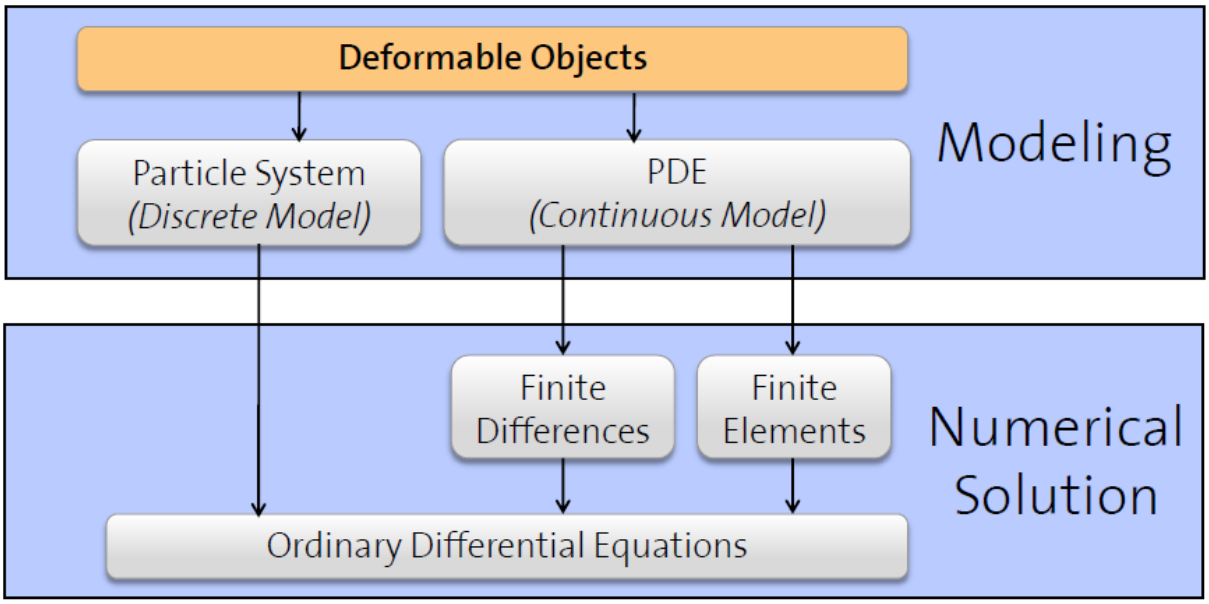
Summary Physically-Based simulation in computer graphics

1 Physical simulation roadmap



2 Particle System

2.1 Mass-Spring systems

2.1.1 Mass points

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

2.1.2 Forces

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

    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.

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

- Internal forces: **Dissipation / Point damping**

- $\boldsymbol{F}^{pd}(t) = -\gamma \cdot \boldsymbol{v}(t)$
- Damps all motion (translations and rotations)

2.2 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

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

2.3 First-order numerical integration of ODE's

2.3.1 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

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

2.3.2 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

2.3.3 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

2.3.4 Midpoint rule

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

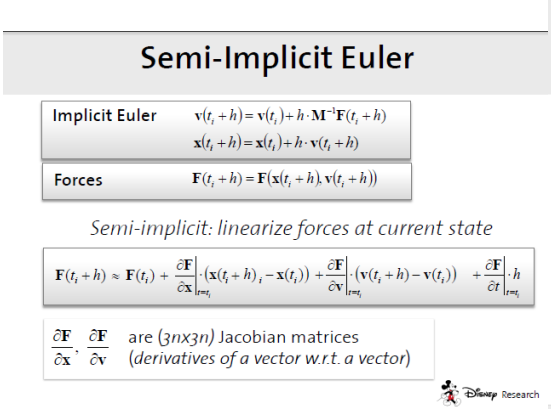
2.3.5 Backwards Euler

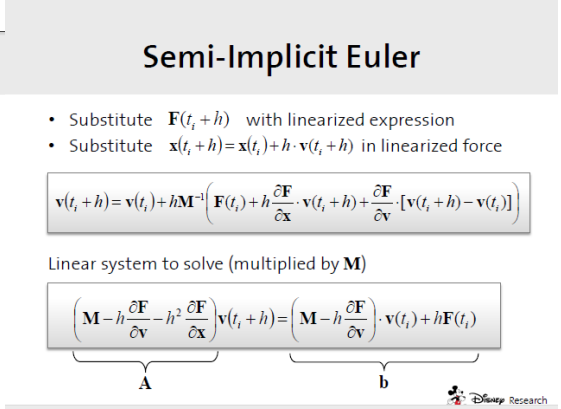
- Butcher scheme: $\begin{pmatrix} 1 & 1 \\ & 1 \end{pmatrix}$
- Update formula: $y_{k+1} = y_k + hf(t_{k+1}, y_{k+1})$

- Generally involves solving a set of nonlinear equations, i.e. Newton-iteration.
- Accuracy: Order 1

2.3.6 Semi-implicit Euler

- Same as backwards Euler, but approximate $f(t_{k+1}, y_{k+1})$ using Taylor approximation (i.e. avoid nonlinearity)





- Emerging LSE has a sparse symmetric matrix A
  - 3x3 blocks relate 3d forces to 3d positions
  - A_{ij} is only nonzero if $i = j$ or if there is a spring between nodes I and j
  - Solvable with Krylov-subspace methods ((P)CG, Jacobi, Gauss-Seidel, Cholesky decomposition)

2.4 Higher-Order numerical integration

2.4.1 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

2.4.2 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

2.4.3 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

2.5 Analysis of integrators

Analyse integrators with respect to accuracy, convergence, stability and efficiency.

2.5.1 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.

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

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

2.5.2 Convergence

2.5.3 Stability

- Apply integration scheme to test equation, e.g. $y' = \lambda y$
  Example explicit Euler: $y_{n+1} = y_n + h\lambda y_n = (1 + h\lambda)y_n$
  Step size restriction: $y_{n+1} < \infty \Leftrightarrow |1 + h\lambda| < 1 \Rightarrow h < \frac{2}{\lambda}$
- 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.

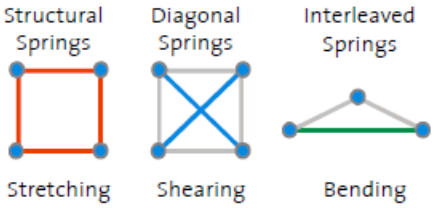
2.5.3.1 Stiff problems

Explicit methods are efficient, but are in general stiff, i.e.

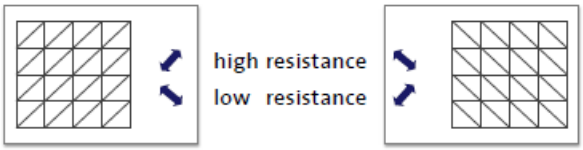
- Require very small time steps for stable integration
- Inefficient since step size is determined by stability, not accuracy requirements

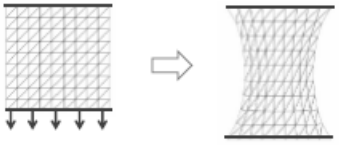
2.6 Practical issues of mass-spring systems

2.6.1 Types of springs

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

2.6.2 Problems

- Material behaviour strongly dependent on spring network (i.e. spring layout and spring constants)
  

- Spring couple different deformation modes: Bending and shear springs both respond to stretch.
  

- How to distinguish between stretches and shear in **triangular** meshes?
- No explicit volume preservation for solids

2.7 Method of constraints

- Alternative to Mass spring system
- Strictly enforces conditions (not just via force imposing)
- Restricts motion (with only mass-spring motion is unlimited), e.g. for bead on a wire, incompressible deformations

2.7.1 Penalty forces / Soft constraints

The simulation with the method of constraints replaces the internal forces from the mass-spring system with penalty forces.

Penalty forces tolerate violations, no strict enforcement of constraints.

- A constraint $C(x_1, \dots, x_n)$: $\mathbb{R}^n \rightarrow \mathbb{R}$ is satisfied if and only if $C = 0$.
  - For example, $C(\boldsymbol{x_1}) = \boldsymbol{x_1} - \boldsymbol{p}$ would denote a constant position p.
  - C: $\mathbb{R}^2 \rightarrow \mathbb{R}$ denotes a constant length
  - C: $\mathbb{R}^3 \rightarrow \mathbb{R}$ denotes a constant area
  - C: $\mathbb{R}^4 \rightarrow \mathbb{R}$ denotes a constant volume
- Constraint energy: $E_C = \frac{1}{2} k C^2$
- Constraint force aka penalty force: $F_j = -\frac{\partial E_C}{\partial \boldsymbol{x_j}} = -kC \frac{\partial C}{\partial \boldsymbol{x_j}}$
- Explicit Euler forward step: $v_{n+1} = v_n + hM^{-1}(F^C(x) + F^{ext})$

2.7.2 Hard constraints

See slides 55 – 58 of MassSpring2

- Constraint stays satisfied, not only a correction term, but governing equation. Theoretical background comes from Lagrangian multipliers.
- Constraint force magnitude: $\lambda_n = -f(F^{ext}, C)$. Involves solution of a LES in general.
- Constraint force: $F_n^C = \frac{\partial C(x_n)}{\partial x} \lambda_n$
- Explicit Euler forward step: $v_{n+1} = v_n + hM^{-1}(F^C(x) + F^{ext})$

2.7.2.1 Advantages / Disadvantages

- +: Solves correction exactly, as it adds just enough force to maintain constraint
- +: Requires no high stiffness (numerically advantageous)
- -: Complicated formulation, i.e. complicated derivation of equations.
- -: Requires solution of LSE.

3 PDE

Look at FEM Guideline to find out how the method works.

3.1 PDE Classification

Given a 2nd order linear PDE of the form

$$Au_{xx} + 2Bu_{xy} + Cu_{yy} = F(x, y, u, u_x, u_y)$$

The PDE is

Hyperbolic	If $B^2 - AC > 0$	Wave equation
Parabolic	If $B^2 - AC = 0$	Heat equation

Elliptic	If $B^2 - AC < 0$	Laplace equation, Poisson equation

3.2 Poisson equation with finite differences

$$\nabla^2 u = f$$

$$\nabla^2 u[i,j] = \frac{u[i+1,j] + u[i-1,j] + u[i,j-1] + u[i,j+1] - 4u[i,j]}{h^2} = f[i,j]$$

3.2.1 Problems of FD

- +: Easy to understand and implement
- -: Requires regular grid
- -: High smoothness requirements on solution (C^2)
- -: Higher order derivatives require large stencils

3.3 Poisson equation with FEM

3.3.1 Local stiffness matrices

$$\int_K grad\ b_N^i\ grad\ b_N^j\ dx \approx |K| \cdot \frac{1}{2|K|} n^i \cdot \frac{1}{2|K|} n^j$$

n^i is the normal to the i-th edge. The i-th edge is opposite node i. K denotes the triangle. The local stiffness matrix (and the Galerkin matrix) is symmetric positive definite and sparse for hat basis functions.

3.3.2 Element load vector

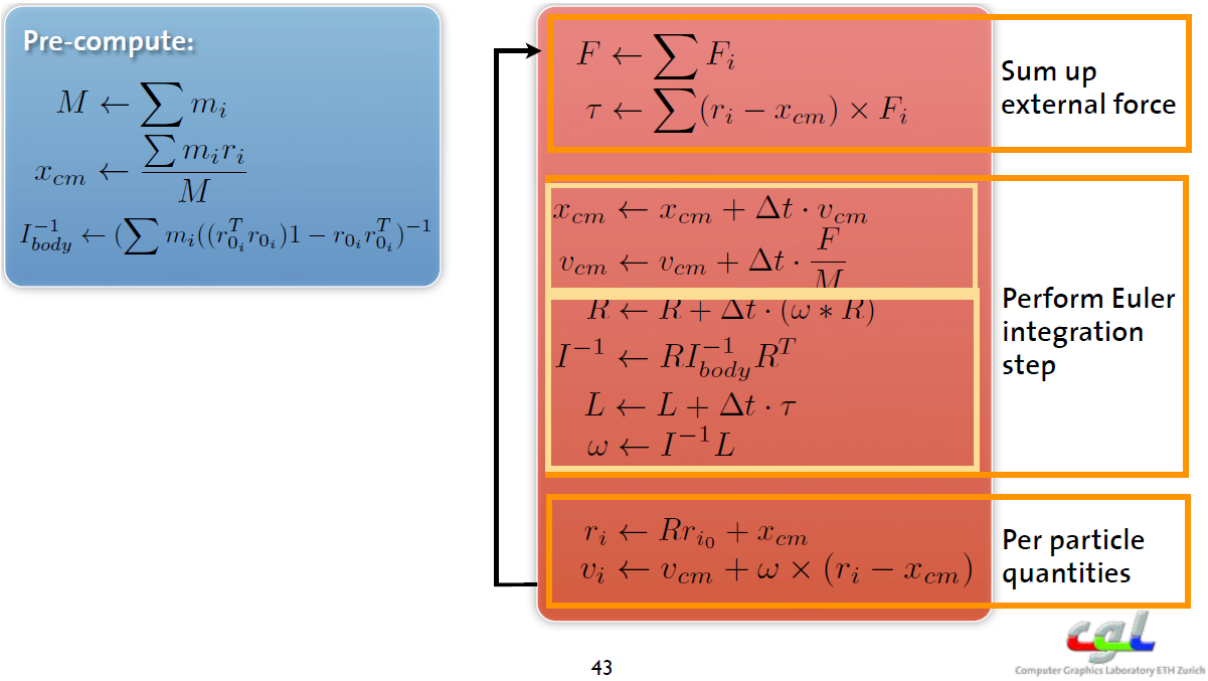
$$\int_K f(x) b_{N|K}^j(x) dx \approx \frac{|K|}{3} \begin{pmatrix} f(a_1) \\ f(a_2) \\ f(a_3) \end{pmatrix}$$

For the RHS, the composite trapezoidal rule is used.

3.3.3 Advantages of FEM

- +: No problem for complex geometry
- +: Solution defined on whole domain with interpolating basis functions
- +: Weaker smoothness constraints on solution (C^1)

4 Rigid body simulation

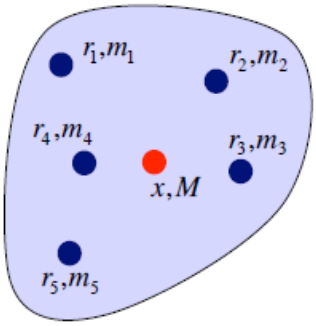


Input: Rigid body (positions, masses); External force

Output: Transformed position of rigid body according to external force

4.1 Rigid body

A rigid body is defined by multiple particle positions and masses. The distances between each particle must not change. The center of mass is constant and computed as



$$x_{cm} = \frac{\sum_i x_i m_i}{\sum_i m_i}$$

4.1.1 Object space / World space

In object space, the center of mass is the origin. r_{0i} denotes the position of the i-th coordinate in object space.

In world space,

$$r_i = R r_{0i} + x_{cm}$$

Where R is a 3x3 rotation matrix, defining the rotation of the object.

4.2 Equations of motion

An application of an external force not only induces a displacement of the rigid body, but also a spin of it. Therefore, Newton's equations of motion and the conservation of angular momentum must be considered.

4.2.1 Newton's equations of motion

Newton's equations of motion apply to the center of mass, i.e.

$$\frac{\partial}{\partial t} x_{cm} = v_{cm}$$

$$\frac{\partial}{\partial t} v_{cm} = \frac{\sum_i F_i}{\sum_i m_i}$$

4.2.2 Conservation of angular momentum

Angular momentum is defined as

$$L(t) = I(t)\omega(t)$$

Where $I(t)$ is the inertia tensor (Trägheitsmoment) (3x3 matrix) and $\omega(t)$ is the angular velocity vector.

If the motion is defined around the center of mass,

$$L(t) = \sum_i (\boldsymbol{r}_{0i} \times m_i \boldsymbol{v}_i)$$

The time derivative then reads

$$\frac{\partial}{\partial t} L(t) = \sum_i \frac{d\boldsymbol{r_{0i}}}{dt} \times m_i \boldsymbol{v}_i + \boldsymbol{r}_{0i} \times \frac{d(m_i \boldsymbol{v_i})}{dt} = \sum_i 0 + r_{0i} \times F_i = \sum_i (r_i(t) - x(t)) \times F_i(t)$$

This **links external force to angular momentum**. Note that the cross-product of velocity and momentum is zero, because these vectors are parallel. Again, r_{0i} is in object space. If no external force is applied, $\frac{\partial}{\partial t} L(t) = 0$. This is the conservation of angular momentum.

The torque (Drehmoment) is defined as

$$\tau = \sum_i \tau_i = \sum_i (r_i(t) - x(t)) \times F_i(t)$$

4.2.3 Linking angular velocity to particle position

The application of an external force has an effect on the angular momentum and induces an angular velocity, since

$$\omega(t) = I^{-1}(t) L(t)$$

To relate angular velocity and particle positions, we have to relate $\omega(t)$ to $R(t)$ in a similar fashion as $\dot{x}(t) = v(t)$. It turns out that we can write

$$\dot{R}(t) = \left(\omega(t) \times \begin{pmatrix} R_{11} \\ R_{12} \\ R_{13} \end{pmatrix}, \omega(t) \times \begin{pmatrix} R_{21} \\ R_{22} \\ R_{23} \end{pmatrix}, \omega(t) \times \begin{pmatrix} R_{31} \\ R_{32} \\ R_{33} \end{pmatrix} \right) \equiv \omega(t) * R(t)$$

This **links angular velocity to particle positions**. $*$ denotes a column-wise cross-product operator.

4.2.4 Linking angular velocity to particle velocity

Particle velocities are defined as $\frac{\partial}{\partial t} r_i(t)$. Starting from the definition of r_i, we can plug in the definition of $\dot{R}(t)$

$$\begin{aligned} r_i(t) &= R(t) r_{0i} + x_{cm} \\ \frac{\partial}{\partial t} r_i(t) &= \dot{R}(t) r_{0i} + v_{cm} \\ \frac{\partial}{\partial t} r_i(t) &= \omega(t) * \big(R(t) r_{0i} + x(t) - x(t)\big) + v_{cm} \end{aligned}$$

$$\frac{\partial}{\partial t} r_i(t) = \omega(t) * \left(\underbrace{R(t) r_{0i} + x(t)}_{= r_{i(t)}} - x(t) \right) + v_{cm}$$

$$\frac{\partial}{\partial t} r_i(t) = \omega(t) * \big(r_i(t) - x(t) \big) + v_{cm}$$

4.3 Collision handling

There are two types of collision contacts:

- Resting contacts. Objects remain together (unelastischer Stoss)
- Colliding contacts. Objects move apart (elastischer Stoss)

To compute the change in velocity for a colliding contact, there are two possibilities:

- Force driven (F=ma): Penetration causes forces. Changes velocity only indirectly.
- Impulse driven (J=mv): Velocities are directly changed. Changes linear and angular velocity.

For a resting contact, there is no impulse (since it rests relative to other objects around). Therefore, a resting force must be computed. All resting forces have to be computed simultaneously which leads to a quadratic programming problem.

5 Continuum mechanics with FEM

5.1 1D continuous elasticity

5.1.1 Hooke's law

$$\sigma = E\epsilon$$

Stress (Spannung): $\sigma = \frac{f_{int}}{A}$

Strain (Dehnung): $\epsilon = \frac{\Delta l}{L}$

Young's elasticity modulus: $E \in \mathbb{R}$

For these definitions, an equivalent form of Hooke's law reads

$$f_{int} = EA\frac{\Delta l}{L}$$

This form is valid only if the elongation is linear to the force. In general, this is only true for small elongations.

5.1.2 Strong form (equilibrium equations)

$$\begin{aligned} EA\partial_{xx}u(x) &= -f_b(x), x \in]0,1[\\ EA\partial_x u(1) &= f^s \\ u(0) &= 0 \end{aligned}$$

$u = x - x'$: deformation

Note that we have a Neumann boundary condition at x=1 and a Dirichlet boundary condition at x=0

5.1.3 Weak form

$$\int_0^1 EA\frac{\partial}{\partial x}u\frac{\partial}{\partial x}v\,dx = \int_0^1 f^b v\,dx + \underbrace{f^s v(1)}_{Neumann\ B.C.}$$

5.2 3D deformations

5.2.1 Hooke's law

Hooke's law also applies in 3 dimensions

$$\sigma = E\epsilon$$

Stress tensor (Spannung): $\boldsymbol{\sigma}(x) = \begin{pmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z \end{pmatrix}$. $\boldsymbol{\sigma}$ is a 3x3 symmetric matrix

Strain tensor (Dehnung): $\boldsymbol{\epsilon}(x) = \begin{pmatrix} \epsilon_x & \gamma_{xy} & \gamma_{xz} \\ \gamma_{xy} & \epsilon_y & \gamma_{yz} \\ \gamma_{xz} & \gamma_{yz} & \epsilon_z \end{pmatrix}$. $\boldsymbol{\epsilon}$ is a 3x3 symmetric matrix

Young's elasticity modulus matrix: $\boldsymbol{E} \in \mathbb{R}^{9,9}$

Note: $\sigma = E\epsilon$ is computed by reshaping σ and ϵ to 9x1 vectors.

Note: Since σ and ϵ are symmetric, they can be reduced to 6x1 vectors. In this case, $E \in \mathbb{R}^{6,6}$

Again, Hooke's law is only valid for small deformations.

5.2.2 Green strain

$$\epsilon_G = \frac{1}{2}\left(\boldsymbol{F^T F} - \boldsymbol{I}\right) = \frac{1}{2}(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T + \nabla \boldsymbol{u}^T \nabla \boldsymbol{u})$$

$F = I + \nabla u$: Deformation gradient

$$\boldsymbol{\nabla u} = \begin{pmatrix} \partial_x u & \partial_y u & \partial_z u \\ \partial_x v & \partial_y v & \partial_z v \\ \partial_x w & \partial_y w & \partial_z w \end{pmatrix}$$

$\boldsymbol{u}$ is the displacement vector, i.e. $\boldsymbol{x'} = \boldsymbol{u} + \boldsymbol{x}$

Problem: Green strain is nonlinear in **u**.

5.2.3 Cauchy strain

Linearized version of Green strain

$$\epsilon_C = \frac{1}{2}(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T)$$

$$\epsilon_C = \frac{1}{2}\begin{pmatrix} 2\partial_x u & \partial_y u + \partial_x v & \partial_z u + \partial_x w \\ \partial_x v + \partial_y u & 2\partial_y v & \partial_z v + \partial_y w \\ \partial_x w + \partial_z u & \partial_y w + \partial_z v & 2\partial_z w \end{pmatrix}$$

For the reduced 6x1 vector form,

$$\widetilde{\epsilon_C} = \frac{1}{2}\begin{pmatrix} 2\partial_x u \\ 2\partial_y u \\ 2\partial_z u \\ \partial_y u + \partial_x v \\ \partial_z v + \partial_y w \\ \partial_z u + \partial_x w \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 2\partial_x & & \\ & 2\partial_y & \\ & & 2\partial_z \\ \partial_y & \partial_x & \\ & \partial_z & \partial_y \\ \partial_z & & \partial_x \end{pmatrix} \cdot \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

Problem: Cauchy strain is not rotational invariant, but Green strain is rotational invariant.

5.2.4 Strong form

$$\nabla \cdot \sigma = -f^b(x), x \in V$$

$$\sigma(x) \cdot n(x) = f^s(x), x \in S^f$$
$$u(x) = u^s(x), x \in S^u$$

Note that on S^f, Neumann boundary condition apply, while Dirichlet boundary conditions apply on S^u

5.2.5 Weak form

$$\int_V \epsilon(\overline{\boldsymbol{u}})^T \boldsymbol{E} \epsilon(\boldsymbol{u}) dV = \int_V \overline{\boldsymbol{u}}^T \boldsymbol{f}^b dV + \int_{S^f} \overline{\boldsymbol{u}}^T \boldsymbol{f}^S \, dS^f$$

Where $\overline{u}$ is the test function. $u, \overline{u} \in \mathbb{R}^3$, $\epsilon(\boldsymbol{u}) \in \mathbb{R}^6$, $\boldsymbol{E} \in \mathbb{R}^{6,6}$. Note that ϵ is written in the 6x1 vector form, but is equivalent to a 3x3 symmetric matrix. $\epsilon(u)$ is computed by discretising u and then perform the matrix-vector multiplication with $\widetilde{\epsilon_C}$ (or any other strain matrix).

- The dimension of the Galerkin matrix is 3n x 3n for n nodal points.
- The local stiffness matrices have dimension 9x9. For linear tetrahedral elements (volume discretisation), the local stiffness matrices have dimension 12x12.

5.2.6 Linear dynamic deformations

The strong form of the PDE becomes

$$\nabla \cdot \boldsymbol{\sigma} = -\boldsymbol{f}^b(x) \rightarrow \rho \ddot{\boldsymbol{u}} = \nabla \cdot \boldsymbol{\sigma} + \boldsymbol{f}^b$$

This leads to a matrix equation

$$M\ddot{u} + Ku + f$$

This can be solved using an ODE integrator.

5.2.7 Corotational strain

Problem: Cauchy strain creates ghost forces due to rotation.

Insight: Object displacements consists of deformation and rotation.

Idea: Compute Cauchy strain on deformation only and apply rotation separately.

1. Compute object rotation between two timesteps
2. Rotate positions back to get object which is displaced only by deformation
3. Apply Cauchy strain on this object and get forces.
4. Rotate forces and apply them.

5.2.8 Summary

	Cauchy strain	**Green strain**	**Corotational strain**
Linearity	Linear	Non-linear	Linear
Artifacts	Not rotational invariant	Accurate for arbitrary deformations	No visual artifacts
Numerical stability / efficiency	Stable and fast	Slow and potentially unstable	Robust

6 Fluid simulation

6.1 Eulerian (grid-based) viewpoint

6.1.1 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)

6.1.2 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.

6.1.2.1 Advection

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

6.1.2.2 Viscosity / Diffusion

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

Since the viscosity of water is nearly zero, this term can be dropped.

6.1.2.3 Gravity

$$g = g$$

Embarrassingly easy.

6.1.2.4 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.

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

The discretisation for the two-dimensional Laplace operator reads

$$\nabla^2 f \approx \frac{4f_{i,j} - f_{i+1,j} - f_{i,j+1} - f_{i-1,j} - f_{i,j-1}}{\Delta x^2}$$

Applying this discretisation to the pressure equation, a linear system $Ap = d$ must be solved. A is sparse and symmetric.

6.1.2.4.1 Boundary conditions

Solid boundaries (like walls): $p_{wall} = p_{fluid}(\partial\Omega)$

Free surface boundaries (like water-air): Linear interpolation between fluid cell and air cell

6.2 Lagrangian (particle-based) viewpoint (SPH)

6.2.1 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.

6.2.2 Discretisation

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

6.2.2.1 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
- Differentiation only affects the kernel.

6.2.2.2 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: $W_{ij} \coloneqq W(x_{ij}, h)$

6.2.2.3 Density

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

6.2.2.4 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:

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

6.2.2.5 *Incompressibility*

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

6.2.2.6 *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)

6.2.2.7 *Viscosity force density*

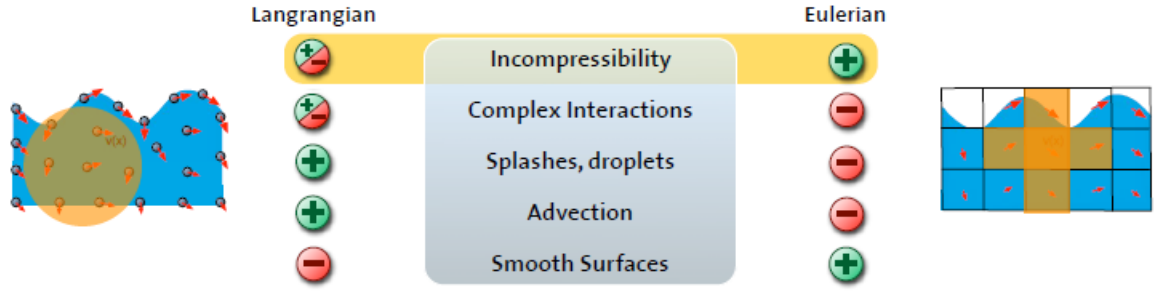
$$\mu \nabla^2 \boldsymbol{u} = \mu \sum_j \frac{m_j}{\rho_j} \text{abs}(\boldsymbol{u_j} - \boldsymbol{u_i}) \nabla^2 W_{ij}$$

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.

6.2.3 Algorithm overview

1. Compute neighbourhoods (with cell lists. Cell lists reduce computational cost from $O(N^2)$ to $O(9cN)$ where c is the number of particles in the cells)
2. Compute densities and velocities
3. Compute forces
4. Compute new velocity, new position
5. Do collision handling
6. Start again

6.3 Comparison



	Eulerian	Lagrangian / SPH
Incompressibility	Is an assumption	Leads to high stiffness coefficient. This can be improved by using an iterative predictor-corrector scheme to estimate quantities in next timestep (PCISPH)
Splashes, droplets	Difficult	Easy, on a particle level
Advection	Not so easy	Trivial
Smooth surfaces		Bumps related to particle distribution are visible. Remedy: More particles

6.4 Extensions

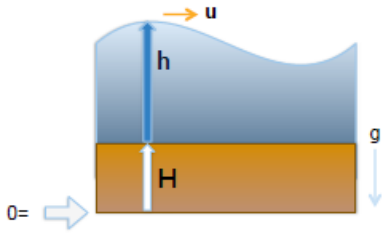
6.4.1 Multiscale methods

Idea: Use a hierarchy of particle sizes, e.g. coarse resolution for bulk, fine resolution for surface, droplets etc.

Task: Define interaction between regions of different particle resolution.

6.4.2 Shallow water equations

Shallow water equations are derived from the Navier-Stokes equations, but are only two-dimensional. The third dimension is reduced to a 2D height field.



$$\frac{Dh}{Dt} = -h\nabla \cdot \boldsymbol{u}$$
$$\frac{D\boldsymbol{u}}{Dt} = -g\nabla(h + H) + \boldsymbol{a}_{ext}$$

Note that the ∇ operator in the second equation is a gradient.

Algorithm

1. Advect h (Semi-Lagrangian)
2. Advect u_1, u_2
3. Update height (Central differences on a staggered grid)
4. Update velocities

6.4.2.1 Boundaries

Reflecting boundaries

- Applies to walls
- No flux through wall: x-component of velocity at boundaries set to zero.

Absorbing boundaries

- Applies to open water surface
- Damping of velocity

6.4.2.2 Rigid body coupling

Combining SWE and 2D SPH

Idea: Interpret each position as a cylinder with a certain height. Link SWE and SPH by interpreting the height as density. Translate heights to forces which are applied on the rigid body.

6.4.3 Turbulence

Turbulence is largely based on underlying resolution.

Problem: Inaccurate advection causes diffusion => loss of energy.

Idea: Reintroduce this energy as vorticity. $\omega = \nabla \times u$

Algorithm

1. Compute vorticity $\boldsymbol{\omega} = \nabla \times \boldsymbol{u}$
2. Add forces in tangential direction $\boldsymbol{f} = c\,\boldsymbol{n} \times \boldsymbol{\omega}$
   c is a user-definable strength

7 Collision detection

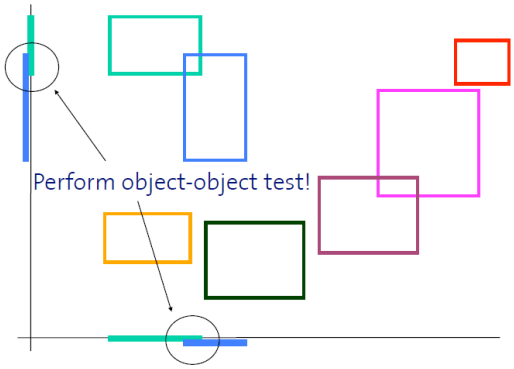
7.1 General algorithm

1. Broad phase
   - Collision of bounding volumes
2. Narrow phase
   - Collision of primitives (culling)

7.2 Collision algorithms

7.2.1 Sweep and prune

Requires Axis-Aligned Boundary-Boxes (AABB)



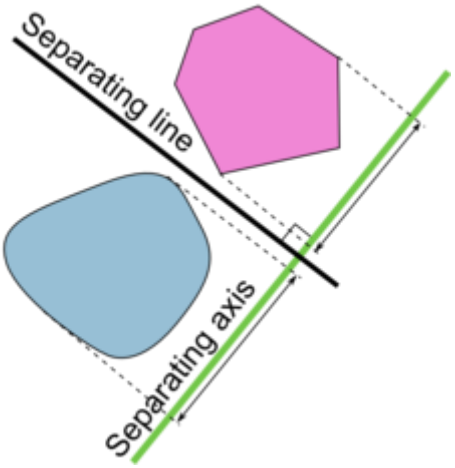
1. Sort AABB of every object: $O(n \log n)$. Sorting with temporal coherence using insertion sort: $O(n)$
2. Report intersecting pairs: $O(k)$

7.2.2 Separating Axis Test (SAT)

Applies for any **convex** boundary box.

In easy terms, the hyperplane separation theorem states:

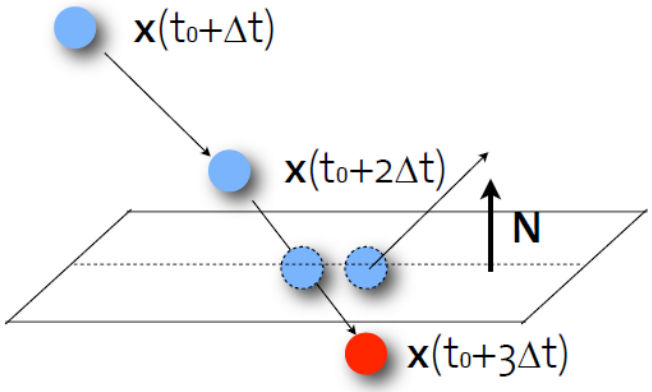
- For two convex boundary boxes, find the minimal distance between them. This is the separating axis.
- If such a axis exists, every hyperplane which is orthogonal to the separating axis separates the two boundary boxes.



Therefore, SAT is reduced to finding a “non-separating” axis. That is, find any two points on the two bounding volumes for which the (oriented) distance is < 0.

In a naïve implementation, such a non-separating axis is found or not found after k^2 (squared) distance operations, where k is the number of corners of the bounding box.

7.2.3 Particle collisions



compute new position (intersection point)

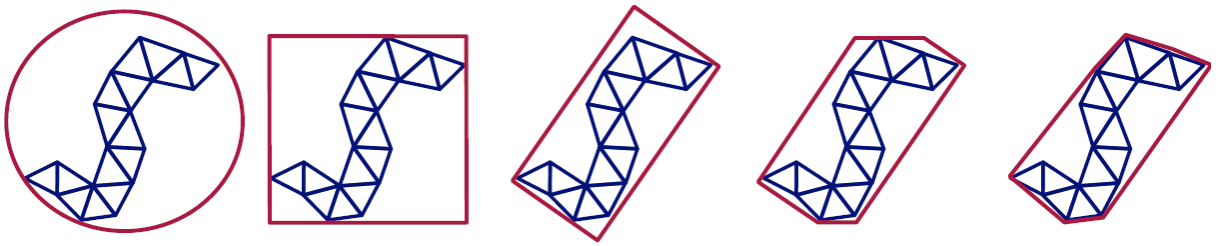
backstep

project onto plane

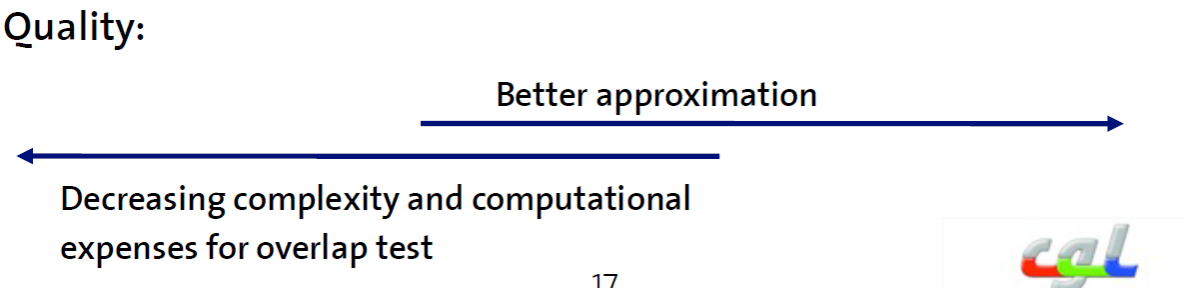
compute new velocity

Artifact: Forces pile up at boundary

7.3 Bounding volumes



Sphere ABB OBB 6-DOP Convex Hull



- ABB: Axis-aligned bounding box
- OBB: Oriented bounding box: Computed using PCA.
- 6-DOP: 6-Discrete orientation polytope: Fixed set of k oriented planes. ABB is a 2-DOP.

Problem: Parallel Close Proximity

Bounding boxes collide, although there is no collision. => Expensive primitive collision detection (narrow phase) must be done.

7.3.1 Bounding volume hierarchies

A hierarchical coarse-to-fine strategy can be applied for bounding volumes.

There are two possibilities to construct a bounding volume tree for an object.

- Top-down: Fit a bounding volume to the object and recursively split the object.
- Bottom-up: Fit bounding volumes to the primitives of an object and group these bounding volumes recursively.

Problem: In case of deformation of an object, the BV tree needs to be rebuilt in each time step.

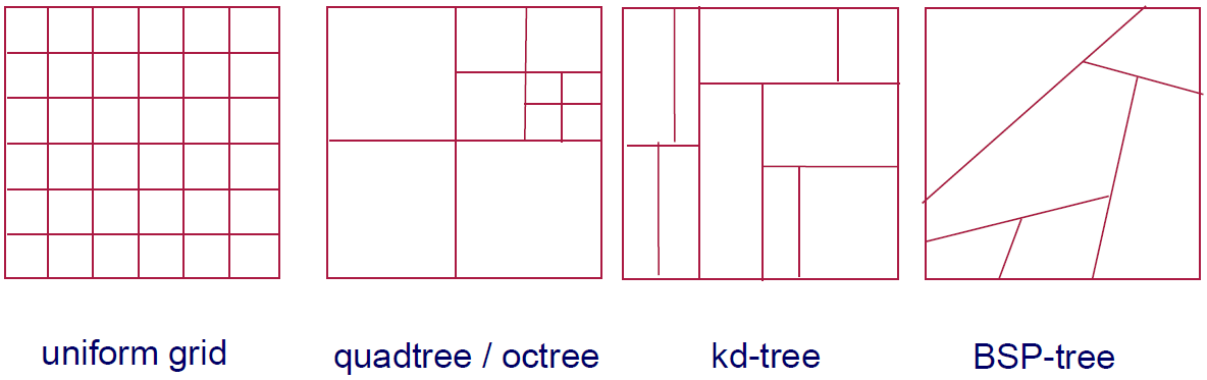
7.4 Spatial partitioning

Alternative or extension of bounding volume hierarchies.

- Divide space into cells
- Only primitives within the same cell are checked for collision

Advantages over bounding volume hierarchies

- +: Efficient handling of object deformations (not whole structure needs to be updated)



7.4.1 Cell-based partition and hashing

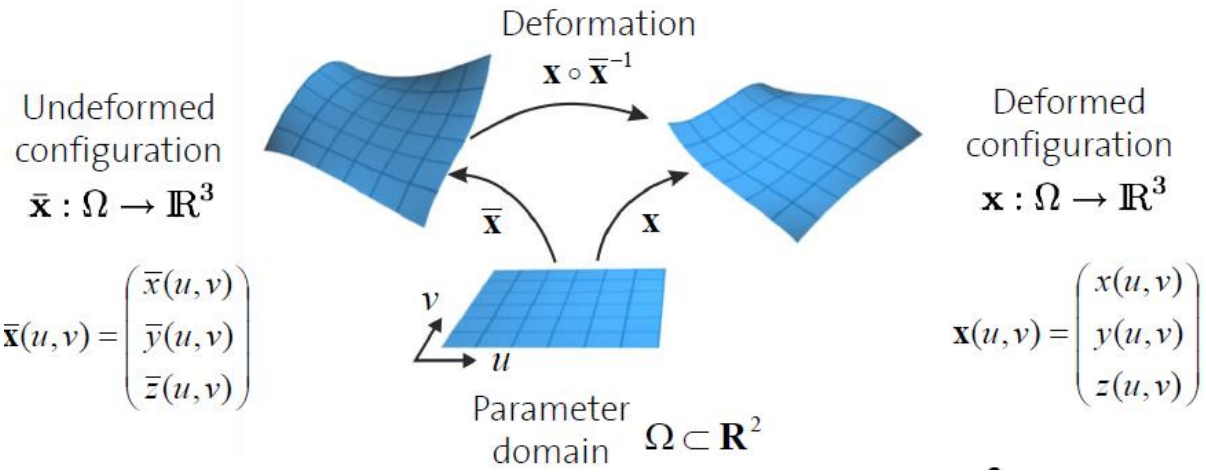
Given: Vertices, triangles (triple of vertex indices), uniform grid

1. Hash all vertices according to their grid cell
2. Hash all triangles according to all grid cells that they (partially) overlap.
3. For every hash element, check if an indexed vertex and an indexed triangle overlap (Yes: intersection, No: no intersection. Exception: Overlapping vertex is from the same triangle)

8 Shells and rods

- Applicable to: Alu cans, hats, leaves
- Unique Deformation: Strong resistance to stretching, but much less resistance to bending => Buckling

8.1 Parameterized midsurface representation



Kirchhofe-Love Hypothesis

1. Normals stay perpendicular to surface
2. Thus, no deformation in thickness direction

8.2 Surface metrics

First fundamental form, describes metric on surface

$$\boldsymbol{I} = \begin{bmatrix} \boldsymbol{x}_u^T \boldsymbol{x}_u & \boldsymbol{x}_u^T \boldsymbol{x}_v \\ \boldsymbol{x}_v^T \boldsymbol{x}_u & \boldsymbol{x}_v^T \boldsymbol{x}_v \end{bmatrix}$$

Where $\boldsymbol{x}_u = \frac{\partial \boldsymbol{x}}{\partial u}$

Second fundamental form, describes curvature

$$\boldsymbol{II} = \begin{bmatrix} \boldsymbol{x}_{uu}^T \boldsymbol{n} & \boldsymbol{x}_{uv}^T \boldsymbol{n} \\ \boldsymbol{x}_{vu}^T \boldsymbol{n} & \boldsymbol{x}_{vv}^T \boldsymbol{n} \end{bmatrix}$$

8.2.1 Deformation

$$d^T d - \bar{d}^T \bar{d} = d^T (I - \bar{I}) d$$

Where $d = x_2 - x_1$ the distance between two arbitrary points, and overbar denotes the basic state. This is quite similar to the deformation in continuum mechanics and FEM.

8.3 Governing equation

Shell deformation energy

$$E_S = \int_\Omega \underbrace{k_S \|I - \bar{I}\|^2}_{In-plane\ deformation} + \underbrace{k_b \|II - \bar{II}\|^2}_{Bending\ deformation} \, d\Omega$$

Energy minimization

Minimize

$$E_S(u) + E_{ext}(u)$$

E_{ext} is an external force.

Problems

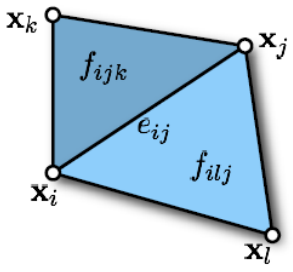
- E_S nonlinear in u.
- u must be twice differentiable
- Requires parameterization of curved shells
- => Only theoretical use

8.4 Discrete shells

Idea: Since discretising continuous energy formulation is too complicated, define a discrete energy.

- Stretching: Change of edge lengths

$$E_{DS} = k_{\text{stretch}} \cdot \sum_{e_{ij} \in E} |\bar{e}_{ij}|^2 \left(\frac{|e_{ij}|}{|\bar{e}_{ij}|} - 1 \right)^2$$

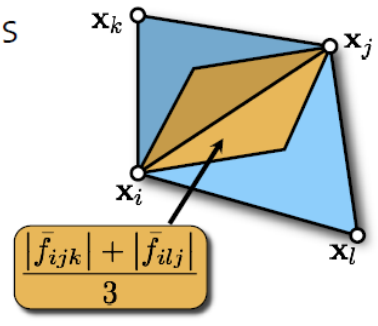


- Stretching: Change of triangle areas

$$E_A = k_{\text{area}} \cdot \sum_{f_{ijk} \in F} |\bar{f}_{ijk}| \left(\frac{|f_{ijk}|}{|\bar{f}_{ijk}|} - 1 \right)^2$$

- Bending: Change of dihedral angles

$$E_B = k_{\text{bend}} \cdot \sum_{e_{ij} \in E} \frac{3 |\bar{e}_{ij}|^2}{|\bar{f}_{ijk}| + |\bar{f}_{ilj}|} (\theta_{ij} - \bar{\theta}_{ij})^2$$



- Total energy

$$E_{DS} = E_L + E_A + E_B$$

28

8.4.1 Time integration

- Stretching forces are stiff, require implicit integration

How to integrate bending forces?

- Soft shells and cloth
  - Bending forces are small → integrate *explicitly*
  - Only energy gradients (*forces*) are required
- Rigid shells
  - Bending forces can be large → integrate *implicitly*
  - Additionally requires energy Hessian (*force Jacobian*)
  - Involved derivations & computations

8.4.2 Mass-spring systems

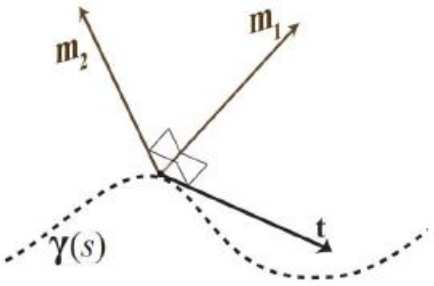
Diagonal springs don't work for shells

- Undeformed configuration is curved
- Incorrect (ambiguous) energy minima

8.5 Elastic rods

8.5.1 Kirchhoff rod model

- Centerline: $\gamma(s)$
- Orthonormal material frame: $\{t(s), m_1(s), m_2(s)\}$
- Material frame is adapted to centreline, i.e. $t(s) = \gamma'(s)$



8.5.2 Deformation

Deformation can be due to bending and twisting, not through stretching

Bending

$$b_1 = \boldsymbol{t}' \cdot \boldsymbol{m}_1$$

$$b_2 = \boldsymbol{t}' \cdot \boldsymbol{m}_2$$

Twisting

$$t = \boldsymbol{m}_1' \cdot \boldsymbol{m}_2$$

x' denotes evolution of x along curve

Energy of elastic rod

$$E = \int_0^t \underbrace{k_{b1} b_1(s)^2 + k_{b2} b_2(s)^2}_{Bending} + \underbrace{k_t t(s)^2}_{Twisting} \, ds$$

8.5.3 Curve-angle representation

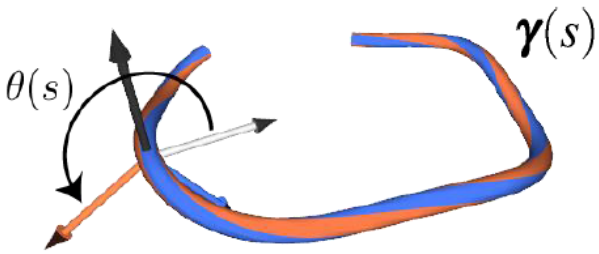
Explicit centerline

Reference frame

- $\{\mathbf{u}, \mathbf{v}\}$ twist-free

Material frame

- relative angle



Coordinates

$\boldsymbol{\gamma}(s) : \mathbb{R} \rightarrow \mathbb{R}^3$

$\boldsymbol{\theta}(s) : \mathbb{R} \rightarrow \mathbb{R}$

Bending

$\boldsymbol{\kappa} = \boldsymbol{\gamma}''$

Twisting

$m = \boldsymbol{\theta}'$

Disney Research