Physics for Game Development

Robert Bruce

CS-330: Introduction to Game Programming

Components in a physics game engine

A generic physics game engine contains the following components:

- Physics models
- Simulated objects manager
- Collision detection engine
- Collision response module
- Force effectors
- Numerical integrator
- Game engine interface

Source: *Physics for Game Developers*, pp. 283

Vector mathematics and classical mechanics in Physics.

Before implementing a physics engine, it is vital we review the following:

- Vector mathematics:
  - We will be computing forces on objects using vector mathematics.
- Classical mechanics in physics.
  - Cloth and rope simulation is possible using Newton's second law (F = m * a) and Hooke's law (F = k * x).

Vector mathematics

A vector contains two components:

- A scaler value
- A direction

Example: vector $v = A\hat{i} + B\hat{j} + C\hat{k}$

Normalizing a vector

vector $v = A\hat{i} + B\hat{j} + C\hat{k}$

- Normalizing a vector involves dividing the vector by its magnitude. In doing so, the result is a unit vector with magnitude one.
- Normalizing a vector is important when one cares only about direction (e.g. direction of a force) rather than direction and magnitude.
- Normalized vector $v = v / ||v|| = v / \mathbf{SQRT}(A^2 + B^2 + C^2)$

Computing vector dot product

vector $v_1 = A\hat{i} + B\hat{j} + C\hat{k}$

vector $v_2 = D\hat{i} + E\hat{j} + F\hat{k}$

The dot product for two vectors, $v_1 \bullet v_2 = ||v1|| * ||v2|| * \cos(\theta)$

Note: theta (θ) represents the angle between vectors v_1 and v_2.

The dot product generates a scalar result, not a vector.

Computing vector cross product

vector $v_1 = A\hat{i} + B\hat{j} + C\hat{k}$

vector $v_2 = D\hat{i} + E\hat{j} + F\hat{k}$

The cross product for two vectors, $v_1 \times v_2 = ||v1|| * ||v2|| * \sin(\theta)\hat{a}$

Note: theta (θ) represents the angle between vectors v_1 and v_2.

The cross product of v_1 and v_2 is a vector ($\hat{a}$) that is orthogonal to both v_1 and v_2.

Right hand rule

Right hand rule:

$\hat{i} \times \hat{j} = \hat{k}$

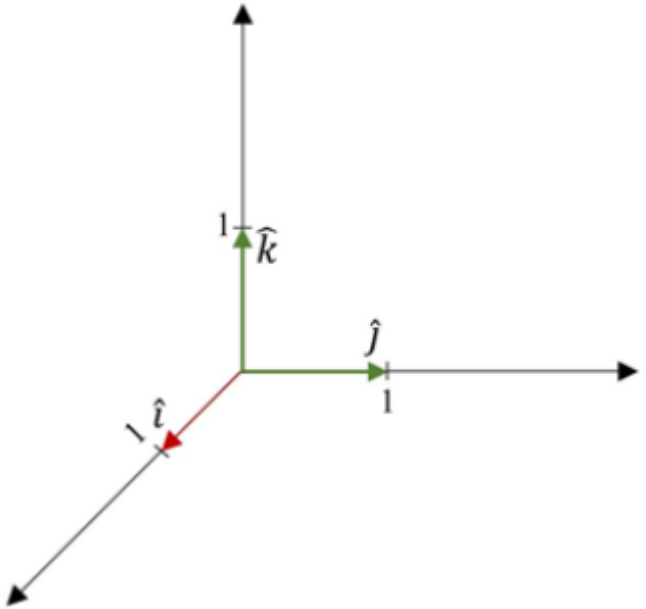
$\hat{j} \times \hat{k} = \hat{i}$

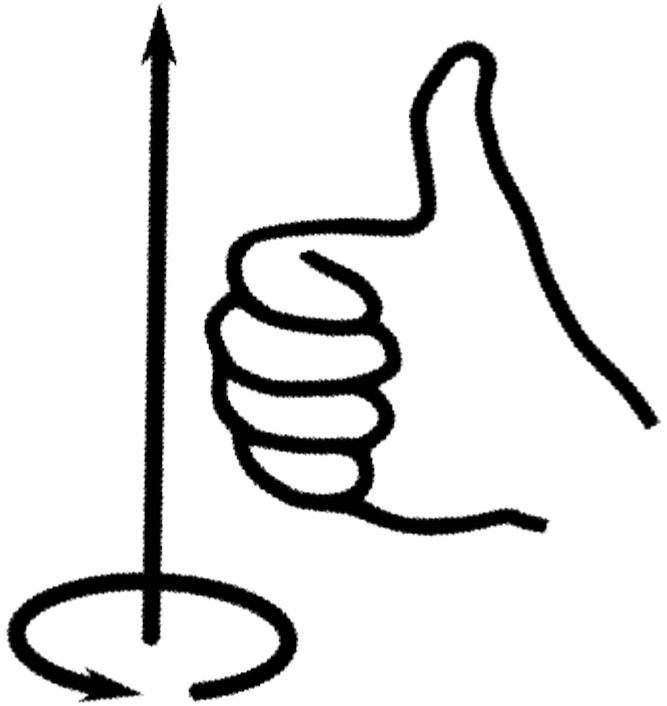
$\hat{k} \times \hat{i} = \hat{j}$

$\hat{i} \times \hat{k} = -\hat{j}$

$\hat{j} \times \hat{i} = -\hat{k}$

$\hat{k} \times \hat{j} = -\hat{i}$





Types of forces

- Gravitational force
- Static and dynamic friction forces
- Fluid dynamics drag force
- Buoyancy force
- Spring and damper forces

Source: *Physics for Game Developers*, pp. 71-80

Gravitational force

Newton's law of gravitation:

$F_a = G \times m_1 \times m_2 / r^2$

G is the gravitational constant = 6.673 × 10-11 N-m^2/kg^2

F_a is gravitational force of attraction between the two bodies.

m_1 and **m_2** are the masses of the two bodies.

r is the distance between the center of mass of m_1 and m_2.

Source: *Physics for Game Developers*, pp. 72

Static force

Static force:

$F_s = \mu_s \times N$

F_s is the static force exerted on a body.

μ_s is the coefficient of static friction (a constant).

N is the normal force between a body and surface.

Source: *Physics for Game Developers*, pp. 74

Dynamic force

Dynamic force:

$F_d = \mu_d \times N$

F_d is the dynamic force exerted on a body.

μ_d is the coefficient of dynamic friction (a constant).

N is the normal force between a body and surface.

Source: *Physics for Game Developers*, pp. 74

Buoyancy force

“Buoyancy is a force that develops when an object is immersed in a fluid. It’s a function of the volume of the object and density of the fluid and results from the pressure differential between the fluid above the object and the fluid just below the object.”

For spheres, cubes, and cylinders calculating volume is easy. For other shapes, you may need to use techniques such as numerical integration.

Source: *Physics for Game Developers*, pp. 77-78

Spring force

$F_s = K_s \times (L - r)$

F_s is the spring force.

K_s is a spring constant.

L is the length of a spring in tension or compression.

r is the length of the spring at rest (not in tension or compression).

Source: *Physics for Game Developers*, pp. 79

Damper force

Damper forces act as a drag force.

$F_d = K_d \times (v_1 - v_2)$

F_d is the damping force.

K_d is a damping constant.

v_1 and **v_2** are the respective velocities of two connected objects.

Rob note: when computing simulations on cloth or rope, I recommend using damper forces in your simulations to absorb energy. The drag induced by the damper forces will reduce oscillation effects in cloth and rope simulations.

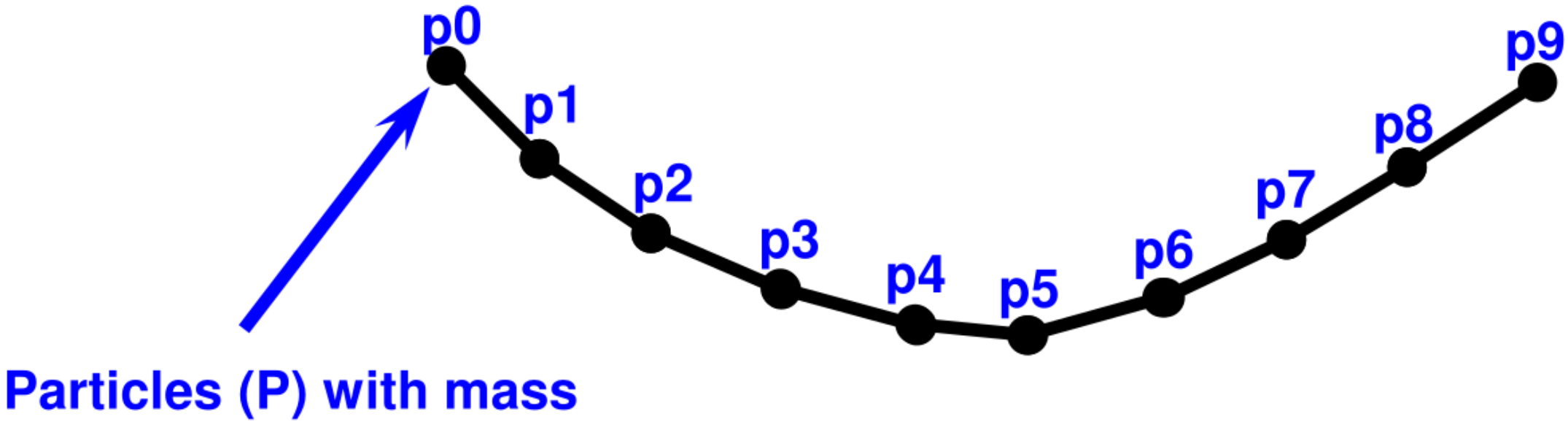
Source: *Physics for Game Developers*, pp. 79

Connecting objects

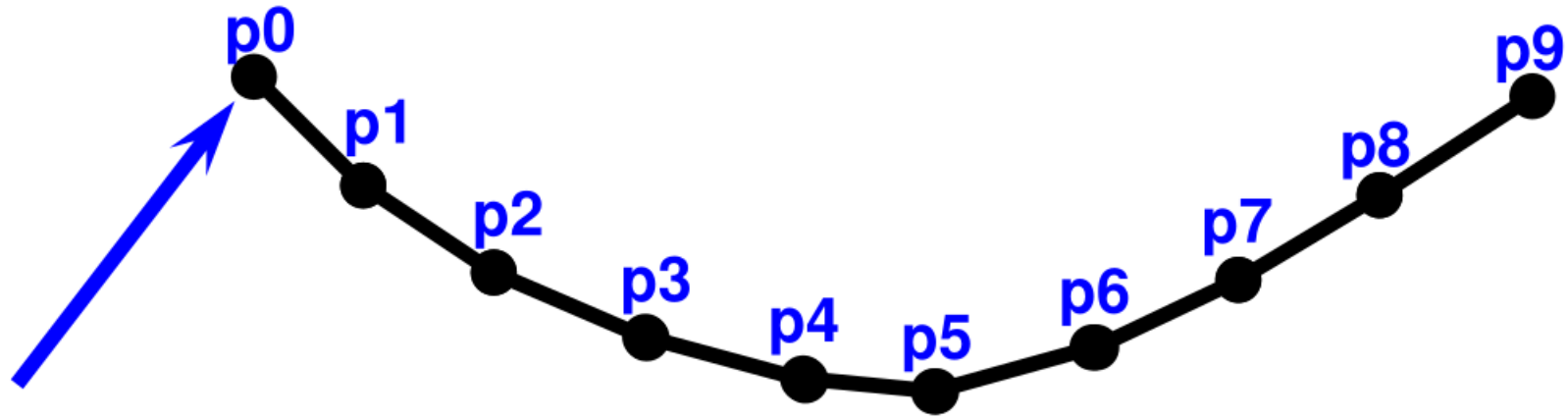
Connecting objects involves rigid body particles to simulate rope or cloth using the following:

- Hooke’s law, F=kx
- gravity
- rigid body particles (with mass)
- springs (no mass)

Connecting objects



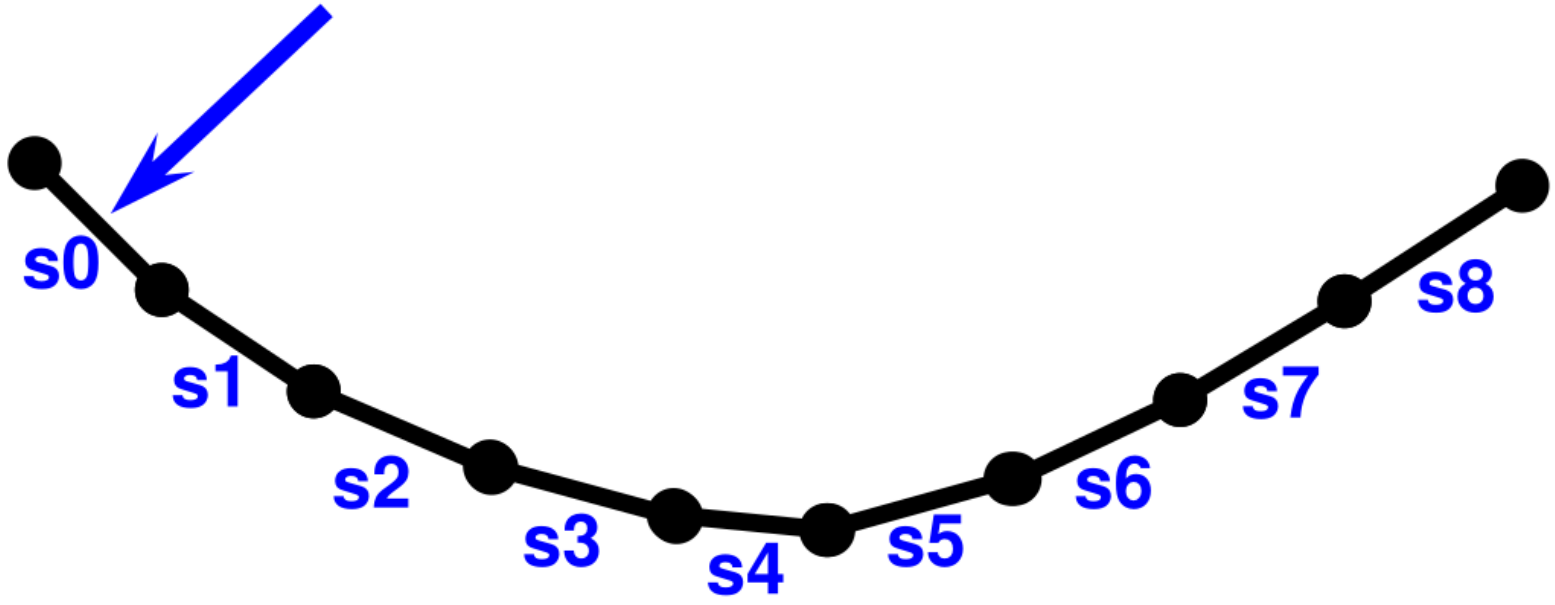
Connecting objects



Each particle is pinned to the next particle but allows rotation. This is called a “Pinned joint”.

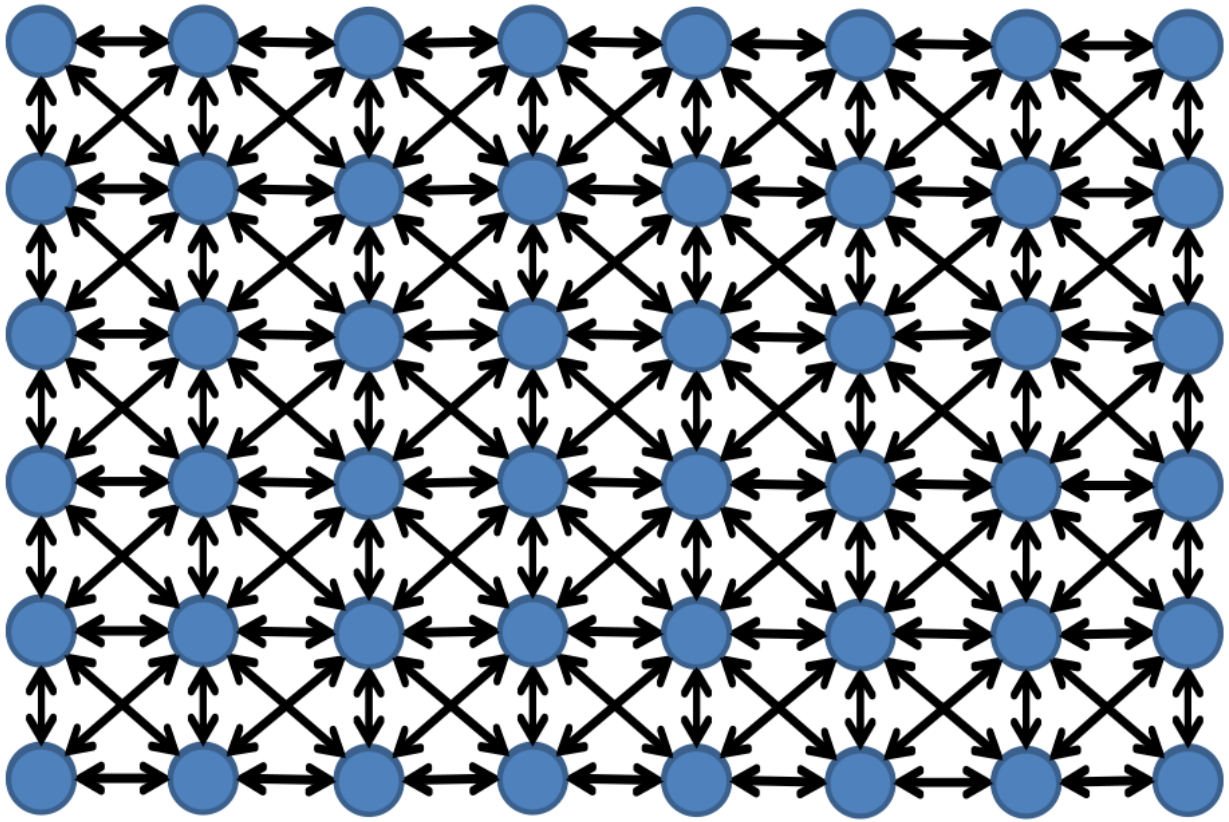
Connecting objects

Weightless, invisible springs (S) connect the particles together.



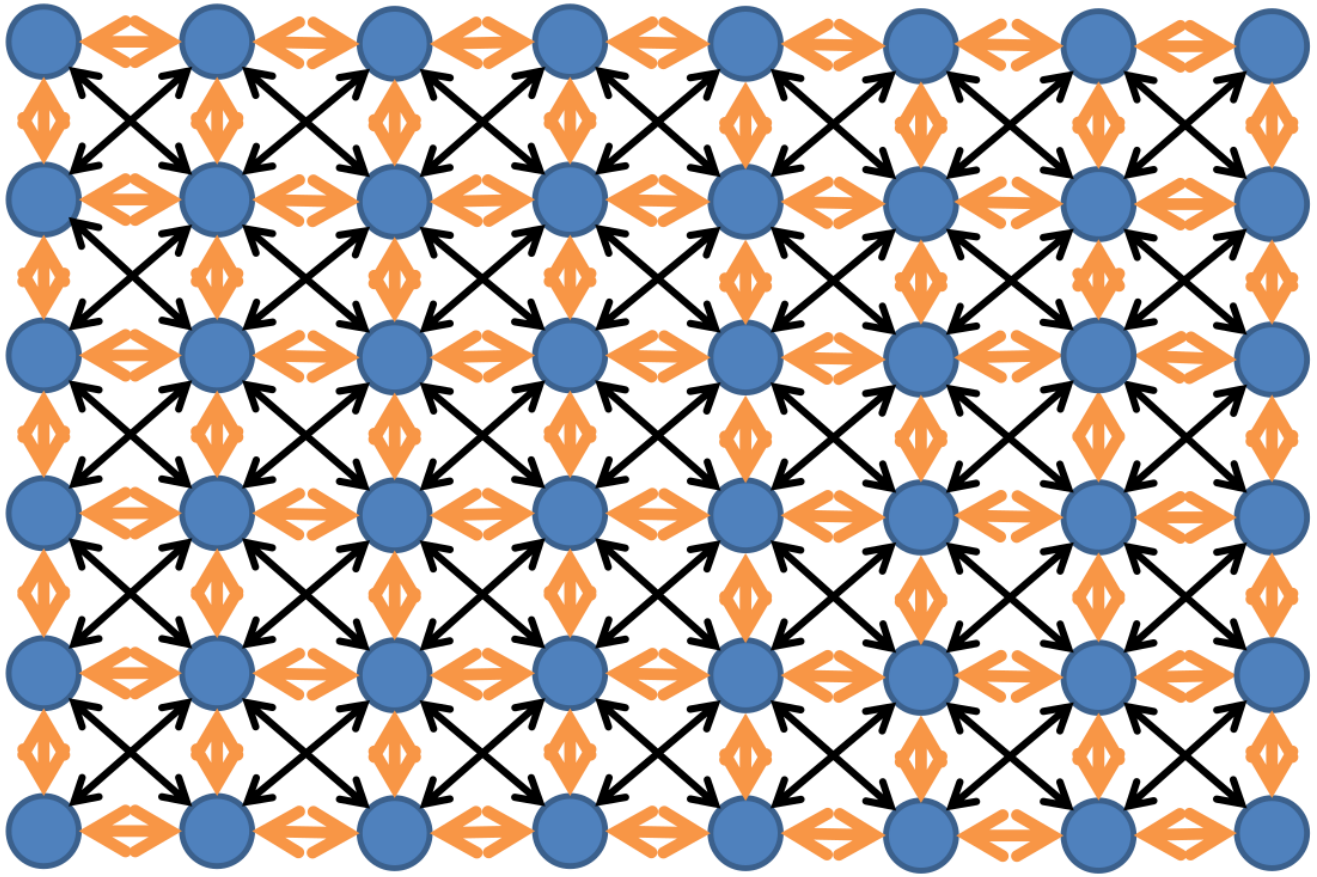
Mass spring model

In a mass spring model, cloth is comprised of an array of particles interconnected via weightless springs (denoted by the arrows):



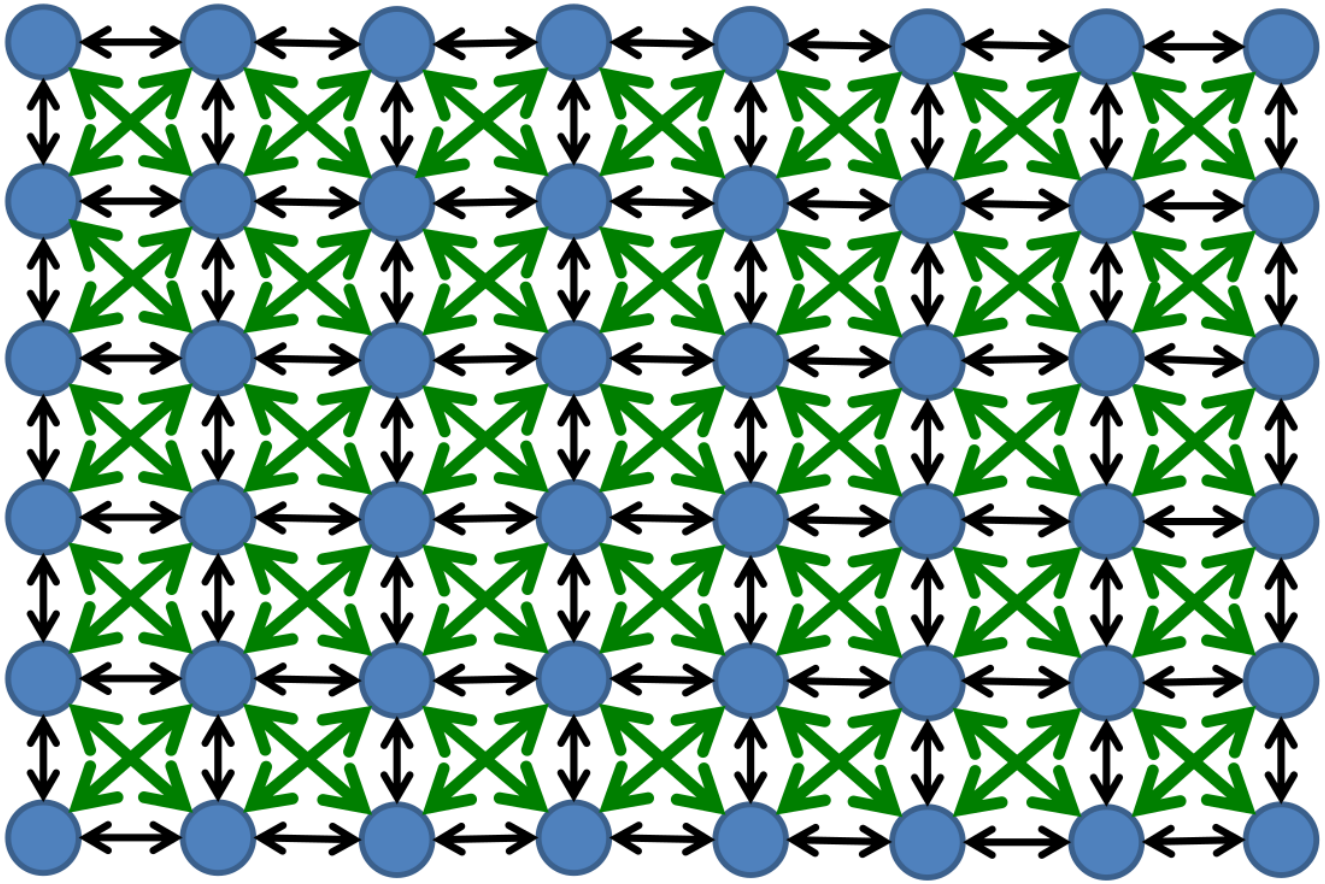
Mass spring model: anatomy

In the mass spring model, the orange-colored arrows represent structural springs:



Mass spring model: anatomy

In the mass spring model, the green-colored arrows represent shear springs:



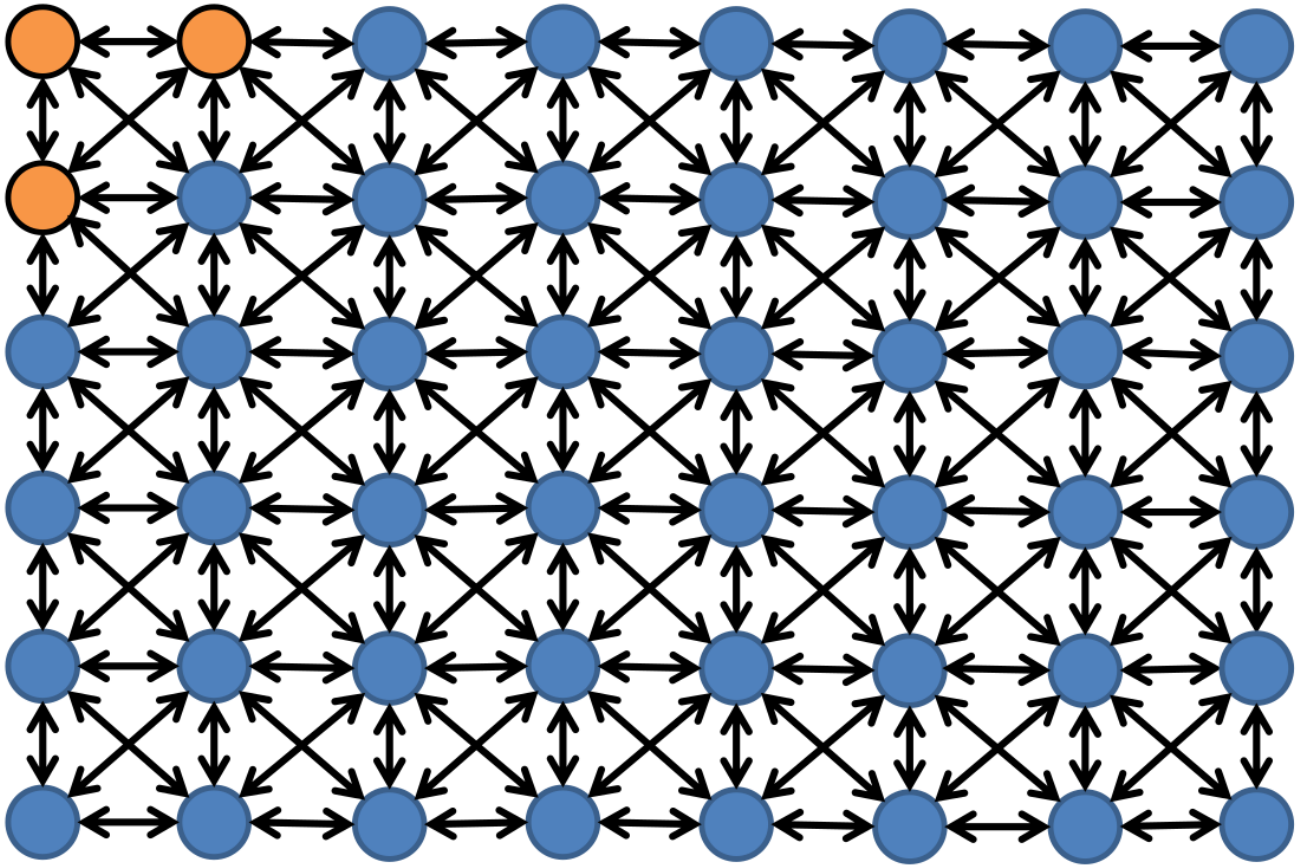
Computing surface normals

Why compute surface normals?

- To calculate wind forces pushing against the cloth.
- To calculate gravitational forces pulling the cloth.
- To calculate collision detection with solid objects.

Computing surface normals

Choose three adjacent particles that form a triangle:



Computing surface normals

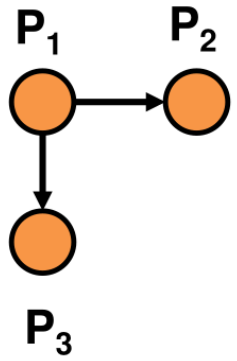
Compute the normal vector of the triangle defined by the position of the particles p1, p2, and p3:

Vector $v_1 = (P_2 - P_1)x + (P_2 - P_1)y + (P_2 - P_1)z$

Vector $v_2 = (P_3 - P_1)x + (P_3 - P_1)y + (P_3 - P_1)z$

cross product of v_1 and $v_2 = v_1 \times v_2$

Normalized $v_1 \times v_2 = (v_1 \times v_2) / (\text{magnitude of } v_1 \times v_2)$



Computing surface normals

Wind forces act on planes of the cloth

- A plane is formed from three particles in a triangular pattern.
- The normal of the plane is then used to compute how much wind forces act on the cloth at the particular plane.
- The wind force is a vector. It has values in
  - X direction
  - Y direction
  - Z direction
- Gravity is a force too but it has only has values in Y direction. Y is down.

Computing wind forces on the cloth

- The wind force is a force vector.
- To compute how much wind force moves the three particles forming a plane:
  - Compute the dot product between the wind force (a vector) and the normalized cross product of the three particles (a vector).
- Apply the dot product (computed above) as a force to each of the three particles (P1, P2, P3) that make up the plane. This will affect how each of the three particles is then displaced by the wind force.

Cloth optimization trick

For a stiffer cloth you could increase the “k” value of the spring or add a set of “flexion springs” between (i,j) and (i+2,j) and (i,j+2) and (i,j+2). Where (i,j) represent the row and column of each particle.

For further reading...

- *3D Math Primer for Graphics and Game Development* (2nd edition) by Fletcher Dunn and Ian Parberry
- *Advanced Character Physics* by Thomas Jakobsen.
- *Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior* by Xavier Provot.
- *Physics for game developers* (2nd edition) by David M. Bourg and Bryan Bywalec.
- *Real time physics class notes* by Matthias Müller, Jos Stam, Doug James, and Nils Thürey.