Air and fluid resistance

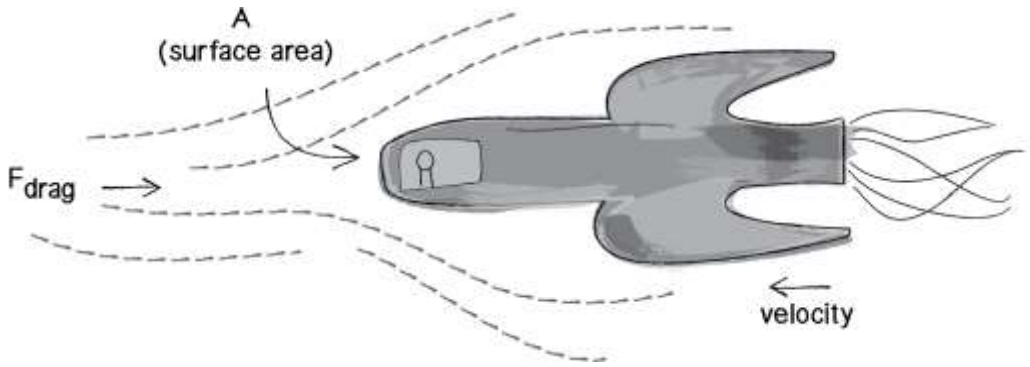


Diagram of fluid resistance around a plane

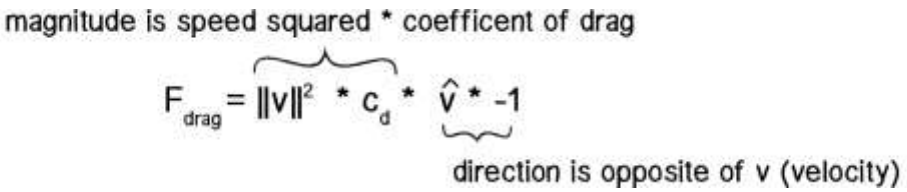
Friction also occurs when a body passes through a liquid or gas. This force has many different names, all really meaning the same thing: viscous force, drag force, fluid resistance. While the result is ultimately the same as our previous friction examples (the object slows down), the way in which we calculate a drag force will be slightly different. Let's look at the formula:

$F_d = -12 \rho v^2 A C_d v$^

Now let's break this down and see what we really need for an effective simulation in ProcessingJS, making ourselves a much simpler formula in the process.

- F_dF_dF, start subscript, d, end subscript refers to drag force, the vector we ultimately want to compute and pass into our `applyForce()` function.
- -1/2 is a constant: -0.5. This is fairly irrelevant in terms of our ProcessingJS world, as we will be making up values for other constants anyway. However, the fact that it is negative is important, as it tells us that the force is in the opposite direction of velocity (just as with friction).
- \rho$\rho$rho is the Greek letter *rho*, and refers to the density of the liquid, something we don't need to worry about. We can simplify the problem and consider this to have a constant value of 1.

- vvv refers to the speed of the object moving. OK, we've got this one! The object's speed is the magnitude of the velocity vector: `velocity.mag()`. And v^2v2v, start superscript, 2, end superscript just means vvv squared or v * vv∗vv, times, v.
- AAA refers to the frontal area of the object that is pushing through the liquid (or gas). An aerodynamic Lamborghini, for example, will experience less air resistance than a boxy Volvo. Nevertheless, for a basic simulation, we can consider our object to be spherical and ignore this element.
- C_dCdC, start subscript, d, end subscript is the coefficient of drag, exactly the same as the coefficient of friction (ρ). This is a constant we'll determine based on whether we want the drag force to be strong or weak.
- \hat{v}v^ Look familiar? It should. This refers to the velocity unit vector, i.e. `velocity.normalize()`. Just like with friction, drag is a force that points in the opposite direction of velocity.
  Now that we've analysed each of these components and determined what we need for a simple simulation, we can reduce our formula to:



Simplified formula: F_drag = ||v^2|| * c_d * v - 1

or:

```
// Part 1 of our formula (magnitude): v^2 * Cd

var c = 0.1;

var speed = v.mag();

var dragMagnitude = c * speed * speed;


// Part 2 of our formula (direction): v unit vector * -1

var drag = velocity.get();
```

```
drag.normalize();
drag.mult(-1);

// Magnitude and direction together!
drag.mult(dragMagnitude);
```

Let's implement this force in our `Mover` object type with one addition. When we wrote our friction example, the force of friction was always present. Whenever an object was moving, friction would slow it down. Here, let's introduce an element to the environment—a "liquid" that the `Mover` objects pass through. The `Liquid` object will be a rectangle and will know about its location, width, height, and "coefficient of drag"—i.e., is it easy for objects to move through it (like air) or difficult (like molasses)? In addition, it should include a function to draw itself on the screen (and two more functions, which we'll see in a moment).

```
var Liquid = function(x, y, w, h, c) {
  this.x = x;
  this.y = y;
  this.w = w;
  this.h = h;
  this.c = c;
};

Liquid.prototype.display = function() {
  noStroke();
  fill(50);
  rect(this.x, this.y, this.w, this.h);
};
```

The main program will now declare and initialize a new `Liquid` object instance. Note the coefficient is low (0.1), otherwise the object would come to a halt fairly quickly (which may someday be the effect you want).

```
var liquid = new Liquid(0, height/2, width, height/2, 0.1);
```

Now comes an interesting question: how do we get the `Mover` object to talk to the `Liquid` object? In other words, we want to execute the following:

When a mover passes through a liquid it experiences a drag force.

…or in object-oriented speak (assuming we are looping through an array of `Mover` objects with index `i`):

```
// Is the Mover in the liquid?
if (liquid.contains(movers[i])) {
    // Calculate drag force
    var dragForce = liquid.calculateDrag(movers[i]);
    // Apply drag force to Mover
    movers[i].applyForce(dragForce);
}
```

The above code tells us that we need to add two functions to the `Liquid` object type: (1) a function that determines if a `Mover` object is inside the `Liquid` object, and (2) a function that computes the drag force exerted on the `Mover` object.

The first is easy; we can simply use a conditional statement to determine if the location vector rests inside the rectangle defined by the liquid.

```
Liquid.prototype.contains = function(m) {
    var p = m.position;
    return p.x > this.x && p.x < this.x + this.w &&
         p.y > this.y && p.y < this.y + this.h;
```

```
};
```

The `drag()` function is a bit more complicated; however, we've written the code for it already. This is simply an implementation of our formula. The drag force is equal to *the coefficient of drag multiplied by the speed of the `Mover` squared in the opposite direction of velocity!*

```
Liquid.prototype.calculateDrag = function(m) {
  // Magnitude is coefficient * speed squared
  var speed = m.velocity.mag();
  var dragMagnitude = this.c * speed * speed;

  // Direction is inverse of velocity
  var dragForce = m.velocity.get();
  dragForce.mult(-1);

  // Scale according to magnitude
  // dragForce.setMag(dragMagnitude);
  dragForce.normalize();
  dragForce.mult(dragMagnitude);
  return dragForce;
};
```

And with these two functions added to the `Liquid` object type, we're ready to put it all together into one program:

```
/* Forces (Gravity and Fluid Resistance) with Vectors
 * Demonstration of multiple force acting on bodies (Mover object)
 * Bodies experience gravity continuously
 * Bodies experience fluid resistance when in "water"
 */
```

```
var Liquid = function(x, y, w, h, c) {
   this.x = x;
   this.y = y;
   this.w = w;
   this.h = h;
   this.c = c;
};

// Is the Mover in the Liquid?
Liquid.prototype.contains = function(m) {
   var p = m.position;
   return p.x > this.x && p.x < this.x + this.w &&
         p.y > this.y && p.y < this.y + this.h;
};

// Calculate drag force
Liquid.prototype.calculateDrag = function(m) {
   // Magnitude is coefficient * speed squared
   var speed = m.velocity.mag();
   var dragMagnitude = this.c * speed * speed;

   // Direction is inverse of velocity
   var dragForce = m.velocity.get();
   dragForce.mult(-1);

   // Scale according to magnitude
```

```
  // dragForce.setMag(dragMagnitude);
  dragForce.normalize();
  dragForce.mult(dragMagnitude);
  return dragForce;
};

Liquid.prototype.display = function() {
  noStroke();
  fill(28, 120, 186);
  rect(this.x, this.y, this.w, this.h);
};

var Mover = function(m, x, y) {
  this.mass = m;
  this.position = new PVector(x, y);
  this.velocity = new PVector(0, 0);
  this.acceleration = new PVector(0, 0);
};

Mover.prototype.applyForce = function(force) {
  var f = PVector.div(force, this.mass);
  this.acceleration.add(f);
};

Mover.prototype.update = function() {
  this.velocity.add(this.acceleration);
  this.position.add(this.velocity);
```

```
  this.acceleration.mult(0);
};

Mover.prototype.display = function() {
  stroke(0, 0, 0);
  strokeWeight(2);
  fill(123, 217, 176);
  ellipse(this.position.x, this.position.y, this.mass*16, this.mass*16);
};

Mover.prototype.checkEdges = function() {
  if (this.position.x > width) {
    this.position.x = width;
    this.velocity.x *= -1;
  } else if (this.position.x < 0) {
    this.velocity.x *= -1;
    this.position.x = 0;
  }
  if (this.position.y > height) {
    this.velocity.y *= -1;
    this.position.y = height;
  }
};

// Moving bodies
var movers = [];
// Create liquid object
```

```
var liquid = new Liquid(0, height/2, width, height/2, 0.1);

var draw = function() {
  background(219, 253, 255);

  // Draw water
  liquid.display();

  for (var i = 0; i < movers.length; i++) {

    // Is the Mover in the liquid?
    if (liquid.contains(movers[i])) {
      // Calculate drag force
      var dragForce = liquid.calculateDrag(movers[i]);
      // Apply drag force to Mover
      movers[i].applyForce(dragForce);
    }

    // Gravity is scaled by mass here!
    var gravity = new PVector(0, 0.1*movers[i].mass);
    // Apply gravity
    movers[i].applyForce(gravity);

    // Update and display
    movers[i].update();
    movers[i].display();
    movers[i].checkEdges();
```

```
  }


  fill(0, 0, 0);
  text("click mouse to reset",10,30);
};



// Restart all the Mover objects randomly
var resetMovers = function() {
  for (var i = 0; i < 9; i++) {
    movers[i] = new Mover(random(0.5, 3), 20+i*width/9, 0);
  }
};


// Not working???
var mousePressed = function() {
  resetMovers();
};


resetMovers();
```

Running the program, you should notice that we are simulating balls falling into water. The objects only slow down when crossing through the gray area at the bottom of the window (representing the liquid). You'll also notice that the smaller objects slow down a great deal more than the larger objects. Remember Newton's second law? `A = F / M`. Acceleration equals force divided by mass. A massive object will accelerate less. A smaller object will accelerate more. In this case, the acceleration we're talking about is the

“slowing down” due to drag. The smaller objects will slow down at a greater rate than the larger ones.