously between particles => Kernel functions

SPH summation equation

$$A(\boldsymbol{x}) = \sum_{j}^{N(\boldsymbol{x})} \frac{m_j}{\rho_j} A_j W(\boldsymbol{x} - \boldsymbol{x_j}, h)$$

- A: Quantity A (e.g. density, pressure, etc.)
- $N(\boldsymbol{x})$: Neighborhood of x. Concretely, $N(x) = \{j | \|\boldsymbol{x} - \boldsymbol{x_j}\| < h\}$
- $\frac{m_j}{\rho_j}$: Particle volume
- W: Smoothing kernel

Kernel properties

- Normalisation condition: $\int W(\boldsymbol{x} - \boldsymbol{x}', h) d\boldsymbol{x}' = 1$
- Dirac delta limit: $\lim_{h \to 0} W(\boldsymbol{x} - \boldsymbol{x}', h) = \delta(\boldsymbol{x} - \boldsymbol{x}')$

  Actually, if the kernel was a Dirac delta function, the particles would represent individual molecules.
- Compact condition: $W(\boldsymbol{x} - \boldsymbol{x}', h) = 0$ for $\|\boldsymbol{x} - \boldsymbol{x}'\| > h$
- Example: $W(x_{ij}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - x_{ij}^2)^3, & 0 \le x_{ij} < h \\ 0, & \text{otherwise} \end{cases}$
- Notation: $W_{ij} \coloneqq W(x_{ij}, h)$

Density

$$\rho_i = \sum_j m_j W_{ij}$$

Pressure

Equation of state: $p = \rho RT$

Introduce a rest density of fluid (water: 1000) and roughly approximate RT-terms with a parameter k:

$$p_i = k(\rho_i - \rho_0)$$

Additionally, for stability:

$$p_i = \max\left(0, k(\rho_i - \rho_0)\right)$$

Incompressibility

- Incompressibility can be enforced by a very large k.
- Leads to very stiff systems => Small timesteps

Pressure force density

$$-\nabla p = -\sum_j \frac{m_j}{\rho_j} \frac{p_i + p_j}{2} \nabla W_{ij}$$

Note: $\frac{p_i+p_j}{2}$ instead of only p_i is used to guarantee a symmetric force. (Actio = reactio)

Viscosity force density

$$\mu \nabla^2 \boldsymbol{u} = \mu \sum_j \frac{m_j}{\rho_j} \text{abs}(\boldsymbol{u_j} - \boldsymbol{u_i}) \nabla^2 W_{ij}$$

Note: abs stands for element-wise absolute value. $u_j - u_i$ just denotes the difference vector and must be equal from both sides, i.e. symmetric. Also here, the difference vector guarantees symmetry of the force.

Algorithm overview

1. Compute neighbourhoods
2. Compute densities and velocities
3. Compute forces
4. Compute new velocity, new position
5. Do collision handling
6. Start again