2019

05 Visualizing Projectile Motion

W.A. Morgan

L.Q. English

Visualizing Projectile Motion

In this activity, you will extend what you learned in Chapter 3 about the kinematics equations into two dimensions. These equations describe the motion of a projectile undergoing a constant acceleration.

Usually a projectile, such as a thrown ball or a launched rocket, is propelled at an angle to the ground θ, usually measured from the horizontal. One must take into account the *components* in the horizontal (usually "x") and vertical (usually "y") directions.

5.1 The Physics

The kinematic equations are as follows:

$$x = x_0 + v_{x0}t + \frac{1}{2}a_x t^2; \tag{5.1}$$

$$v_x = v_{x0} + a_x t; \tag{5.2}$$

and

$$v_x^2 = v_{x0}^2 + 2a_x(x - x_0). \tag{5.3}$$

While "x" is used above in Equations (5.1), (5.2), (5.3), they are simply placeholders; one can easily replace them with "y":

$$y = y_0 + v_{y0}t + \frac{1}{2}a_y t^2; \tag{5.4}$$

$$v_y = v_{y0} + a_y t; \tag{5.5}$$

and

$$v_t^2 = v_{y0}^2 + 2a_y(y - y_0). \tag{5.6}$$

Therefore if we have a projectile launched at an angle θ to the ground, then the horizontal component of the velocity is

$$v_x = v\cos\theta \tag{5.7}$$

and the vertical component of the velocity is

$$v_y = v\sin\theta. \tag{5.8}$$

So, if we throw a ball at angle $\theta = 43°$, with a velocity of 10 $\frac{m}{s}$, then the initial x- and y-components of the velocity are (cf. Equations (5.7) and (5.8))

$$v_{x0} = 10\frac{m}{s}\cos 43°$$

and

$$v_{y0} = 10\frac{m}{s}\sin 43°.$$

As for acceleration, if we are talking about locations near the surface of the Earth and ideal conditions (no atmosphere, so therefore no wind), there is no acceleration in the horizontal (x) direction. The only acceleration is the familiar $a_y = -g = -9.8\frac{m}{s}$.

5.2 Exercises

5.2.1 Projectile over Flat Ground

Use Equations (5.1), (5.2), (5.4), (5.5), (5.7), and (5.8) to write VPython code that simulates the flight of a projectile. Make the projectile a sphere of radius 0.1. Assume that there is no air resistance. Assume that we launch the projectile at the origin (0,0), and stop the projectile motion once it hits the ground at y = 0. This means that the terrain is flat. Draw a line (or horizontal plane) that indicates this.

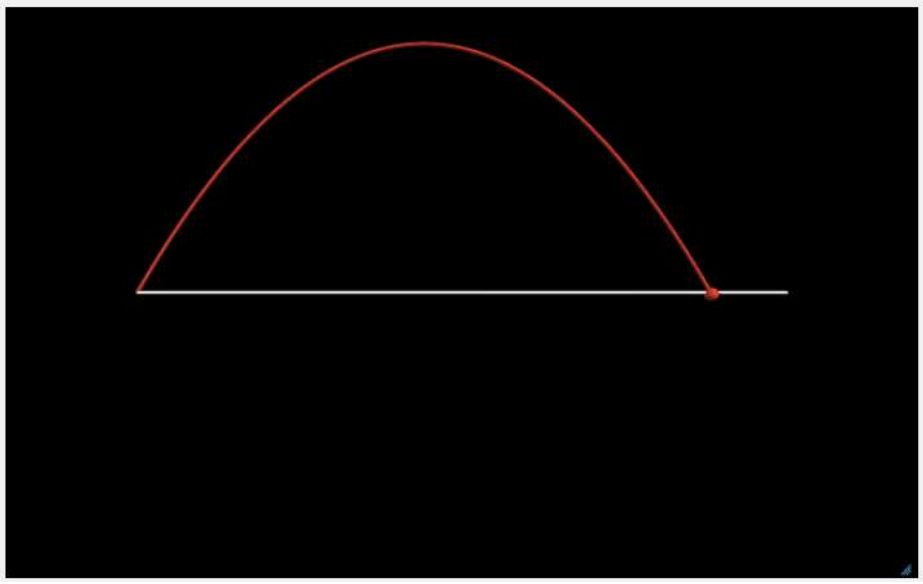

Figure 5.1: A Projectile Trajectory Over a Flat Slope

To accomplish this last condition, use the following `while` loop:

```
while(y>-0.001)
```

Your program should have an output such as depicted in Figure 5.1.

One thing to keep in mind is the built-in unit for angles in VPython is always **radians**. This means that you will need to convert any angle in the argument of a trig-function from degrees to radians.

Now play around with launch angle at constant launch speed. Make a table of the range for each launch angle. For what angle θ do we seem to get the largest range? Is this range consistent with the theoretical formula for the range of a projectile, given by

$$R = \frac{v_0^2 \sin 2\theta}{g}. \tag{5.9}$$

To answer this question, open up Excel, have it compute the theoretical value from Equation (5.9) at regular angle intervals, and then compare the values with your numerical results from VPython simulations.

5.2.2 Projectile over Sloping Ground

Now let's make the ground not level (or flat). Let's have a steady downward slope. Redraw the line (or plane) so that it tilts down with a slope of rise/run = -0.1.

The way we can simulate that in the code is by modifying the WHILE statement. How do you have to do that? Once your group finds a solution, implement it in the code.

What angle now produces the largest range?

Repeat the simulation, but with a terrain that gently slopes up, with a slope of +0.1. What is the best angle now?

5.2.3 Projectile Striking a Target

In addition to the projectile and the (flat) ground, draw in a stationary sphere of radius 0.5. You decide the coordinates of this stationary sphere.

Make the code stop if or when the projectile hits the object, which we will define as anywhere within the volume of the object. If we do not hit the sphere, then make the code stop as before when the ground it hit. Because we are only thinking in two dimensions, in order to hit the sphere we need to just have the projectile get within 0.5 units of the center. How can we do that?

Hint: there is a kind of brute-force approach using the Pythagorean theorem, but there is also a much more elegant method using the two position vectors associated with the projectile and the target sphere. Can you see how you could use the difference of these two vectors?

One thing that will probably need here is the following command:

BREAK

If this command is encountered within a `WHILE`-loop, we immediately leave the loop and continue with statements outside of it. Oftentimes, you will find the `break` command wrapped inside an `IF`-statement all inside a `while`-loop.

Projectile motion is a fairly easy thing to simulate in VPython. In the next chapter, you will see how to make the simulation more realistic by the incorporation of air resistance in the code, with the use of vectors.

5.2.4 Mystery Code

Scan the following VPython code and annotate each line. Discuss what you think this code will do. Make a prediction as to the kind of motion that would be highlighted.

Note that in the VPython environment, to raise a quantity to some power, you do NOT use the carrot-symbol, but instead you use two star-symbols "$**$". Thus, x^2 would be represented in the code as $x**2$.

```
GlowScript 2.7 VPython

graph(width=400, height=250)
xDots=gdots(color=color.green)
redbox=box(pos=vector(4,-1,3),
           size=vector(8,1,6),color=color.red)
ball=sphere(pos=vector(4,50,3),radius=2,color=color.green)

t=0
y0=50
v0=0
t0=0
i=0
while True:
    rate(200)
    t=t+0.01
    y=y0-0.5*9.8*t**2+v0*t
    if(y<0):
        i+=1
        t=0
        y0=0
        v0=20*0.9**i
    ball.pos=vec(4,y,3)
```