Workshop Series: Jello Shape Simulation

Benjamin Kenwright[1]*

Abstract
This practical focuses on simple particle constraint mechanics (e.g., Newton's laws, Hooke's law, distance constraints, collision detection). The student needs to implement a real-time interactive 'jello shape' using Newtonian mechanics (e.g., forces and penalty constraints). When the 'jello shape' is stretched, it will try to contract, when squeezed together, it will tend to inflate back to the original form. As with the previous practicals, you will also create an interesting animation demonstrating your results.

Keywords
Newtonian Mechanics, Hooke's Law, Springs, Particles, Constraints, Soft-Bodies, Classical Mechanics

[1] *Workshop Series (www.xbdev.net) - Benjamin Kenwright*

Contents

Introduction

Particle Constraints The topic of this practical is the implementation of a real-time particle constraint simulation (i.e., an interconnected set of particle bodies). The user should be able to interact with the simulation while it is running (e.g., through the mouse or keyboard) to control the 'jello shape' - such as push or pull the object around. The 'jello shape' should be implemented using classical mechanic principles (e.g., Euler integration, Hooke's law for distance constraint, and sphere-plane collision detection for the ground). The deformable shape will stretch, contract, oscillate, change velocity, bounce off walls using the physical laws for a mass-spring system.

Tasks

1. Visually display an interconnected 'jello shape'
2. Interconnect the point-mass 'jello shape' using a spring-damper structure
3. User input (e.g., mouse or keyboard) to control and move the 'jello shape' around
4. 'Jello shape' should be under the influence of gravity and come to rest on the ground (i.e., sphere-plane collision detection)

1. Overview

Principles and Concepts Remember that, at this point, a particle physics system is still dealing with forces and masses - the challenge is implementing a system which is stable and computationally efficient enough to run at interactive frame rates.

3D Chunk of Jello You will model a 3D chunk of jello, which, when un-deformed, has the shape of a cube of a specific dimension (e.g., 1 meter x 1 meter x 1 meter). The cube will stretch, contract, oscillate, change velocity, bounce off the ground and walls of the virtual environment, based on the physical laws for a *mass-spring system*. The jello shape will stay inside a viewable region (e.g., constrain the jello shape to move within a bounding box). When the jello shape hits a wall or the ground, it should bounce off and move backwards.

You will model the movement of the jello shape by numerically solving a system of ordinary differential equations, which sounds perhaps a bit scary, but is actually not that difficult. The equations to be solved incorporate Newton's second law ($\vec{f} = m\vec{a}$), Hook's linear model of elasticity ($\vec{f} = k\vec{x}$), and linear damping ($\vec{f} = -k\vec{v}$).

2. Mass-Spring System

We begin by reviewing the necessary principles and theory. A mass-spring system is the connection of several point-masses. Each point-mass is connected to each other by springs. Springs expand and stretch, exerting force on the connected point-masses. A common simulation example exploiting springs is cloths and soft-bodies.

Newton's Law At the heart of the mass-spring system is Newton's 2nd law as given below in Equation 1:

$$\vec{F} = m\vec{a} \tag{1}$$

where $\vec{F}$ is the net force, m is the mass, and $\vec{a}$ is the acceleration. Newton's second law tells us how to compute acceleration given the object's force and mass, while Newton's 3rd law tells us the reaction force - i.e., if object A exerts a force $\vec{F}$ on object B, then object B is at the same time exerting force $-\vec{F}$ on A.

Single Spring & Hooke's law Assume $\vec{A}$ and $\vec{B}$ are two mass points connected with a spring. The elastic force exerted on A is given below in Equation 2:

$$\begin{aligned}\vec{F}_{Hooke} &= -k_s(|\vec{L}| - R)\frac{\vec{L}}{|\vec{L}|} \\ &= -k_s(|\vec{L}| - R)\,\hat{L}\end{aligned} \tag{2}$$

where L be the vector pointing from B to A, R is the spring rest length, and k_s is the spring elasticity (aka stiffness). We can deduce when $|\vec{L}| < R$ the spring will want to extend, while when $|\vec{L}| > R$ the spring will want to contract.

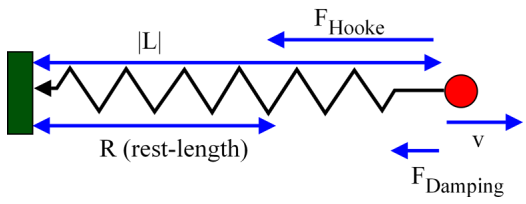


Figure 1. Spring Illustration - Connected particle parameters.

Damping In reality, springs are not completely elastic and absorb energy and tend to decrease the velocity of the mass points attached to them. Damping force depends on the velocity and are defined by Equation 3 below:

$$\vec{F}_{Damping} = -k_d\vec{v} \tag{3}$$

where k_d is the damping coefficient.

For an uncomplicated 3D spring - if we assume A and B two connected mass points the damping force exerted on A is given by Equation 4:

$$\begin{aligned}\vec{F}_{Damping} &= -k_d\frac{(\vec{v}_A - \vec{v}_B)\bullet L}{|\vec{L}|}\frac{\vec{L}}{|\vec{L}|} \\ &= -k_d((\vec{v}_A - \vec{v}_B)\bullet\hat{L})\,\hat{L}\end{aligned} \tag{4}$$

where L is the vector pointing from B to A, v_A and v_B are velocities of points A and B. It's important to remember, a damping force is always opposes the direction of motion (i.e., negative sign).

Network of Springs For a cloth spring-damper simulation (see Figure 2) every mass point is connected to some other points by springs (i.e., each spring represents a distance constraint using Hooke's force and damping). We can also include additional external forces, such as, force fields, gravity, and collision forces to create life-like scenes.

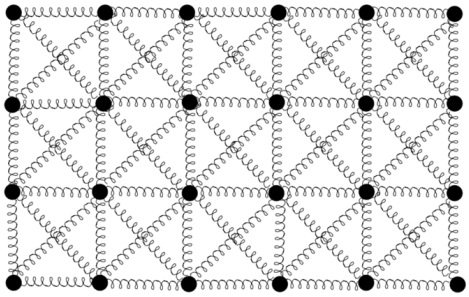
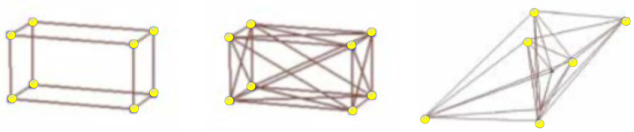


Figure 2. Cloth Simulation Structure - The concept in this practical can also be applied to an interconnected system of particles to representing cloths.

3. How to Organize the Network (Jello Shape)

Example: Jello Cube To understand how to configure your spring-damper system - consider a simple example, i.e., a 'jello cube' system. The particle constraints (i.e., springs) for the 'jello cube' need to be connected is a specific topology to create a stable network which we discuss in this section. Note, the initial 'jello cube' will have the following properties:

- 'Jello cube' is a 8x8x8 mass point network
- 512 discrete points
- Connect the point-masses with springs (i.e., a clever and coordinated topology)



(a) Basic Network (b) Stable Network (c) Out of Control Network

Figure 3. Jello Network - Connecting the jello particles together.

Structural, Shear and Bend Springs For the 'jello cube' to remain stable and in shape (i.e., return to a cube like pose) - you need to incorporate spring types:

- Structural
- Shear
- Bend

Structural Springs Connect every point-mass node to its 'six' direct neighbours. For example, node (i,j,k) connected to: (i+1,j,k), (i-1,j,k), (i,j-1,k), (i,j+1,k), (i,j,k-1), (i,j,k+1). For the surface nodes some of these neighbors might not exist. Structural springs establish the basic structure of the 'jello cube'.

Shear Springs Shear springs disallow excessive shearing and prevent the cube from distorting. Every node (i,j,k) connected to its diagonal neighbors as shown in Figure 6.

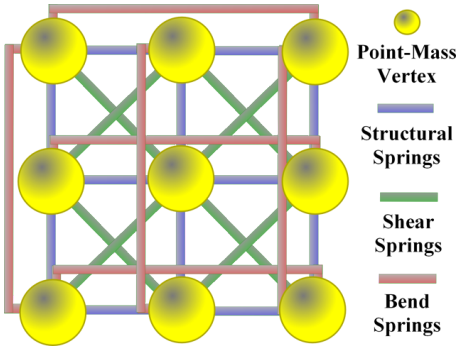


Figure 4. Spring Structures - Element interconnected structure configurations.

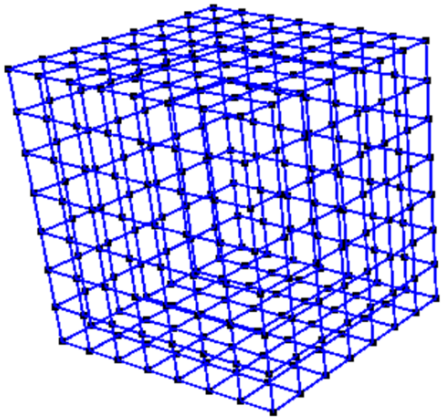


Figure 5. Structural Springs for the 'jello cube' - Only the springs connecting two surface vertices are shown in the rendering.

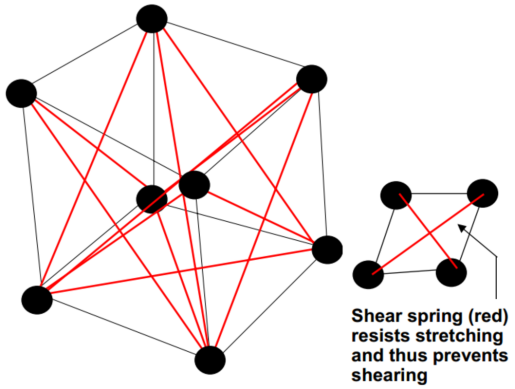


Figure 6. Shear Springs - You should be able to make out the eight point-mass elements and the interconnected circuit for the 3D cube. (Structural springs are black and shear springs are in red).

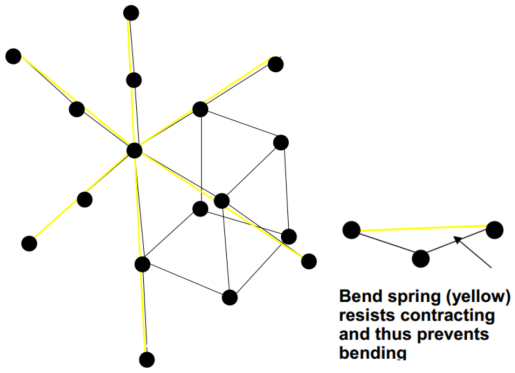


Figure 7. Bend Springs - Where the black lines represent structural springs and yellow lines bend springs (shown for a single node only).

Bend Springs Bend springs prevent the cube from folding over. Every node connected to its second neighbor in every direction (6 connections per node, unless surface node) as shown in Figure 7.

Collision detection The movement of the 'jello cube' is limited - for example, colliding with the terrain (i.e., a plane for the ground and walls). For example, confining the 'jello cube' to a bounding box - the collision detection implemented would check all the vertices to see if they are inside or outside the box. For a plane, we recall the plane equation (i.e., $P(x,y,z) = \vec{n} \bullet \vec{b} = 0$), with all points being initially on the same side of the plane. When $P(x,y,z) > 0$ on one side of the plane and $P(x,y,z) < 0$ on the other. Check all the vertices for this condition.

Collision Response When collisions happen we must perform some action to prevent the object penetrating even deeper. The 'jello cube' should bounce away from the colliding object. Some energy is usually lost during the collision. While there are several ways of handling collision response, we use the penalty-based method. For the penalty-based method, when a collision happens, we put an artificial collision spring at the point of contact, which pushes the object backwards and away from the colliding object. Collision springs have elasticity and damping just like ordinary springs.

Integration Network of point-mass springs using the popular Hooke's law with damping. Through Newton's 2nd law we are able to calculate the acceleration of every point-mass at any given time (i.e., $\vec{F} = m\vec{a}$). We know the mass and acceleration of the system of particles and the 'integration' step is used to compute the 'actual motion'.

$$\begin{aligned} \frac{d\vec{x}}{dt} &= \vec{v} \\ \frac{d^2\vec{x}}{dt^2} &= \frac{d\vec{x}}{dt} \\ &= \frac{1}{m}(\vec{F}_{Hooke} + \vec{F}_{damping} + \vec{F}_{external}) \end{aligned} \tag{5}$$

where x is the point-mass position, v is the point-mass velocity, and $\vec{F}$ is the applied force. Equation 5 describe the movement of a single point-mass. With $\vec{F}_{hook}$ representing sum of all

Hooke forces on a point-mass and $\vec{F}_{damping}$ is the sum of all damping forces on a point-mass. When we put these equations together for all the mass points, we obtain a system of ordinary differential equations (ODEs). In general, impossible to solve analytically and must solve numerically. Methods to solve such systems numerically are called integrators. The most widely used integrators are: Euler, Runge-Kutta 2nd order (aka the midpoint method) and Runge-Kutta 4th order (RK4).

4. Engineering Design Tips

- Numerical stability issues

  For large time-steps the system will 'explode' (e.g., t = 0.001 is a good starting choice for the practical)

  Euler is much more unstable compared to RK2 or RK4, but requires smaller time-steps while being simpler and faster

  Euler is rarely used in practice (i.e., typically RK4)

  Smaller time-steps means more stability and accuracy (but also means more computation)
- The time-step should be inversely proportional to the square root of the elasticity k [Courant condition]
- Computational cost (i.e., trade-off: accuracy vs computation-time)
- Be aware of floating point and double precision calculation errors
- Choosing the right elasticity and damping parameters is an art (i.e., trial and error). Initially, set the ordinary and collision parameters the same to create a stable simulation

5. Summary

This practical introduced the fundamental Newtonian mechanics of a spring-damper network for constructing soft body structures. The soft body structures (i.e., a 'jello shape') was implemented using distance constraints through a penalty-based spring-damper topology. We resolve the multiple forces affecting an interconnected set of point-mass particles and discussed the mathematical relationship between force and acceleration. In the next practical, we will discuss how to implement a modified particle constraint system using a position-based integrator for an aesthetically pleasing and robust simulation result. Your practical work should be structured in a efficient, well commented format, such that the physics simulation is separate from the renderer update, and the physics implementation which you develop can easily expanded and interfaced with your other projects (i.e., modular component programming).

6. Exercises

This practical only gives a brief introduction to mass-spring systems. As an exercise for the student to help enhance their understanding:

Intermediate

- Implement a scene where your 'jello shape' interacts with the environment (e.g., colliding with spheres and bouncing around)
- Implement your 'jello shape' so the spring-damper constrain colours are drawn based on the penalty force (e.g., gradient blue to red with blue for no forces and red for maximum force)
- Implement your 'jello shape' with different spring-damper constraints for regions of the shape (i.e., all the spring-damper coefficients aren't all the same)
- Implement your 'jello shape' so the spring-damper coefficients change over time (e.g., gradually reduce to cause a melting effect)
- Explore and engineer speed-up techniques by utilizing the power of the GPU with CUDA or OpenCL

Advanced

- Implement a variety of different and unique 'jello shape' - e.g., loading surface meshes from external files to create jello shapes for rabbits and tea-pots
- Implement multiple interacting 'jello' shapes in a scene

Acknowledgements

We would like to thank all the students for taking time out of their busy schedules to provide valuable and constructive feedback to make this practical more concise, informative, and correct. However, we would be pleased to hear your views on the following:

- Is the practical clear to follow?
- Are the examples and tasks achievable?
- Do you understand the objects?
- Did we missed anything?
- Any surprises?

The practicals provide a basic introduction for getting started with physics-based animation effects. So if you can provide any advice, tips, or hints during from your own exploration of simulation development, that you think would be indispensable for a student's learning and understanding, please don't hesitate to contact us so that we can make amendments and incorporate them into future practicals.

Recommended Reading

Physics for Game Developers, David M Bourg, Publisher: O'Reilly Media, ISBN: 978-1449392512

Physics-based Animation, Kenny Erleben, Jon Sporring , Knud Henriksen, Publisher: Charles River Media, ISBN: 978-1584503804

Computer Animation: Algorithms & Techniques, Rick Parent, Publisher: Morgan Kaufmann, ISBN: 978-0124158429

Code Complete: A Practical Handbook of Software Construction, Steve McConnell, ISBN: 978-0735619678

Clean Code: A Handbook of Agile Software Craftsmanship, Robert C. Martin, ISBN: 978-0132350884

Game Inverse Kinematics: A Practical Introduction (2nd Edition) Kenwright. ISBN: 979-8670628204

Kinematics and Dynamics Paperback. Kenwright. ISBN: 978-1539595496

Game Collision Detection: A Practical Introduction (Paperback). Kenwright. ISBN: 978-1511964104

Game C++ Programming: A Practical Introduction (Paperback). Kenwright. ISBN: 978-1516838165

Computational Game Dynamics: Principles and Practice (Paperback). Kenwright. ISBN: 978-1501018398

Game Physics: A Practical Introduction (Paperback). Kenwright. ISBN: 978-1471033971

Game Animation Techniques: A Practical Introduction (Paperback). Kenwright. ISBN: 978-1523210688