Physically Based Simulation

Alexander Schlögl

February 3, 2018

Contents

1 **Introduction** 2

2 **Mass Spring Systems** 2

2.1 External Forces f^{Ext} 2

2.2 Internal Forces f^{Int} 2

2.3 Spring Topologies and Parameters 3

2.4 Equations of Motion 3

2.5 Numerical Integration Methods 3

Euler Method 4

Heun's Method 4

Midpoint Method 5

Runge-Kutta Methods 5

Verlet Method 6

Leapfrog Method 6

Implicit Euler 6

2.6 Test Equation 7

2.7 Collision Handling 7

2.8 Problems of Mass Spring Systems 7

This is **my interpretation** of the lecture slides. I tried to be very verbose and explain everything, all while removing irrelevant parts from the lecture. Using this you should be able to pass the lecture easily. **However, I do not take responsibility for any bad results and will not be blamed from anyone. This was a lot of work and I did it to save others (especially students of following semesters) from having to do this themselves. Use this summary at your own responsibility.** If you have any feedback, feel free to create an issue on the git. I don't promise I will fix anything, but I will try.

1 Introduction

Physically based simulation is exactly what it sounds like. Creating accurate visualizations of real world objects by simulating the underlying physical equations. While doing so, a number of shortcuts is taken to speed up computation time. We do not need very accurate results, we need results that **look accurate**.

2 Mass Spring Systems

Mass Spring Systems are a simple way of simulating deformation, and they are often used for hair, clothes etc. The main steps of employing mass spring systems are:

1. Discretizing the object into mass points and connecting them with springs
2. Setting up the equations of motion
3. Discretizing the equations of motion in time
4. Determining internal and external forces acting on mass points
5. Solving the equations numerically

The location and mass distribution of the discretized points has a large influence on the accuracy of the simulation. The simplest approximation is just giving all points equal mass, but better methods exist. Each point has a position $\boldsymbol{x_i}$, a velocity $\boldsymbol{v_i}$ and a mass $\boldsymbol{m_i}$. The points are then connected with springs. The spring topology also heaviliy influences the accuracy, and certain heuristics exist. Each spring stores a stiffness k_q, rest length L_q and current length l_q.

2.1 External Forces f^{Ext}

External forces include gravity ($f_i^G = m_i \begin{bmatrix} 0 & 0 & 9.81 \end{bmatrix}^T m/s^2$), friction and collisions.

2.2 Internal Forces f^{Int}

The internal forces are the forces exerted by the springs, as well as (viscous) damping. Spring force is given by Hooke's law:

$$f = k(L - l) \tag{1}$$

or in 3D

$$\boldsymbol{f_{ij}} = k(L - \|\boldsymbol{x_i} - \boldsymbol{x_j}\|)\frac{\boldsymbol{x_i} - \boldsymbol{x_j}}{\|\boldsymbol{x_i} - \boldsymbol{x_j}\|} \tag{2}$$

for multiple springs connected to a single mass point, just sum up all the individual spring forces.

2.3 Spring Topologies and Parameters

In general springs are connected in three different ways. Structural springs connect neighboring points and oppose stretching; diagonal strings oppose shearing and interleaving springs oppose bending. For cloth or other objects that need to be bent, springs that skip multiple points can be used.

The stiffness for the springs can be calculated with existing heuristics or be set in such a way that they accurately represent known deformations. Usually manual tuning is required.

2.4 Equations of Motion

As the mass of the points, as well as the forces acting on them is known, we can simulate them using Newton's laws of motion. From Newton's second law follows

$$m_i \frac{d^2 \boldsymbol{x_i}(t)}{dt^2} = \boldsymbol{f_i^{Int}}(t) + \boldsymbol{f_i^{Ext}}(t) \tag{3}$$

with which we can calculate the acceleration and update the velocity of the mass point. Equation (3) is a first order ordinary differential equation (ODE), which we can solve and the numerically integrate. For the integration step we differentiate between explicit methods (where all required quantities are known) and implicit methods (where we need to solve a system of equations).

2.5 Numerical Integration Methods

As we actually need two integrations for our systems (one for $\boldsymbol{v}$ and one for $\boldsymbol{x}$) we need two seperate integration steps. For this we simply introduce a variable for velociy and integrate it to get the new position.

All integration methods are derived from Taylor series expansion, and some use multiple Taylor series or more parts of it to reach higher accuracy. As a reminder, the Taylor series for $y(x)$ about development point t is

$$y(x) = \sum_{n=0}^{\infty} \frac{y^{(n)}(t)}{n!}(x-t)^n \tag{4}$$

and if we evaluate this at point $t+h$ we get

$$y(t+h) = y(t) + y'(t)h + \frac{y''(t)}{2!}h^2 + \frac{y'''(t)}{3!}h^3 + ... \tag{5}$$

From that we can derive our numerical integration methods and their accuracy.

Euler Method

Forward Euler is the simplest integration method. It is derived by using the first two summands of the Taylor series, giving us the following equation

$$y(t+h) = y(t) + y'(t)h + \mathcal{O}(h^2) \tag{6}$$

with our error being a function in the realm of $\mathcal{O}(h^2)$.

The following calculations are done for the Forward Euler:

- $\boldsymbol{x}(t+h) = \boldsymbol{x}(t) + h \cdot \boldsymbol{v}(t)$
- $\boldsymbol{f}(t) = \boldsymbol{f}^{Int}(t) + \boldsymbol{f}^{Ext}(t)$
- $\boldsymbol{a}(t) = \frac{1}{m}(\boldsymbol{f}(t) - \gamma\boldsymbol{v}(t))$ (where γ is the damping factor)
- $\boldsymbol{v}(t+h) = \boldsymbol{v}(t) + h \cdot \boldsymbol{a}(t)$

In order to improve behaviour of the Euler method, we can either directly reuse new positions

$$\boldsymbol{x}(t+h) = \boldsymbol{x}(t) + h\boldsymbol{v}(t) \tag{7}$$
$$\boldsymbol{v}(t+h) = \boldsymbol{v}(t) + h\boldsymbol{a}(\boldsymbol{x}(t+h), \boldsymbol{v}(t)) \tag{8}$$

or directly reuse the new velocities

$$\boldsymbol{v}(t+h) = \boldsymbol{v}(t) + h\boldsymbol{a}(\boldsymbol{x}(t), \boldsymbol{v}(t)) \tag{9}$$
$$\boldsymbol{x}(t+h) = \boldsymbol{x}(t) + h\boldsymbol{v}(t+h) \tag{10}$$

Using these improvements is known as the Symplectic Euler.

Heun's Method

This integrator uses more summands in its approximation:

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2}y''(t) + \mathcal{O}(h^3) \tag{11}$$

As we don't know the second derivative, we approximate it with forward differences:

$$y''(t) = \frac{y'(t+h) - y'(t)}{h} + \mathcal{O}(h) \tag{12}$$

By plugging Equation (12) into Equation (11) we get

$$y(t+h) \approx y(t) + hy'(t) + \frac{h^2}{2}(\frac{y'(t+h) - y'(t)}{h}) \tag{13}$$
$$\approx y(t) + hy'(t) + \frac{h}{2}y'(t+h) - \frac{h}{2}y'(t) \tag{14}$$
$$\approx y(t) + \frac{h}{2}(y'(t) + y'(t+h)) \tag{15}$$

As we don't know $y'(t+h)$ yet when we evaluate this, we approximate it with a regular Euler step.

Heun's method uses the average of the current velocity and the next velocity for the update step

The implementation steps for Heun's method are the following

- $\boldsymbol{a}(t) = \frac{1}{m}(\boldsymbol{f}(t) - \gamma \boldsymbol{v}(t))$
- $\tilde{\boldsymbol{v}}(t+h) = \boldsymbol{v}(t) + h\boldsymbol{a}(t)$
- $\tilde{\boldsymbol{x}}(t+h) = \boldsymbol{x}(t) + h\tilde{\boldsymbol{v}}(t)$
- $\tilde{\boldsymbol{a}}(t+h) = \frac{1}{m}(\tilde{\boldsymbol{f}}(t+h) - \gamma \tilde{\boldsymbol{v}}(t+h))$
- $\boldsymbol{x}(t+h) = \boldsymbol{x}(t) + h\frac{\boldsymbol{v}(t)+\tilde{\boldsymbol{v}}(t+h)}{2}$
- $\boldsymbol{v}(t+h) = \boldsymbol{v}(t) + h\frac{\boldsymbol{a}(t)+\tilde{\boldsymbol{v}}(t+h)}{2}$

Midpoint Method

Heun's method averages $y'(t)$ and $y'(t+h)$. The midpoint method only determinse y' at $t+h/2$, giving us the same accuracy with less work. We approximate $y'(t+h/2)$ with an Euler step.

For our mass spring use case we need to approximate both $\tilde{\boldsymbol{v}}$ and $\tilde{\boldsymbol{a}}$ (and thus $\tilde{\boldsymbol{x}}$) for $t+h/2$.

Runge-Kutta Methods

These are more general cases of the Midpoint method. They increase accuracy by performing intermediate steps. The most common variant is RK4. Due to the increased accuracy, larger timesteps can be used, but this increases computational overhead.

RK4 works as follows:

$$k_1 = f(t, y(t)) \tag{16}$$
$$k_2 = f(t+h/2, y(t) + h/2k_1) \tag{17}$$
$$k_3 = f(t+h/2, y(t) + h/2k_2) \tag{18}$$
$$k_4 = f(t, y(t) + hk_3) \tag{19}$$

The value for $y(t+h)$ is the calculated using a weighted average

$$y(t+h) = y(t) + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{20}$$

Implementation of RK4 (or any RK method) should be pretty clear.

Verlet Method

For this method we start with two Taylor series expansions:

$$y(t+h) = y(t) + y'(t)h + \frac{y''(t)}{2!}h^2 + \frac{y'''(t)}{3!}h^3 + \mathcal{O}(h^4) \tag{21}$$

$$y(t-h) = y(t) - y'(t)h + \frac{y''(t)}{2!}h^2 - \frac{y'''(t)}{3!}h^3 + \mathcal{O}(h^4) \tag{22}$$

then, by summing them up we get

$$y(t+h) = 2y(t) - y(t-h) + h^2 y''(t) + \mathcal{O}(h^4) \tag{23}$$

As you can see the velocity terms cancels out, leading to the typical assumption that the force term is independent of velocity (no damping). The Verlet method is a symplectic integrator (it has nice energy conservation properties in Hamiltonian systems) and is also time reversible.

The velocity term here is never explicitly calculated. If we do need it (e.g. for kinetic energy) we need to approximate it either via backward or central differences.

Leapfrog Method

The main goal of this method is to improve the velocity estimate. It is similar to the Verlet method but from the Taylor series for velocity:

$$\boldsymbol{v}(t+h) = \boldsymbol{v}(t) + \boldsymbol{v}'(t)h + \frac{\boldsymbol{v}''(t)}{2!}h^2 + \frac{\boldsymbol{v}'''(t)}{3!}h^3 + \mathcal{O}(h^4) \tag{24}$$

$$\boldsymbol{v}(t-h) = \boldsymbol{v}(t) - \boldsymbol{v}'(t)h + \frac{\boldsymbol{v}''(t)}{2!}h^2 - \frac{\boldsymbol{v}'''(t)}{3!}h^3 + \mathcal{O}(h^4) \tag{25}$$

and by subtracting we get

$$\boldsymbol{v}(t+h) = \boldsymbol{v}(t-h) + 2\boldsymbol{v}'(t)h + \mathcal{O}(h^3) \tag{26}$$

The Leapfrog method then employs shifted updates, with $\boldsymbol{v}$ being updated for $t + h/2$ and $\boldsymbol{x}$ and $\boldsymbol{a}$ being updated for t. This gives us an error of only $\mathcal{O}(h^3)$ with only one force evaluation. We need to initialize $\boldsymbol{v}(t_{1/2})$ with e.g. an Euler step.

Implicit Euler

The Implicit Euler formulates the update equation like as follows

$$y_{n+1} = y_n + h \cdot f(t_{n+1}, y_{n+1}) \tag{27}$$

Due to this, we have to solve for an unknown in every update step. One option is the **fixed-point iteration**

$$y_{n+1}^{(k+1)} = y_n + h \cdot f(t_{n+1}, y_{n+1}^{(k)}) \tag{28}$$

which requires initialization such as $y_{n+1}^{(0)} = y_n$ or $y_{n+1}^{(0)} = y_n + h \cdot f(t_n, y_n)$.

Another option would be **Newton's Method**.

2.6 Test Equation

We can study stability behaviour of a solver via a linear test equation (Dahlquist's equation):

$$y'(t) = \lambda \cdot y(t) \tag{29}$$

which gives us the following solution for $y(t)$ (assuming $\lambda \in \mathbb{R}$)

$$y(t) = e^{\lambda t} y_0 \tag{30}$$

An example for the forward Euler: The update step is

$$y_{n+1} = y_n + h \cdot y'_n \tag{31}$$

by inserting the test equation we get

$$y_{n+1} = (1 + \lambda h) y_n \tag{32}$$

which with recursive evaluation gives us

$$y_{n+1} = (1 + \lambda h)^{n+1} y_0 \tag{33}$$

This shows that the forward Euler is unconditionally unstabel if $\lambda > 0$ and stable iff $-2 < h\lambda < 0$.

2.7 Collision Handling

When using mass spring systems, a simple form of collision detection and handling can be required. For this, penalty forces are used. In the penalty forces approach any penetration of an object into another creates a spring force separating both objects. This approach is only an approximation, but can be very useful due to its simplicity.

2.8 Problems of Mass Spring Systems

The behaviour of the system is topology dependent. Also, we do not have an explicit notion of volume preservation, which can result in systems collapsing on themselves. This is again dependent on their topology.