Introduction To Rigid Body Physics

Module 1: The Motion of Particles

University of Reddit
Instructor: Brian Dolhansky
Course website: http://universityofreddit.com/v2/class.php?id=111
Course email: intro.to.rbp@gmail.com
Instructor email: bdolmail@gmail.com

Contents

1 Introduction

This lecture will cover the motion of particles, or zero-dimensional objects. We will go over how we describe the instantaneous state of a particle, both in one dimension using scalars and in two dimensions using vectors. We will then set up several classes that implement our ideas.

While the Suggested Reading section gives a few basic links on the information I will cover in this lecture, check the References section for more in-depth reading. These are the articles I used to create this lesson as they provide more rigorous proofs and examples.

2 Suggested reading

The first website provides some background information on basic calculus operations in case you are a little rusty on how to take derivatives and integrals. The second link gives a good overview of basic vector math. The final link is a good primer on linear motion.

- http://www.sosmath.com/calculus/calculus.html
- http://hyperphysics.phy-astr.gsu.edu/hbase/vect.html
- http://www.gamedev.net/reference/articles/article434.asp

3 Describing the motion of particles

3.1 Overview

In both the real and virtual world, things are constantly moving. Whether it's the car you drive to work or the virtual race car you control after work, the motion of each object is very important. After all, a car that doesn't move is useless, and a game in which nothing happens would be very boring indeed! Although we might be able to describe the location and velocity of cars on the highway using words (that car *in the far lane* is moving *very quickly*), how do we translate these terms into something a computer might understand?

We can't exactly tell our computer to make a car go *over there* and to move *rather slowly*. Over where? Slowly compared to what? Different cars have different speeds. Well, we could tell the computer to move the car at, say, 10 m.p.h., to the third traffic light on Main Street. But then what direction should the car move in? Does it need to stop moving and change

directions?

Figure 1: This car may have been moving a *bit* too quickly.

The point I am trying to belabor is that we cannot describe motion *qualitatively*. Rather, we need to *quantify* motion. How, can we describe our physical world in terms that a computer understands? First, we need come up with ways to efficiently represent aspects of motion, such as position, velocity and acceleration using *only* numbers. We then have to derive equations that relate how each aspect affects the others.

Thankfully, these steps have already been figured out for us! However, we are going to make a few generalizations for this lesson. One, we are not going to cover how these objects started moving in the first place. This will be covered in Module 2. Two, we will only describe the motion of point particles in this module. That is, we aren't describing how an object might be spinning around. More accurately, *rotation* doesn't have any meaning for point particles, as they are zero-dimensional. There is nothing there to "spin."

3.2 Motion in one dimension

3.2.1 Position and displacement

To simplify things and hopefully to make some of the derivations I'm about to go over more intuitive, we will first examine motion in only one dimension. We can think of motion in one dimension being similar to an ant on a thin piece of material. For example, the ant (we'll call him Alfred) can only move forward and backward on the tube. If he moves left or right, he will fall off!

Figure 2: Alfred leads a one-dimensional life.

But where exactly is Alfred on the tube? Well, he looks to be about an inch away from the end. We could certainly describe his position in this fashion.

Figure 3: Alfred is one inch from certain doom.

And if we wanted Alfred to move to the other end of the tube, we could quantitatively tell him to *move 10 inches away from the left part of the tube.*

Figure 4: Alfred is less than one inch from certain doom.

Another way we can describe Alfred's position is by using the term displacement. He is at position 10, but he is also 10 inches from the beginning. He is also less than one inch from the end of the tube. It is important that you understand the distinction, because we will be examining this concept further when we explore world and body coordinates.

3.2.2 Velocity and acceleration

So far so good. However, Alfred doesn't have the power of teleportation. He walked to the end of the tube with a certain speed. We call this property velocity. Let's say that it took Alfred 10 seconds to traverse the length of the tube. Therefore, we could say his velocity was one inch per second, or $1\frac{\text{in}}{\text{s}}$. His motion might look something like this:

Figure 5: Alfred's stroll.

With each time step, Alfred's displacement increased by 1 inch. But what if Alfred didn't walk at a constant speed? What if he was in a hurry to get to the end? Let's say that with time, Alfred picked up his pace. We call this acceleration. If every second, Alfred increased

his speed by $1\frac{\text{in}}{\text{s}}$, he would be accelerating by $1\frac{\text{in}}{\text{s}^2}$. Note the units on the bottom of that quantity. Using dimensional analysis, it is easy to see why the units of acceleration include a time squared term on the bottom. Alfred increases his speed by $1\frac{\text{in}}{\text{s}}$ *per second*, or $1\frac{\text{in}}{\text{s}} \times 1\frac{1}{s}$. Multiplying the two quantities gives us a unit of $\frac{\text{in}}{\text{s}^2}$. Alfred's motion with acceleration might look something like the figure below. From the picture, it is clear that it only takes Alfred 4s to get to the end of the tube.

Figure 6: Alfred's sprint.

3.2.3 General equations of motion in one dimension

Now that we have a simple example of displacement, velocity and acceleration, is it possible to generalize these properties and relate them to one another? The astute student will notice that Alfred's position at a certain time t is entirely dependent on his initial position and his velocity (assuming acceleration is 0). We can say that Alfred's displacement is a *function of velocity*. Let's explore this concept further using graphs. First, let's say that Alfred is feeling particularly lazy one day, and simply sits at a position of 5 inches for a while. We could plot his displacement versus time, shown in Figure 7.

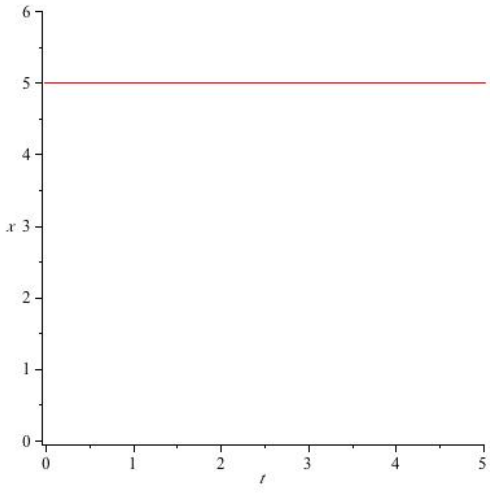

Figure 7: Detailed analysis of Alfred's laziness.

Because Alfred is being lazy, his velocity is zero. While you can qualitatively determine that from the above plot, how do we *quantitatively* figure out his velocity? If you guessed that we should take the derivative, you'd be correct! This is one of the most important concepts in rigid body physics - *velocity is the derivative of position.*

To further illustrate this point, let's instead go over a simple example. Say Alfred starts at a position of 0 inches and moves towards the opposite end of the tube at a constant speed of 1 $\frac{\text{in}}{\text{s}}$. His displacement plot would look like Figure 8. Hopefully you remember that, graphically, the slope of a tangent line touching a curve at a certain point is that curve's derivative at the same point. In this case, the derivative (velocity) is constant along the entire curve because Alfred is moving at a constant speed.

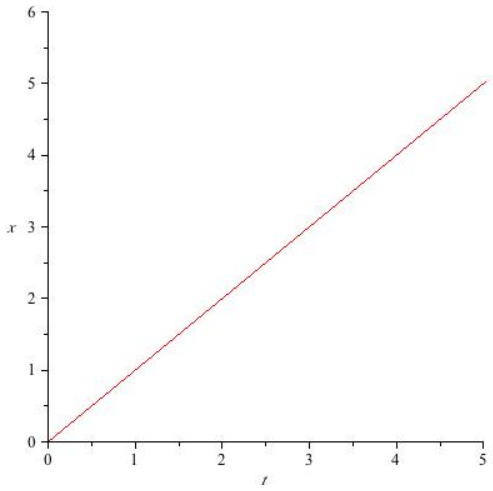

Figure 8: Detailed analysis of Alfred's plodding.

Remember that the processes of integration and derivation are essentially inverse operations. So if velocity is the derivative of position, does that mean we can integrate velocity to get displacement? Yes, it does! Instead of plotting Alfred's displacement versus time, let's instead plot his velocity versus time, and take the area under the curve. Remember that the graphical interpretation of integrating a curve over a certain time period is simply the area under the curve. Figure 9 shows Alfred's velocity over time. The plot is simply a constant line, because his velocity does not change. Therefore, the area is easy to compute. Alfred's displacement after 5 seconds is simply the length times height of the rectangle shown (5 seconds times 1 inch per second), or 5 inches. This result matches our prediction!

This is is also an important concept to note - *displacement can be found by integrating velocity*. This notion is the backbone of any simulator. The integrator, or the portion of a simulator that determines an object's position based on velocity (and acceleration, which we are about to cover), is one of the most important "parts" of the machine. The difference between an unstable and robust simulator is often based on the quality of the integrator.

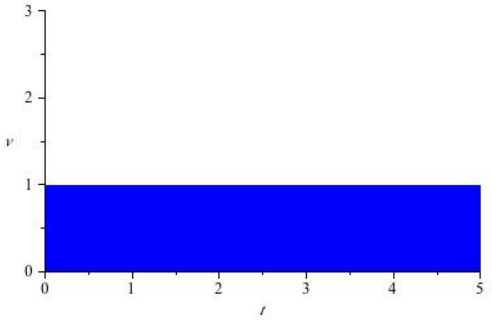

Figure 9: The area under an object's velocity versus time curve is its displacement.

Because plotting these concepts is getting rather tedious, I will use mathematical notation to relate a one-dimensional object's acceleration to its displacement. Alfred's displacement is a function of initial position, velocity (we'll say velocity is constant for now) and time. That is, if we know all three of those quantities, we can determine a particle's position exactly for any time. Let's denote Alfred's displacement by s, his initial position by s_0, his current velocity v and time by t. Therefore, the formula for Alfred's position is:

$$s(t) = vt + s_0 \tag{1}$$

Hopefully the above equation makes sense intuitively. Alfred's position increases (or decreases) linearly at the rate of his velocity, assuming his acceleration is zero. In addition, a different starting position will change the displacement at time t. Let's use the example above to solidify this concept. Let's say Alfred starts at $s_0 = 2$ and his velocity is simply $1\frac{\text{in}}{\text{s}}$. What is his position at $t = 3$? Well, we can use Equation 1 to determine the answer.

$$\begin{aligned} s(3) &= 1(3) + 2 \\ s(3) &= 5 \end{aligned}$$

This matches the example above! Let's prove our previous hypothesis that velocity is the derivative of displacement, and that you can find displacement by integrating velocity (this is assuming acceleration is 0).

$$\begin{aligned} s(t) &= vt + s_0 \\ \frac{ds}{dt} &= v \\ \int v\ dt &= vt + s_0 \\ \int_0^3 1\ dt &= 1(3) + 2 \\ s(3) &= 5 \end{aligned}$$

You can see that, in one dimension, displacement and velocity are related.

Note
For simplicity's sake, from now on we will use Newton's notation for derivatives. That is, a single derivative will be denoted by a single dot over the variable of interest, and a double derivative by two dots. For example, the first and second derivatives of x are $\dot{x}$ and $\ddot{x}$, respectively.

I had mentioned earlier that not only are velocity and displacement related, but displacement, velocity and acceleration are all interrelated. How does acceleration fit into the picture? Well, according to the definition we specified earlier, acceleration is how velocity is changing. In other words, we can again use the derivative to derive a formula for acceleration. This time we will derive the velocity function. If we allow velocity to have an initial value, and we recognize that velocity changes linearly with acceleration, we arrive at the following equation (where a is the acceleration at the current time step):

$$v(t) = at + v_0 \tag{2}$$

To check our previous assertion, let's derive the velocity equation and integrate acceleration:

$$\begin{aligned} v(t) &= at + v_0 \\ \dot{v} &= a \\ \int a\ dv &= at + v_0 \end{aligned}$$

Now that we've related displacement and velocity, and we've also connected velocity and acceleration, can we relate displacement and acceleration? Let's integrate acceleration and see what we arrive at (hopefully you still remember the rules of integration).

$$\int a\ dv = v(t) = at + v_0$$
$$\int at + v_0\ dt = s(t) = \frac{1}{2}at^2 + v_0 t + s_0$$

This is a very important result. If we assume the simple case, that an object starts at displacement 0 with an initial velocity of 0, we can determine where an object is at any time *simply by knowing an object's acceleration*! This forms the foundation of our rigid body simulator. Using a numerical integrator, the basic steps we use to update our objects might look something like Figure 10:

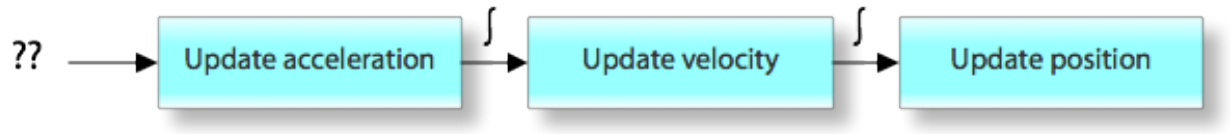

Figure 10: The high-level flow of a physics simulator.

You may say that we can't just set the acceleration of an object out of nowhere. Something *causes* acceleration. That is represented by the "??" in our diagram. We will cover what that something is in the next lesson.

3.3 Vectors

Because vectors are the building blocks of most of the concepts we will be going over in 2 dimensions, we need to take a detour and go over some vector math.

3.3.1 Definition

Vectors have many different definitions depending on the field we're discussing. For instance, in computer science, a vector is a data structure,[1] while vector graphics use primitives to create infinte-resolution graphics.[2] In our case, we will be using Euclidean vectors, which are

"geometric objects that have both length and direction."[3]

There are two ways we can represent vectors, by giving the magnitude (length) and direction, or by giving the components of the vector. The first method is not very useful in this application, so we will use the second. Figure 11 graphically shows the components of a vector. Essentially, v_x is the number of units along the bottom of the grid the point of the vector is, and v_y is the number of units along the left side of the grid the point of the vector is.

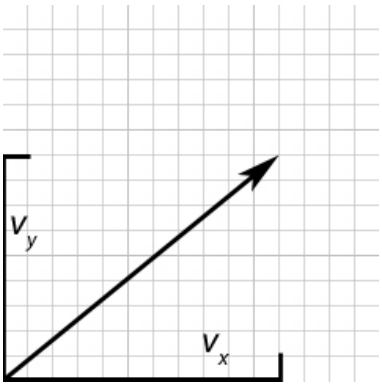

Figure 11: A vector decomposed into its components.

Note
The following notation will now be used for vectors: $$\mathbf{v} = \langle v_x, v_y \rangle$$

3.3.2 Vector magnitude

The magnitude of a vector is simply its "length." We'll see that later on, the concept of length doesn't really have much meaning in our application. This is why we use the term "magnitude," because the length of a vector is often better defined as "strength."

Note
The following notation will now be used to define vector magnitude: $$\|\mathbf{v}\|$$

Finding the magnitude of a vector is based geometrically, specifically in the Pythagorean theorem:

$$\begin{aligned} c^2 &= a^2 + b^2 \\ c &= \sqrt{a^2 + b^2} \end{aligned}$$

Consider the arbitrary vector shown in Figure 12. We can imagine the vector composing the hypotenuse of a right triangle, while its components are the sides of the triangle.

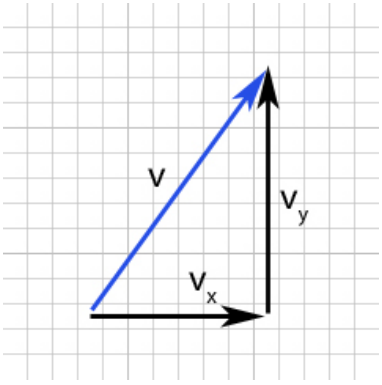

Figure 12: A vector forms the hypotenuse of a right triangle.

If this analogy is clear, we can relate Pythagoras's theorem to vector magnitude. Because a vector can be viewed as the hypotenuse of a right triangle, its magnitude can be found in much the same way.

$$|\mathbf{v}|^2 = v_x^2 + v_y^2 \tag{3}$$

$$|\mathbf{v}| = \sqrt{v_x^2 + v_y^2} \tag{4}$$

3.3.3 Vector scaling

Multiplying a vector by a scalar, well, scales the vector. That is, the vector's components and magnitude are multiplied by the scalar. To scale a vector, simply multiply the components

by the scalar:

$$a\mathbf{v} = \langle av_x, av_y \rangle \tag{5}$$

The following proves that the magnitude of the vector is *also* scaled by the scalar:

$$\begin{aligned} a\,|\mathbf{v}| &= \sqrt{(av_x)^2 + (av_y)^2} \\ a\,|\mathbf{v}| &= \sqrt{(a^2)(v_x^2 + v_y^2)} \\ a\,|\mathbf{v}| &= a\sqrt{v_x^2 + v_y^2} \end{aligned}$$

3.3.4 Vector addition and subtraction

Vector addition and subtraction are straightforward operations. To add two vectors, simply sum the x and y components of each to obtain the new vector. Formally:

$$\mathbf{c} = \mathbf{a} + \mathbf{b} \tag{6}$$

$$\mathbf{c} = \langle a_x + b_x, a_y + b_y \rangle \tag{7}$$

Vector subtraction is the same:

$$\mathbf{c} = \mathbf{a} - \mathbf{b} \tag{8}$$

$$\mathbf{c} = \langle a_x - b_x, a_y - b_y \rangle \tag{9}$$

Let's go over a graphical example of vector addition. We will use a simple case, where the vectors we are using are orthogonal, that is, the two vectors form a right angle. We will define our vectors as $a = \langle 7, 0 \rangle$ and $b = \langle 0, 10 \rangle$. To add two vectors graphically, we simply line the tail up with the head of the other. The summed vector can be drawn from the origin to the head of the second vector, as can be seen in Figure 13.

These examples are pretty simple, but it's important that you understand the basic concepts of vector math. There are more vector operations than the few we covered here, but we will go over those when the need arises. For now, we will be able to build a point-mass simulator with only vector addition and scaling.

3.3.5 Exercise: Vector class

Because vectors are such an integral part of our simulator, we will certainly need a vector class that contains all of the necessary operations. As an exercise, try coding your own Vec-

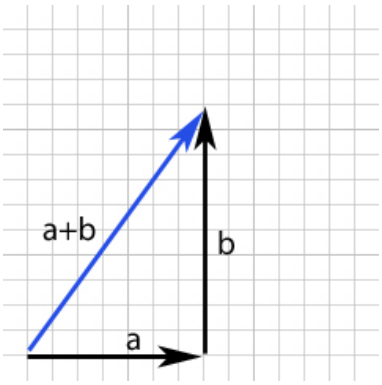

Figure 13: The result of vector addition.

tor class. For ease of use, let's put our Vector class in a Python module called `rbpmath.py`. We'll be including this module in most of our classes from now on.

For practice, I've included `rbpmath_stub.py` in the `src/` directory of this Module. It looks something like this:

```
import math

class Vector(object):
        # Initializes 2d vector.  Default is 0.
        def __init__(self, x=0, y=0):
                self.x = x
                self.y = y

        # Prints components of vector.
        def __str__(self):
                return "X: " + str(self.x) + " Y: " + str(self.y)

        # Checks to see if the components of two vectors are equal.
        def __eq__(self, other):
                if isinstance(other, Vector):
                        return ((self.x == other.x) and (self.y == other.y)
                else:
                        return NotImplemented

        # Checks to see if the components are not equal.
        def __ne__(self, other):
                eq = self.__eq__(other)
```

```
            if eq is NotImplemented:
                return NotImplemented
            return not eq

    # Vector addition
    def __add__(self, other):

    # Vector addition and assignment
    def __iadd__(self, other):

    # Vector subtraction
    def __sub__(self, other):

    # Vector subtraction and assignment
    def __isub__(self, other):

    # Scalar multiplication
    def __mul__(self, other):

    # Scalar multiplication and assignment
    def __imul__(self, other):

    # Returns the magnitude of the vector.
    def magnitude(self):

    # Returns tuple of components for drawing functions
    def components(self):
```

All the methods surrounded by double underscores are operator overloads. We're going to be doing a lot of vector math, so using fully-named methods would get cumbersome. You can read the full documentation online,[4] but for convenience the relevant operations are given in Table 1.

See if you can fill out the method stubs using the definitions given in the preceding sections. If you want to check your answers, I've also included the current version of `rbpmath.py` in the Module's source directory.

Table 1: Python Operator Overloading

Operator	*Method name*
==	__eq__
!=	__ne__
+	__add__
+=	__iadd__
-	__sub__
-=	__isub__
*	__mul__
*=	__imul__

3.4 Motion in two-dimensions

3.4.1 Position and displacement

Now that we have a solid understanding of the simple motion of point masses in one dimension, we need to come up with a description of movement in two dimensions. Although this may seem like a daunting task, in reality, linear movement in two dimensions is closely related to movement in one dimension.

The main difference between one and two dimensions is how we represent position, velocity and acceleration. In one dimension, we could represent each of these quantities with a scalar, or one number. Alfred's position could only increase or decrease - he couldn't move up or down. However, this does not apply in two dimensions. Imagine Alfred is now on a large sheet of paper. He can move up, down, left and right.

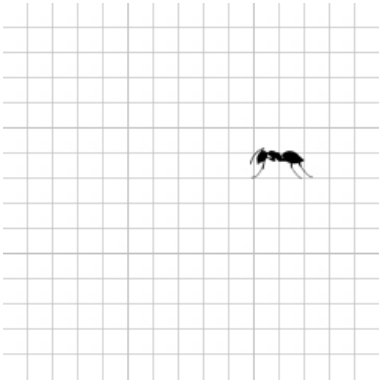

Figure 14: Alfred and his newfound sense of freedom.

How do we represent Alfred's position? Remember, we can't use a scalar to describe any position Alfred might take. While he may be 3 inches from the top-right corner of the piece of paper, if he moves left or right this method of measurement is meaningless.

We could certainly give Alfred's position along each edge of the paper using Cartesian coordinates.

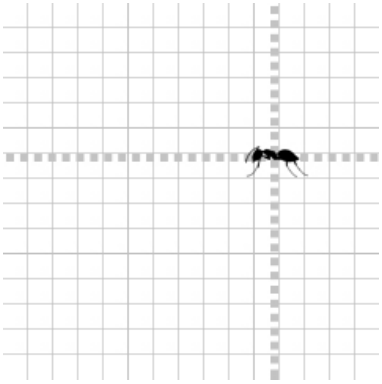

Figure 15: The position of Alfred in Cartesian coordinates.

Let's say Alfred is 6 inches along the bottom of the paper and 5 inches along the right side of the paper. We could give Alfred's position simply as $(6, 5)$. Another way to visualize this is to imagine Alfred sitting on the head of an arrow that starts at the bottom left corner of the paper, or, in other words, a vector. Since the head of the vector is also 6 inches along the bottom of the paper and 5 inches along the right side, we can also denote this with a vector, $\langle 6, 5 \rangle$. The tail of the vector lies on the origin, (0,0). We use vectors to describe position, velocity and acceleration in 2 dimensions.

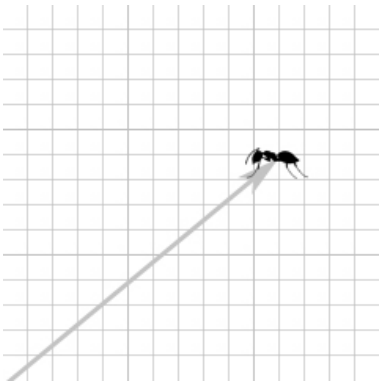

Figure 16: The vector describing Alfred's position.

Using vectors is an easy way to store position in 2D. All of our 2D bodies will have a "position" attribute that is simply an instance of the Vector class.

Let's lay down a more rigorous definition of position in 2D. Because we are working in two dimensions, we will not represent Alfred's position using a function with one variable. We will define his position (represented by $\mathbf{s}$) parametrically. That is, we will describe his position *as functions along both the x and y axes*. Remember, position is a vector!

$$\mathbf{s}(t) = \langle x(t), y(t) \rangle \tag{10}$$

From the notation in Equation 10, it is clear that we can represent Alfred's position at any time t with a vector by determining the values of $x(t)$ and $y(t)$ at the same time t. How do we come up with the individual components of $\mathbf{s}(t)$? By integrating velocity in 2D.

3.4.2 Velocity

As was mentioned earlier, velocity in 2D is also represented by a vector. Suppose Alfred takes a mathematically defined stroll one day. Because we are working two dimensions, we define his path parametrically. For example, let's say we know the definitions of $x(t)$ and $y(t)$ from Equation 10.

$$\begin{aligned} s(t) &= \langle x(t), y(t) \rangle, \\ x(t) &= \frac{1}{2}t^2 \\ y(t) &= 2t - t^2 \end{aligned}$$

As we plug in various values of t (time), we will get a different value for x and y. Remember that position is expressed as a vector. The plot of Alfred's position is given in Figure 17.

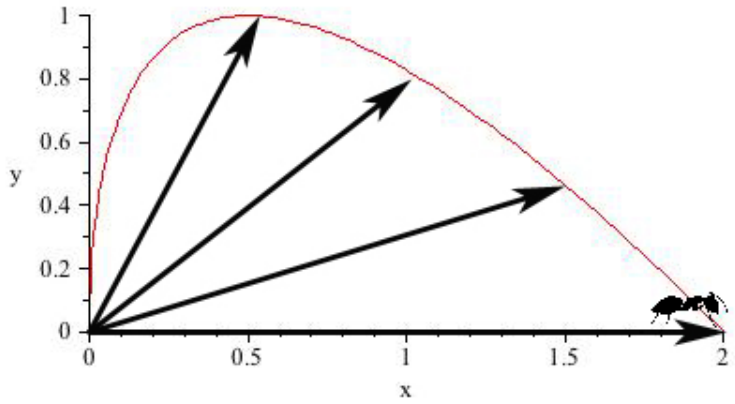

Figure 17: Alfred's position with position vectors at various times.

The same kinematic rules for one dimension apply to two dimensions - velocity is still the derivative of position. Therefore, we can derive $s(t)$ to obtain $v(t)$:

$$\begin{aligned} \dot{x}(t) &= t \\ \dot{y}(t) &= 2 - 2t \\ \mathbf{v}(t) &= \langle t, 2 - 2t \rangle \end{aligned}$$

If we use $t = 1$, we obtain a velocity of $\langle 1, 0 \rangle$. Remember, velocity is *also a vector*. To verify our result, we plot the velocity at $t = 1$ on our position plot. The velocity vector should lie tangent to the curve.

As we can see in Figure 18, the velocity does indeed lie tangent to the position curve, proving that velocity is the derivative of displacement in two dimensions. Likewise, the

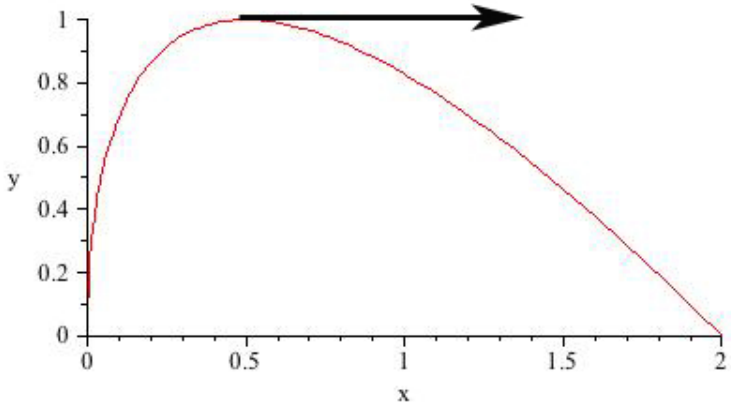

Figure 18: Alfred's velocity at $t = 1$.

integral of velocity versus time is displacement:

$$\mathbf{v} = \dot{\mathbf{s}} \tag{11}$$

$$\int \mathbf{v}\, dt = \mathbf{s} + \mathbf{s_0} \tag{12}$$

3.4.3 Acceleration

I hope that you are now convinced that displacement, velocity and acceleration are all related. Just as in one dimension, acceleration is the derivative of velocity, and velocity can be found by integrating acceleration:

$$\mathbf{a} = \dot{\mathbf{v}} \tag{13}$$

$$\int \mathbf{a}\, dt = \mathbf{v} + \mathbf{v_0} \tag{14}$$

3.4.4 Exercise: Body class

Position, velocity and acceleration are all intrinsic attributes for non-static bodies, whether they are point masses or rigid bodies. Therefore, it would make sense for us to create a

"Body" superclass that holds these values. All moving bodies would then inherit this class. We would then have a generic interface to hand off to our integrator.

Given below is the start of a simple Body superclass. We'll put this in a module called `rbpbodies.py`. See if you can fill in the missing assignments in the `__init__` function. You should make sure that the values passed for position, velocity and acceleration are vectors and not scalars (*Hint:* look up the `isinstance` method). Again, the full class is given in the Code section, but give it a try first!

```
from tmath import Vector

class Body(object):
        def __init__(self, position, velocity, acceleration):

        # Prints information about the body.
        def __str__(self):
                return  "Position: " + str(self.pos) + "\n"\
                                "Velocity: " + str(self.vel) + "\n"\
                                "Acceleration: " + str(self.acc) + "\n"
```

4 Conclusion

We've covered a lot in this chapter. We now know the fundamental equations of motion in both one and two dimensions, and how we can represent two dimensional values in an efficient matter. In addition, we've laid the foundation down for our actual simulator by writing both a Vector class and a general Body superclass that the rest of our objects will inherit.

The next lesson will cover how we determine acceleration based on the forces acting on a body. In addition, we will implement Euler's method to begin making our bodies move on-screen. For now, make sure that it is very clear that if all we have is a body's acceleration, initial position and velocity, we can determine that body's position for all future values. We will further expound on this concept in the next lesson.

You may be disappointed that we didn't program anything that moves on screen. Don't worry! By the end of the next lesson, we will have a simple fireworks simulation complete with gravity, wind and burst physics. However, we need to learn about integrators first,

which is why our simulation must wait until next week.

5 Chapter summary

Review

Vector math

$$\begin{aligned} |\mathbf{v}| &= \sqrt{\mathbf{v_x}^2 + \mathbf{v_y}^2} \\ a\mathbf{v} &= \langle a\mathbf{v_x}, a\mathbf{v_y} \rangle \\ \mathbf{v} + \mathbf{u} &= \langle \mathbf{v_x} + \mathbf{u_x}, \mathbf{v_y} + \mathbf{u_y} \rangle \\ \mathbf{v} - \mathbf{u} &= \langle \mathbf{v_x} - \mathbf{u_x}, \mathbf{v_y} - \mathbf{u_y} \rangle \end{aligned}$$

Equations of motion in one dimension

$$\begin{aligned} a(t) &= a \\ v(t) &= at + v_0 \\ s(t) &= \frac{1}{2}at^2 + vt + s_0 \end{aligned}$$

Equations of motion in two dimensions

$$\begin{aligned} \mathbf{a}(t) &= \mathbf{a} \\ \mathbf{v}(t) &= \mathbf{a}t + \mathbf{v_0} \\ \mathbf{s}(t) &= \frac{1}{2}\mathbf{a}t^2 + \mathbf{v}t + \mathbf{s_0} \end{aligned}$$

$$\begin{aligned} \mathbf{s} &= \int \mathbf{v}\ dt \\ \mathbf{v} &= \int \mathbf{a}\ dt \end{aligned}$$

6 Keyword Definitions

acceleration

The rate of change of velocity. It is represented by a scalar in one dimension and a vector in two dimensions. Acceleration over time can be integrated to find velocity.

Cartesian coordinate system

A coordinate system in which points are specified by position along two orthogonal axes.

component (vector)

The length of a vector projected onto two orthogonal axes. In other words, in the Cartesian coordinate system, the distance the vector head lies along both the x and y axes. Components are perpendicular.

displacement

The shortest distance between two points, independent of the path taken.[5] Displacement can be represented by a scalar in one dimension and a vector in two dimensions. Can also be described as a relative position, usually in relation to the world origin. The derivative of displacement versus time is velocity (in relation to the reference point). Likewise, displacement can be found by integrating velocity over time.

integrator

A routine or function that performs integration. In physical simulators, numerical integrators are typically used. No closed form solution is found - instead, values at discrete time steps are calculated using a variety of methods.[7]

operator overloading

A specific case of polymorphism[6] in which the behavior of operator characters, such as `*` or `+=`, is customized. Useful for classes that have methods that correspond to the basic scalar operations, like addition or subtraction. In this module, the Vector class uses operator overloading for vector addition, subtraction, scaling and assignment.

position

A location in a coordinate system.[8] In one dimension, position is given as a scalar, while in two dimensions, position can be specified using a vector.

speed

The magnitude of the velocity vector. Should not be confused with velocity, as speed does not include any direction information.

vector

A geometric object that has both a length and direction. Vectors are used extensively in 2 dimensions to represent any attribute that has a location, direction or magnitude. In this text, vectors are given in the form $\langle x, y \rangle$ where x and y are the vector components.

velocity

The rate of change of position. Velocity is represented by a scalar in one dimension and a vector in two dimensions. While velocity is analogous with speed in one dimension, in two dimensions speed is the magnitude of the velocity vector. Velocity over time can be integrated to find displacement, while integrating acceleration over time will give the velocity.

7 Code

rbpmath.py

```
import math

class Vector(object):
        # Initializes 2d vector.  Default is 0.
        def __init__(self, x=0, y=0):
                self.x = x
                self.y = y

        # Prints components of vector.
        def __str__(self):
                return "X: " + str(self.x) + " Y: " + str(self.y)

        # Checks to see if the components of two vectors are equal.
        def __eq__(self, other):
                if isinstance(other, Vector):
                        return ((self.x == other.x) and (self.y == other.y)
                else:
                        return NotImplemented
```

```
# Checks to see if the components are not equal.
def __ne__(self, other):
        eq = self.__eq__(other)
        if eq is NotImplemented:
                return not eq
        return NotImplemented

# Vector addition
def __add__(self, other):
        if isinstance(other, Vector):
                return Vector(self.x+other.x, self.y+other.y)
        else:
                return NotImplemented

# Vector addition and assignment
def __iadd__(self, other):
        if isinstance(other, Vector):
                self.x += other.x
                self.y += other.y
                return self
        else:
                return NotImplemented

# Vector subtraction
def __sub__(self, other):
        return Vector(self.x-other.x, self.y+other.y)

# Vector subtraction and assignment
def __isub__(self, other):
        if isinstance(other, Vector):
                self.x -= other.x
                self.y -= other.y
                return self
        else:
                return NotImplemented

# Scalar multiplication
def __mul__(self, other):
        if isinstance(other, Vector):
```

```
                        return (self.x*other.x + self.y*other.y)
                elif isinstance(other, int) or isinstance(other, float):
                        return Vector(self.x * other, self.y * other)
                else:
                        return NotImplemented

        # Scalar multiplication and assignment
        def __imul__(self, other):
                if isinstance(other, Vector):
                        self.x *= other
                        self.y *= other
                        return self
                else:
                        return NotImplemented

        # Cross product
        # In 2d it is a scalar
        def __mod__(self, other):
                if isinstance(other, Vector):
                        return (self.x*other.y - self.y*other.x)
                else:
                        return NotImplemented

        # Returns the magnitude of the vector.
        def magnitude(self):
                return math.sqrt(self.x*self.x + self.y*self.y)

        # Returns tuple of components for drawing functions
        def components(self):
                return (self.x, self.y)
```

rbpbodies.py

```
from tmath import Vector

# This is the basic superclass for all bodies
class Body(object):
        def __init__(self, position, velocity, acceleration):
                # Make sure position, velocity and acceleration are vectors
```

```
                if isinstance(position, Vector):
                        self.pos = position
                if isinstance(velocity, Vector):
                        self.vel = velocity
                if isinstance(acceleration, Vector):
                        self.acc = acceleration

        # Prints information about the body.
        def __str__(self):
                return  "Position: " + str(self.pos) + "\n"\
                                "Velocity: " + str(self.vel) + "\n"\
                                "Acceleration: " + str(self.acc) + "\n"
```

8 References

1. http://en.wikipedia.org/wiki/Vector
2. http://en.wikipedia.org/wiki/Vector_graphics
3. http://en.wikipedia.org/wiki/Euclidean_vector
4. http://docs.python.org/reference/datamodel.html
5. http://en.wikipedia.org/wiki/Displacement_(vector)
6. http://en.wikipedia.org/wiki/Operator_overloading
7. http://en.wikipedia.org/wiki/Integrator
8. http://en.wikipedia.org/wiki/Position