Some Potential/Force Fields for Particle Systems

D.A. Forsyth, largely lifted from Steve Rotenberg

Independent particles

- Force on particle does not depend on other particles
- Attractive for obvious reasons

- Simple gravity
- Drag
- Attractors/repellors

Uniform Gravity

- A very simple, useful force is the uniform gravity field:

$$\mathbf{f}_{gravity} = m\mathbf{g}_0$$

$$\mathbf{g}_0 = \begin{bmatrix} 0 & -9.8 & 0 \end{bmatrix} \quad \frac{m}{s^2}$$

- It assumes that we are near the surface of a planet with a huge enough mass that we can treat it as infinite
- As we don't apply any equal and opposite forces to anything, it appears that we are breaking Newton's third law, however we can assume that we are exchanging forces with the infinite mass, but having no relevant affect on it

Aerodynamic Drag

- Aerodynamic interactions are actually very complex and difficult to model accurately
- A reasonable simplification it to describe the total aerodynamic drag force on an object using:

$$\mathbf{f}_{aero} = \frac{1}{2}\rho|\mathbf{v}|^2 c_d a\mathbf{e} \qquad \mathbf{e} = -\frac{\mathbf{v}}{|\mathbf{v}|}$$

- Where ρ is the density of the air (or water…), c_d is the coefficient of drag for the object, a is the cross sectional area of the object, and **e** is a unit vector in the opposite direction of the velocity

Aerodynamic Drag

- Like gravity, the aerodynamic drag force appears to violate Newton’s Third Law, as we are applying a force to a particle but no equal and opposite force to anything else
- We can justify this by saying that the particle is actually applying a force onto the surrounding air, but we will assume that the resulting motion is just damped out by the viscosity of the air

Force Fields

- We can also define any arbitrary force field that we want. For example, we might choose a force field where the force is some function of the position within the field

$$\mathbf{f}_{field} \propto \mathbf{f}(\mathbf{r})$$

- We can also do things like defining the velocity of the air by some similar field equation and then using the aerodynamic drag force to compute a final force
- Using this approach, one can define useful turbulence fields, vortices, and other flow patterns

Dependent particles

- Force on particle DOES depend on other particles
  - computation is nasty
  - many improvements to simulation

- Gravity
- Spring Mass

Gravity

- If we are far away enough from the objects such that the inverse square law of gravity is noticeable, we can use Newton’s Law of Gravitation:

$$\mathbf{f}_{gravity} = \frac{G m_1 m_2}{d^2}\mathbf{e}$$

$$G = 6.673\times 10^{-11} \qquad \frac{m^3}{kg \times s^2}$$

Gravity

- The law describes an equal and opposite force exchanged between two bodies, where the force is proportional to the product of the two masses and inversely proportional to their distance squared. The force acts in a direction **e** along a line from one particle to the other (in an attractive direction)

$$\mathbf{f}_{gravity} = \frac{G m_1 m_2}{d^2}\mathbf{e} \qquad \mathbf{e} = \frac{\mathbf{r}_1 - \mathbf{r}_2}{\left|\mathbf{r}_1 - \mathbf{r}_2\right|}$$

Gravity

- The equation describes the gravitational force between two particles
- To compute the forces in a large system of particles, every pair must be considered
- This gives us an N^2 loop over the particles
- Actually, there are some tricks to speed this up, but we won’t look at those

Springs

- A simple spring force can be described as:

$$\mathbf{f}_{spring} = -k_s \mathbf{x}$$

- Where k is a ‘spring constant’ describing the stiffness of the spring and **x** is a vector describing the displacement

Springs

- In practice, it's nice to define a spring as connecting two particles and having some rest length *l* where the force is 0
- This gives us:

$$\mathbf{x} = x\mathbf{e}$$

$$x = \left|\mathbf{r}_1 - \mathbf{r}_2\right| - l \quad \text{(scalar displacement)}$$

$$\mathbf{e} = \frac{\mathbf{r}_1 - \mathbf{r}_2}{\left|\mathbf{r}_1 - \mathbf{r}_2\right|} \quad \text{(direction of displacement)}$$

Springs

- As springs apply equal and opposite forces to two particles, they should obey conservation of momentum
- As it happens, the springs will also conserve energy, as the kinetic energy of motion can be stored in the deformation energy of the spring and later restored
- In practice, our simple implementation of the particle system will guarantee conservation of momentum, due to the way we formulated it
- It will not, however guarantee the conservation of energy, and in practice, we might see a gradual increase or decrease in system energy over time
- A gradual decrease of energy implies that the system damps out and might eventually come to rest. A gradual increase, however, it not so nice… (more on this later)

Dampers

- We can also use damping forces between particles:

$$\mathbf{f}_{damp} = -k_d \mathbf{v}$$

- Dampers will oppose any difference in velocity between particles
- The damping forces are equal and opposite, so they conserve momentum, but they will remove energy from the system
- In real dampers, kinetic energy of motion is converted into complex fluid motion within the damper and then diffused into random molecular motion causing an increase in temperature. The kinetic energy is effectively lost.

Dampers

- To compute the damping force, we need to know the *closing* velocity of the two particles, or the speed at which they are approaching each other

$$\mathbf{e} = \frac{\mathbf{r}_1 - \mathbf{r}_2}{\left|\mathbf{r}_1 - \mathbf{r}_2\right|}$$

$$v = \mathbf{v}_1 \times \mathbf{e} - \mathbf{v}_2 \times \mathbf{e}$$

- This gives us the instantaneous closing velocity of the two particles

Dampers

- Another way we could compute the closing velocity is to compare the distance between the two particles to their distance from last frame

$$v = \frac{\left|\mathbf{r}_1 - \mathbf{r}_2\right| - d_0}{\Delta t}$$

- The difference is that this is a numerical computation of the approximate derivative, while the first approach was an exact analytical computation

Dampers

- The analytical approach is better for several reasons:
  - Doesn’t require any extra storage
  - Easier to ‘start’ the simulation (doesn’t need any data from last frame)
  - Gives an exact result instead of an approximation
- This issue will show up periodically in physics simulation, but it’s not always as clear cut

This gives us a spring-mass model

- Set of masses
- Springs between them
- Dampers
- Apply forces
  - Gravity
  - Rest length of springs changes with time
  - Pull, push, etc.
  - Now integrate

- Cases
  - Cloth (-ish)
  - Jelly (-ish)
  - Wobbly stuff

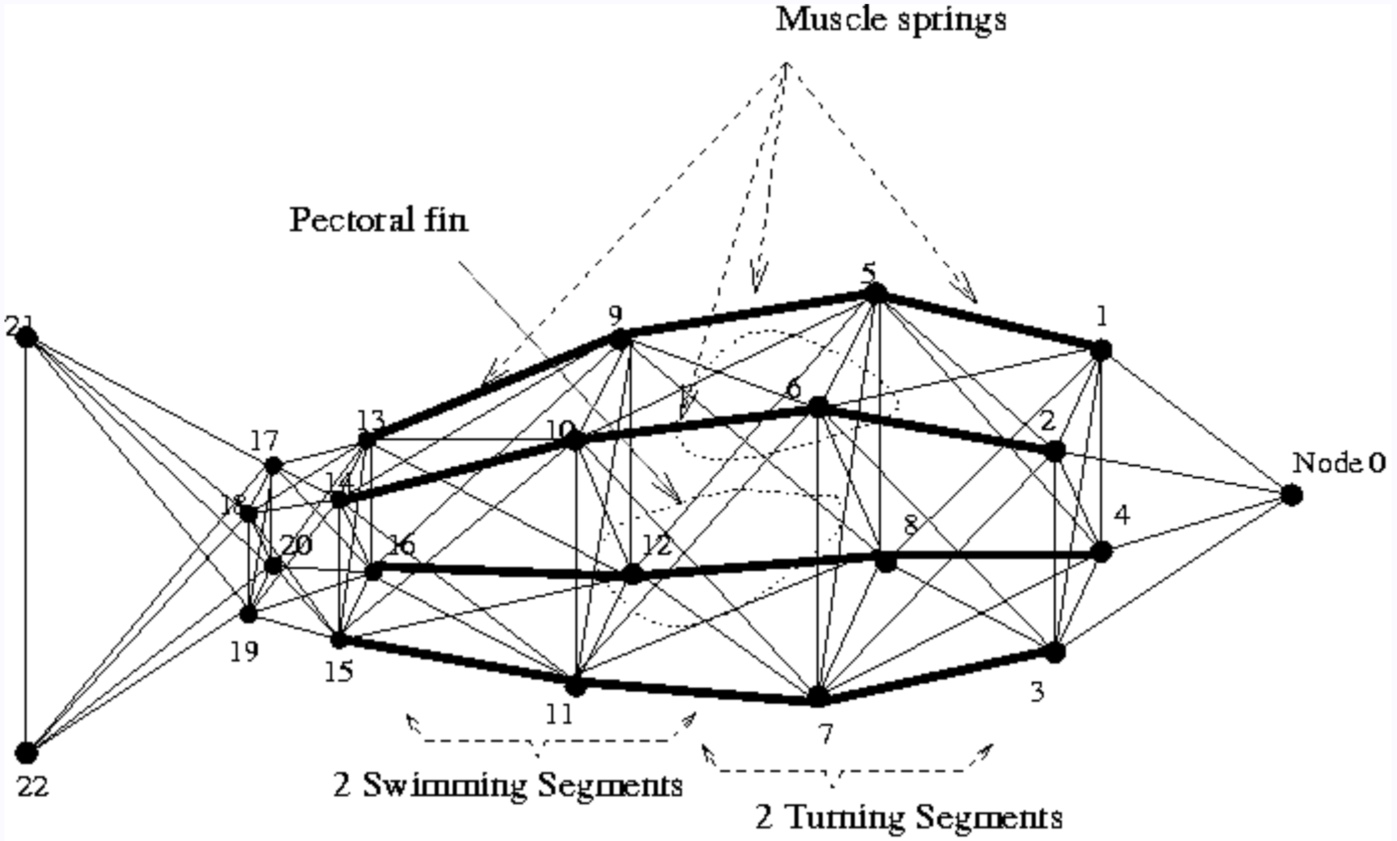

Figure 4.1: The biomechanical fish model. Black dots indicate lumped masses. Lines indicate deformable elements at their natural, rest lengths.

Due to Xiaoyuan Tu, http://www.dgp.toronto.edu/people/tu

Spring Mass fish swimming

Force Fields

- We can also define any arbitrary force field that we want. For example, we might choose a force field where the force is some function of the position within the field

$$\mathbf{f}_{field} \propto \mathbf{f}(\mathbf{r})$$

- We can also do things like defining the velocity of the air by some similar field equation and then using the aerodynamic drag force to compute a final force
- Using this approach, one can define useful turbulence fields, vortices, and other flow patterns

Conservation of Energy

- Particles conserve energy
    - in principle; our integrator might not

Collisions with potential fields

- A potential “barrier” repels
  - so particles bounce off it
  - ideally, without energy gain,
  - BUT...
- As a consequence, “soft” barriers are great, “hard” are a problem
-

Collision detection

- Particles are straightforward
  - -ish (geometry is easy)
  - issues: undetected collisions
    - strategies:
      - take fixed time steps, fixup collision
        - but we may not be able to tell a collision has occurred!
      - potential barrier
        - but this may force quite small time steps; stiffness
        - backward Euler helps, but only within limits
      - identify safe bounds within which to advance time, search
        - use priority queue
        - but this may force quite small time steps

Explicit collisions

- Advance particles in time
- Find collisions
- Resolve using Impulse

Impulses

Collisions & Impulse

- A collision is assumed to be instantaneous
- However, for a force to change an object’s momentum, it must operate over some time interval
- Therefore, we can’t use actual forces to do collisions
- Instead, we introduce the concept of an impulse, which can be though of as a large force acting over a small time

Impulse

- An impulse can be thought of as the integral of a force over some time range, which results in a finite change in momentum:

$$\mathbf{j} = \int \mathbf{f}\,dt = \Delta\mathbf{p}$$

- An impulse behaves a lot like a force, except instead of affecting an object’s acceleration, it directly affects the velocity
- Impulses also obey Newton’s Third Law, and so objects can exchange equal and opposite impulses
- Also, like forces, we can compute a total impulse as the sum of several individual impulses

Impulse

- The addition of impulses makes a slight modification to our particle simulation:

```
// Compute all forces and impulses
$\mathbf{f} = \sum \mathbf{f}_i$
$\mathbf{j} = \sum \mathbf{j}_i$
// Integrate to get new velocity & position
$\mathbf{v}' = \mathbf{v}_0 + \frac{1}{m}(\mathbf{f}\Delta t + \mathbf{j})$
$\mathbf{r}' = \mathbf{r}_0 + \mathbf{v}'\Delta t$
```

Collisions

- Today, we will just consider the simple case of a particle colliding with a static object
- The particle has a velocity of **v** before the collision and collides with the surface with a unit normal **n**
- We want to find the collision impulse **j** applied to the particle during the collision

Elasticity

- There are a lot of physical theories behind collisions
- We will stick to some simplifications
- We will define a quantity called elasticity that will range from 0 to 1, that describes the energy restored in the collision
- An elasticity of 0 indicates that the closing velocity after the collision is 0
- An elasticity of 1 indicates that the closing velocity after the collision is the exact opposite of the closing velocity before the collision

Collisions

- Let’s first consider a collision with no friction
- The collision impulse will be perpendicular to the collision plane (i.e., along the normal)

$$v_{close} = \mathbf{v} \times \mathbf{n}$$

$$\mathbf{j} = -(1 + e) m v_{close} \mathbf{n}$$

- That’s actually enough for collisions today. We will spend a whole lecture on them next week.

Combining Forces

- All of the forces we've examined can be combined by simply adding their contributions
- Remember that the total force on a particle is just the sum of all of the individual forces
- Each frame, we compute all of the forces in the system at the current instant, based on instantaneous information (or numerical approximations if necessary)
- We then integrate things forward by some finite time step

Systems

Particle Systems

- In computer animation, particle systems can be used for a wide variety of purposes, and so the rules governing their behavior may vary
- A good understanding of physics is a great place to start, but we shouldn’t always limit ourselves to following them strictly
- In addition to the physics of particle motion, several other issues should be considered when one uses particle systems in computer animation

Particles

- In physics, a basic particle is defined by it’s position, velocity, and mass
- In computer animation, we may want to add various other properties:
  - Color
  - Size
  - Life span
  - Anything else we want…

Creation & Destruction

- The example system we showed at the beginning had a fixed number of particles
- In practice, we want to be able to create and destroy particles on the fly
- Often times, we have a particle system that generates new particles at some rate
- The new particles are given initial properties according to some creation rule
- Particles then exist for a finite length of time until they are destroyed (based on some other rule)

Creation & Destruction

- This means that we need an efficient way of handling a variable number of particles
- For a realtime system, it’s usually a good idea to allocate a fixed maximum number of particles in an array, and then use a subset of those as active particles
- When a new particle is created, it uses a slot at the end of the array (cost: 1 integer increment)
- When a particle is destroyed, the last particle in the array is copied into its place (cost: 1 integer decrement & 1 particle copy)
- For a high quality animation system where we’re not as concerned about performance, we could just use a big list or variable sized array

Creation Rules

- It’s convenient to have a ‘CreationRule’ as an explicit class that contains information about how new particles are initialized
- This way, different creation rules can be used within the same particle system
- The creation rule would normally contain information about initial positions, velocities, colors, sizes, etc., and the variance on those properties
- A simple way to do creation rules is to store two particles: mean & variance (or min & max)

Creation Rules

- In addition to mean and variance properties, there may be a need to specify some geometry about the particle source
- For example, we could create particles at various points (defined by an array of points), or along lines, or even off of triangles
- One useful effect is to create particles at a random location on a triangle and give them an initial velocity in the direction of the normal. With this technique, we can emit particles off of geometric objects

Destruction

- Particles can be destroyed according to various rules
- A simple rule is to assign a limited life span to each particle (usually, the life span is assigned when the particle is created)
- Each frame, it’s life span decreases until it gets to 0, then the particle is destroyed
- One can add any other rules as well
- Sometimes, we can create new particles where an old one is destroyed. The new particles can start with the position & velocity of the old one, but then can add some variance to the velocity. This is useful for doing fireworks effects…

Randomness

- An important part of making particle systems look good is the use of randomness
- Giving particle properties a good initial random distribution can be very effective
- Properties can be initialized using uniform distributions, Gaussian distributions, or any other function desired

Particle Rendering

- Particles can be rendered using various techniques
  - Points
  - Lines (from last position to current position)
  - Sprites (textured quad's facing the camera)
  - Geometry (small objects…)
  - Or other approaches…
- For the particle physics, we are assuming that a particle has position but no orientation. However, for rendering purposes, we could keep track of a simple orientation and even add some rotating motion, etc…