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
  + Position
  + Velocity

### Forces

* **Forces** = external forces + internal forces
* Internal forces: **Elastic springs**
  + 1D:
  + 3D:   
    where is the spring constant, is the elongated length and is the original length. and are the end positions of the spring, i.e.
  + Total spring force at mass point 0 with being the set of adjacent mass points.
* Internal forces: **Dissipation / Point damping**

## Equations of motion

* For each mass point:
* Coupled first order problem: , where  
  ,

## First-order numerical integration of ODE’s

**Local error** (single step):

**Global error** (accumulated):

**Accuracy**: Method is accurate of order p, if the local error is . The accuracy can be determined by comparing the update formula with the Taylor expansion.

### General Runge-Kutta methods

* Given the **IVP**:
* **s-step RK solution**:
* Intermediate steps:
* The defining coefficients a,b and c are conveniently depicted in the Butcher scheme

### Euler method

* Butcher scheme:
* Update formula:
* EOM:
* Accuracy: Order 1, error per step

### Heun’s method

* Butcher scheme:
* Idea: . For 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:
* Accuracy: Order 2, error per step

### Midpoint rule

* Butcher scheme:
* Accuracy: Order 2

### Backwards Euler

* Butcher scheme:
* Update formula:
* Generally involves solving a set of nonlinear equations
* Accuracy: Order 1

### Semi-implicit Euler

* Same as backwards Euler, but approximate 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

* +: Accuracy: Order 2
* +: Only one force evaluation
* -: A posteriori approximation of velocities
* -: Two-step method problematic at discontinuities

### Leapfrog

* +: Accuracy: 2nd order
* +: Only one force evaluation
* -: a(t) must not depend on v(t) => no damping

### Symplectic Euler

* -: Accuracy: 1st order
* +: Good stability for oscillatory motion
* +: Good conservation of momentum and energy

## Stability

* Apply integration scheme to test equation, e.g.
* Von Neumann stability check, i.e. assume solution to be of form
* CFL-criterion: , 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 is satisfied if and only if

# Fluid simulation

## Eulerian (grid-based) viewpoint

### Navier Stokes Equation

With: (Continuity equation)

### Discretisation

Separate time-dependent NSE part.

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

#### Advection

: Use Semi-Lagrangian advection

#### Diffusion

#### Gravity

Embarrassingly easy.

#### Pressure equation and divergence-freedom

Pressure is defined through the continuity equation. Therefore,

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

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

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

## Lagrangian (particle-based) viewpoint

### Navier-Stokes equation

* Force density formulation. Note:
* 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

* : Quantity A (e.g. density, pressure, etc.)
* Neighborhood of x. Concretely,
* Particle volume
* Smoothing kernel

#### Kernel properties

* Normalisation condition:
* Dirac delta limit:   
  Actually, if the kernel was a Dirac delta function, the particles would represent individual molecules.
* Compact condition: for
* Example:
* Notation:

#### Density

#### Pressure

Equation of state:

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

Additionally, for stability:

#### Incompressibility

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

#### Pressure force density

Note: instead of only is used to guarantee a symmetric force. (Actio = reactio)

#### Viscosity force density

Note: abs stands for element-wise absolute value. 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