Nature of Code

Patrick Dwyer
Fall 2005
Week 2 - September 13th

Newton's Laws

We're going to base our simulation of the physics of motion on Newton's laws of motion, a simplified but functional set of rules guiding how bodies react to gravity and mass.

First Law:

When no force acts on an object (or when the forces acting on it cancel), it moves in a straight line at constant speed.

From the first law of Newtonian motion we find that a body at rest will remain at rest unless an external force acts upon it; likewise a body in motion will remain in motion unless acted upon by an external force.

Second Law:

The acceleration of an object equals the total force acting on it, divided by its (constant) mass.

The second law of Newtonian motion outlines how acceleration affects a body. The acceleration applied to an object creates a force on that object equal to the mass of the object multiplied by the acceleration.

Making Sense of Newtonian Physics

- When talking about ***mass*** we are referring to the quantity of matter contained in an object. The ***mass*** for any object is a constant value measured in kilograms.
- The ***weight*** of an object is the force of gravity upon an object and is often confused with ***mass***. The ***mass*** of a 1 kilogram object is the same on Earth as it is on the sun, but the ***weight*** would be significantly different due to gravity. We can use the Second law to calculate the ***weight*** of an object as the ***mass*** * gravity. ***Weight*** is measured in Newtons.

Third Law:

Whenever one body exerts force upon a second body, the second body exerts an equal and opposite force upon the first body.

The third law of Newtonian motion is the common recited, "For every action there is an equal and opposite reaction."

Objects and Forces

Applying Newtonian physics to objects in Processing is fairly straight forward. Our basic object is in a class called "SimObject":

```
public class SimObject {
      Vector2D loc;
      Vector2D vel;
      Vector2D accel;
      float mass;

      public SimObject(Vector2D loc_, Vector2D vel_, Vector2D accel_,
float m_) {
```

```
            loc = loc_;
            vel = vel_;
            accel = accel_;
            mass = m_;
        }
        ...
    }
```

The basic constructor for our object includes the following basic properties:

- loc - The location; a 2D vector that we'll use as a Point
- vel - The velocity of the object; a 2D vector
- accel - The forces of acceleration acting upon the object; a 2D vector
- mass - The mass of the object; a scalar floating point number

With these basic properties we can simulate the motion of the object in our code by following a few simple steps:

- Add the acceleration to the velocity
- Add the velocity to the location
- Reset the acceleration to zero
- Draw our object on screen

Our method in the SimObject class to simulate the motion of the object looks like:

```
    public void update() {

            // add acceleration to velocity
            vel = vel.add(accel);

            // limit our velocity to a maximum speed
            vel.limit(max_velocity);

            // move our object according to our velocity
            loc = loc.add(vel);

            // reset the accerlation forces to zero in
            // preperation for the next step in time
            accel.setXY(0.0f, 0.0f);
    }
```

The last step of our update method resets the acceleration of our object to zero. We do this because at each step of the program we're recalculating the forces that are acting upon our object. Our object can be effected by any number of forces, whose force accumulates. In our basic example there will be two or three forces applied to the object depending upon it's location:

- Wind - There will be a light wind blowing towards the top of the screen
- Gravity - There will be a constant gravity pulling towards the bottom of the screen
- Friction - In the middle of the screen our object encounters an area of friction, which creates a force opposite to the velocity of our object

In a situation where friction is not being applied we can look at the following values to see what is happening to our object:

- Object at location (10, 10) with a mass of 5 and an initial velocity of (-3, -5)
- Wind blowing to the right with force (3, 0)
- Gravity pulling down with a force of (0, 4)

Using our algorithms for calculating the forces being applied to our object (Force = Mass * Acceleration, Acceleration = Force / Mass) we can determine:

Acceleration from Wind:

Wind Force (3, 0) / Mass (5) = (0.6, 0)

Acceleration from Gravity:

Gravity Force (0, 4) / Mass (5) = (0, 0.8)

Total Acceleration:

Wind (0.6, 0) + Gravity (0, 0.8) = (0.6, 0.8)

Object Velocity:

Velocity (-3, -5) + Acceleration (0.6, 0.8) = (-2.4, -4.2)

Location:

Location (10, 10) + Velocity (-2.4, -4.2) = (7.6, 5.8)

We simplify these steps in our code by creating a method that allows us to add a force to our object:

```
void addForce(Vector2D f) {
      f = f.divide(mass);
      accel = accel.add(f);
}
```

Here we divide the vector representing the force being added by the mass of our object, and add the resulting acceleration to our acceleration vector. While we've included mass in our equation here to better follow the rules of Newtonian motion, it isn't necessary to do so, and we'll often skip it in favor of using acceleration = force.

Friction and Viscosity

The equations above assume that we have already found values for and created vectors to represent our forces. Among the forces we can explore are Friction and Viscosity. Friction and Viscosity are “dissipative” forces, that is; they slow down our object. Friction is the force exerted on our object while moving along a solid surface. Viscosity is the force exerted upon our object by moving through a liquid. Both of these forces can create interesting effects, but require different modes of application.

*Friction = -1 * C * (Velocity Unit Vector)*

To calculate the force of friction we need *C*, the “coefficient of friction”. Every surface has a different coefficient of friction, the smoother and more slippery an object the lower the coefficient, the “stickier” the surface and the harder to move against, the higher the coefficient.

*Viscosity = -1 * C * (Velocity Vector)*

Viscosity is directly related to how fast the object is moving, so the higher the velocity the harder it is to move through. *C* in this case is similar to the coefficient of friction above.

Example:

Assume we have a SimObject (so) that we’ve already constructed with a location, velocity and acceleration, and we are moving the object through an area with Friction, where the coefficient of friction is -0.13:

```
float c = -0.13;
Vector2D vel = so.getVelocity();
vel.normalize();
Vector2D force = vel.multiply(c);
so.addForce(force);
```

Gravitational Force

Gravity is the force exerted upon each other by two bodies. To calculate the force on each body we need:

- G, the gravitational constant
- M_1 and M_2, the masses of each object
- D, the distance between the two object
- L_1 and L_2, the location of each object

Gravitational Force Magnitude = $(G * M_1 * M_2) / (D * D)$

While the magnitude of gravitational force is the same for each object, the direction is different for each object:

Gravitational Force Direction (For Object #1) = $(L_2 - L_1) / \| L_2 - L_1 \|$

Here we’ve taken the magnitude of the difference between the location of the two objects ($\| L_2 - L_1 \|$) to calculate the direction of the gravitational force for object #1. In order to make this easier in our SimOb-

Gravity on Earth

We can simply our formulas for gravitational force when dealing with normal objects near the surface of the Earth. In this case the distance D is nearly constant, and most items have a mass that is distinctly less than that of the Earth. Because the surface of the Earth is approximately flat for objects on the ground, we can simply use an approximate value for gravity of 9.8 meters per second per second, or (0, 9.8).

ject, we add a method to the SimObject that allows us to calculate the gravitational attraction of our object to another object:

```
Vector2D getGravityForce(SimObject so) {
      Vector2D direction = Vector2D.subtract(so.getLocation(), loc);
      float distance = direction.magnitude();
      direction.normalize();
      float force = ( G * mass * so.getMass()) / (distance * distance);
      direction = direction.multiply(force);
      return direction;
}
```

If in our program we have an array of SimObjects:

```
SimObject[] sims = new SimObject[5];
```

To calculate the forces these object exert upon one another we would use the following code in our `draw` method:

```
void draw() {
      for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                  if (i != j) {
                        Vector2D f = sims[i].getGravityForce(sims[j]);
                        sims[i].addForce(f);
                  }
            }
            sims[i].simulate();
      }
}
```

Real World versus Simulations

It is important to note that following any of these formulas exactly will not always result in simulations that "look" or "feel" accurate. Scenarios that are true to reality often make for uninteresting or uncontainable visual simulations. For instance, if you use the following values in our Gravitational Force formulas:

- Object 1: Location (10.2, 10.0) and a mass of (5)
- Object 2: Location (10.3, 9.9) and a mass of (5)
- Gravitational Constant = 1.0

The results will indicate a gravitational force of 1250.0 units, which is hard to contain in our screen. Because of this it is often necessary to "fake" or "tweak" our values to make our visualization look interesting. Even small variations in some of our numbers and formulas can throw the equations out of whack, resulting in programs that function erratically or are incomprehensible.

Assignment

Choose one or more of the following and create a program in Processing:

- Change one of the examples to work in 3 dimensions
- Add viscosity to any of the examples
- Find and implement a force not covered here
- Create an example of repulsive force instead of attractive force; for instance an attractor that pushes objects away instead of drawing them in.