Abstract

English title

Keywords: k1, k2

Streszczenie

Tytuł polski Tytuł polski Tytuł polski

Słowa kluczowe: slowo1, slowo2

Bolesław Prus Warsaw, ...

Nr albumu 100000

Declaration

I hereby declare that the thesis entitled „English title”, submitted for the magisters degree, supervised by dr inż. Promotor Promotorski, is entirely my original work apart from the recognized reference.

....

Bolesław Prus

Spis treści

Introduction

Soft Body Dynamics - Overview

Mass Spring

Formulation

The simplest method to simulate soft body deformation is Mass Spring System. Such a system includes a set of N particles with masses m_i, positions x_i and velocites v_i, where $i \in 1...N$. The particles are connected by a set of springs S. A single spring $s \in S$ consists of $s = (i, j, l_0, k_s, k_d)$, where i and j are the indices of connected particles, l_0 is the rest length, k_s is the spring stiffness and k_d is the damping coefficient. To calcuate the forces acting on particles i and j, we use the following formula:

$$f_i^S = f^S(x_i, x_j) = k_s \frac{x_j - x_i}{|x_j - x_i|}(|x_j - x_i| - l_0) \tag{1.1}$$

$$f_j^S = f^S(x_j, x_i) = -f^S(x_i, x_j) \tag{1.2}$$

It is easy to see that the forces conserve momentum, since $(f_i + f_j = 0)$

We apply damping by computing the damping forces:

$$f_i^D = f^D(x_i, v_i, x_j, v_j) = k_d(v_j - v_i)\frac{x_j - x_i}{|x_j - x_i|} \tag{1.3}$$

$$f_j^D = f^D(x_j, v_j, x_i, v_i) = -f_i^D \tag{1.4}$$

Combining two forces we get the final spring force

$$f(x_i, v_i, x_j, v_j) = f^S(x_i, x_j) + f^D(x_i, v_i, x_j, v_j) \tag{1.5}$$

Simulation

In order to simulate the mass spring system, we use the Newton's second law of motion,

$$f = m\ddot{x} \tag{1.6}$$

where f is the force, m is the mass and $\ddot{x}$ is the acceleration or the second derivate of position with respect to time. By transforming the equation to solve for acceleration, we get a second order ordinary differential equation(ODE):

$$\ddot{x} = \frac{f}{m} \tag{1.7}$$

In order to solve it, we can split this equation into two first order ODEs

$$\dot{v} = \frac{f}{m} \tag{1.8}$$

$$\dot{x} = v \tag{1.9}$$

Analytically, these can be solved by definite integrals:

$$v(t) = v_0 \int_{t_0}^{t} \frac{f(t)}{m} dt \tag{1.10}$$

$$x(t) = x_0 \int_{t_0}^{t} v(t) dt \tag{1.11}$$

where $v_0 = v(t_0)$ and $x_0 = x(t_0)$ are the initial conditions.

Numerical Integration

Explicit Euler Integration

One of the most basic numerical integration of ODE is explicit Euler integration scheme. The scheme approximates the derivatives using finite differences:

$$\dot{v} = \frac{v^{t+1} - v^t}{\Delta t} \tag{1.12}$$

$$\dot{x} = \frac{x^{t+1} - x^t}{\Delta t} \tag{1.13}$$

where Δt is a discrete time step and t is the index of the simulation iteration. By substituting these equations into Eq. 1.8 and Eq. 1.9, we get get the explicit Euler integration method:

$$v^{t+1} = v^t + \Delta t \frac{f(x^t, v^t)}{m} \tag{1.14}$$

$$x^{t+1} = x^t + \Delta t v^{t+1} \tag{1.15}$$

The term 'explicit' comes from the fact that information of the next time step can be directly computed using the information at the current time step.

The entire simulation can be summed with the following algorithm:

f^g is the gravity force and f^{coll} collision forces.

Algorithm 1 Mass Spring Simulation

```
1: procedure Simulation
2:     while true do
3:         for all particles $i$ do
4:             $f_i = f^g + f_i^{coll} + \sum_{j,(i,j) \in S} f(x_i, v_i, x_j, v_j)$
5:         for all particles $i$ do
6:             $v_i = v_i + \Delta t \frac{f_i}{m_i}$
7:             $x_i = x_i + \Delta t v_i$
```

A known drawback for explicit Euler integration is the fact that it requires small time steps to remain stable. This problem accurs becouse explicit Euler does not account for the near future and it assumes that the force is constant during the entire time step. Let us assume a system of two particles connected with a spring. Assume the following configuration: the spring is stretched and the two particles start moving towards each other. If we take a large time step to compute the next configuration, the particles might pass the equilibrium configuration, which in theory means that the force should change its sign during that time step. Sadly, since the force is constant throughtout the entire time step, the sign change of the force is not accounted for. This might lead to particles overshooting and gaining energy which in turn leads to a so called simulation explosion. Other numerical integration methods exist that are more accurate. Among the most popular are the second and fourth order Runge-Kutta integrators. These schemes compute forces multiple times during a single time step, which might reduce the effect of the problem mentioned above.

Runge-Kutta Integration

The second order Runge-Kutta integrator has a different method of numerically solving ODEs. The approximation of explicit Euler Eq. 1.14 and Eq. 1.15 are instead computed by the formulas:

$$\begin{aligned} a_1 &= v^t \\ a_2 &= \frac{f(x^t, v^t)}{m} \\ b_1 &= v^t + \frac{\Delta t}{2} a_2 \\ b_2 &= \frac{f(x^t + \frac{\Delta t}{2} a_1, v^t + \frac{\Delta t}{2} a_2)}{m} \\ x^{t+1} &= x^t + \Delta t b_1 \\ v^{t+1} &= v^t + \Delta t b_2 \end{aligned} \tag{1.16}$$

It is easy to see that the forces are computed twice during one time step. This makes the secord order Runge-Kutta integrator more accurate compared to the simple first order explicit Euler method.

One of the most popular methods of integrating ODE is a forth order Runge-Kutta integrator. It extends the the second order Runge-Kutta by computing the force four times during a single time step. Making it even more accurate. The accuracy obviously comes with longer computations.

Implicit Euler

Another way to improve stability is to use implicit integrator. Among the most populars is the implicit Euler method. As opposed to explicit integrators, the implicit is more physically correct.

$$v^{t+1} = v^t + \Delta t \frac{f(x^{t+1})}{m} \tag{1.17}$$

$$x^{t+1} = x^t + \Delta t v^{t+1} \tag{1.18}$$

First difference lies in the force function f. Now it only depends on the position. In another words, the force does not include friction. It is said that implicit integration introduces enough numerical damping to accomedate for physical damping(TODO citation). However, if needed, the friction force can be added in the explicit step after the implicit solve. The most important change, is the fact that the force f depends now on the position of next step x^{t+1}. Thus, it is no longer possible to explicitly compute the two equations. Instead we now deal with a algebraic system, with unknowns being x^{t+1} and v^{t+1}.

In order to compute these equation we must first construct the algebraic system. The position, velocites and forces are concatanted into vectors:

$$\begin{aligned} x &= [x_1, x_2, ..., x_n] \\ v &= [v_1, v_2, ..., v_n] \\ f(x) &= [f_1(x_1, ..., x_n), ..., f_n(x_1, ..., x_n)] \end{aligned} \tag{1.19}$$

Further, we construct a mass matrix $M \in \mathbb{R}^{3N \times 3N}$ which is diagonal with values $m_1, m_1, m_1, ..., m_N, m_N, m_N$ on the diagonal.

$$Mv^{t+1} = Mv^t + \Delta t f(x_{t+1}) \tag{1.20}$$

$$x^{t+1} = x + \Delta t v^{t+1} \tag{1.21}$$

Substituing Eq. 1.21 into Eq. 1.20, results in single system of algebraic equations:

$$Mv^{t+1} = Mv^t + \Delta t f(x + \Delta t v^{t+1}) \tag{1.22}$$

We solve this system for v^{t+1}.

Examples

Example implementation of mass spring system can found in open source software created by the author of this thesis in [2]. TODO Show screen TODO Give profesional example

Conclusions

Mass spring systems are easy to implement and for many applications give good enough results(e.g. computer games). However, relatively expensive ODE integrators have to be used in order to keep the simulation stable. Moreover, modeling physically correct materials can be a complicated task, since the parameters of the system hardly reflect reality.

Position Based Dynamics

As the name suggests, Position Based Dynamics(PBD) omits integrating over velocity and works directly on positions. The biggest advantage over mass spring system is avoidance of overshooting problem during integration step.

Formulation

The system of PBD includes a set of N particles and a set of M contraints. Each particle i has three attributes:

1. mass m_i
2. position x_i
3. velocity v_i

Each constraint j has five attributes:

1. Cardinality - n_j
2. Scalar constraint function - $C_j : \mathbb{R}^{3n_j} \to \mathbb{R}$
3. Set of indices - $\{i_1, ..., i_{n_j}\}, i_k \in [1, ..., N]$

4. Stiffness parametr - $k_j \in [0...1]$
5. Type - *unilateral* or *bilateral*

It is said that bilateral contraint j is satisfied if $C_j(x_{i_1}, ..., x_{i_{n_j}}) = 0$ or if the case of unilateral $C_j(x_{i_1}, ..., x_{i_{n_j}}) \geqslant 0$. The strength of the contraint is defined by the stiffness parameter k_j.

Given initial conditions for positions and velocities the simulation proceeds as follows:

Algorithm 2 Particle Based Dynamics

```
1: procedure Simulation
2:     while true do
3:         for all particles $i$ do
4:             $v_i = v_i + \Delta t \frac{f_i}{m_i}$
5:             $p_i = x_i + \Delta t v_i$
6:             $generateCollisionContraints(x_i, p_i)$
7:         while iteratively do
8:             $projectContraints(C_1, ..., C_{M+M_{ext}}, p_1, ..., p_N)$
9:         for all particles $i$ do
10:            $v_i = \frac{(p_i - x_i)}{\Delta t}$
11:            $x_i = p_i$
```

The lines 4 and 5 compute explicit Euler integration on velocities and positions. However, output positions p_i are only used as predictions. The line 6 generates external constrain such as collisions. The original and predicted positions x_i, p_i can be used in this step in order to perform continuous collision detection. The simulation then computes line 8 which iteratively corrects the predicted positions such that they satisfy the M_{ext} external and M internal constraints.

Finite Element Method

Computational Mesh

Creating Compulational Mesh

Finite Element Method

Advanced

Formulation

Some general math from my Presentation and

Strong formulation

Weak formulation

Discrete formulation

FEM - Idea

Final formulation

Finite Element - Tetrahedron

Material

Load

Assembly Process

Sparse Matrix Representation

Boundary Conditions

Solver

Implementation

CUDA

GPU Architecture

Pascal Architecture

Matrix Vector Multipliation

Linear System of Equations

CG

CG with precondition

LU

RTFEM

RTFEM Integration into Game Engine

Rendering

Collision

Tests

Benchmarks

Float vs Double

Materials

Structures

Speed

Conclusions

Bibliografia

[1] Matthias Muller, Jos Stam, Doug James, Nils Thurey, *Real Time Physics Class Notes.*

[2] Jakub Ciecierski, *https://github.com/Jakub-Ciecierski/SoftBodySimulation*

[3] Michał Kleiber, Piotr Kowalczyk, *Wprowadzenie do Nieliniowej Termomechaniki Ciał Odkształcalnych.*

[4] Zienkiewicz *The Finite Element Method. Volume 1: The Basis.*

[5] Zienkiewicz *The Finite Element Method. Volume 2: Solid Mechanics.*

[5] Altair University *Practical Aspects of Finite Element Simulation.*

Wykaz symboli i skrótów

nzw.	nadzwyczajny
*	operator gwiazdka
~	tylda

Spis rysunków

Spis tabel

Spis załączników

Załącznik 1, załącznik 2 – mają się znajdować na końcu pracy (to jest notka przypominająca)