

Modeling motion with VPython

Every program that models the motion of physical objects has two main parts:

1. **Before the loop:** The first part of the program tells the computer to:
 - a. Create numerical values for constants we might need
 - b. Create 3D objects
 - c. Give them initial positions and momenta
2. **The “while” loop:** The second part of the program, the loop, contains the statements that the computer reads to tell it how to increment the position of the objects over and over again, making them move on screen. These statements tell the computer:
 - a. How to calculate the net force on the objects
 - b. How to calculate the new momentum of each object, using the net force and the momentum principle
 - c. How to find the new positions of the objects, using the momenta

We will now introduce how to model motion with a computer. The program you will write will model the motion of a “fan cart” on a track. A small electric fan mounted on a cart pushes on the air with a constant force, so the air exerts a constant force on the fan blades.

1. Before the loop

Creating the objects

- Open IDLE for Python, and type the usual beginning statement:

```
from visual import *
```

First we’ll create a box object to represent the track that is one meter long.

- On the next line type the following:

```
track = box(pos=vector(0,-.05, 0), size=(1.0, 0.05, .10))
```

The **pos** attribute of a box object is the position of the *center* of the box. The **size** attribute gives the box’s length in the x, y and z directions.

To represent the fan cart, we will also use a box object. We will place the cart at the left end of the track to start.

- On the next line type the following:

```
cart = box(pos=vector(-0.5,0,0), size=(0.1, 0.04, 0.06), color=color.green)
```

Run the program to see the track and “cart”. If you like, you can bring in the “coordinate axes” boilerplate from the course website Files page. If you do so, you’ll need to adjust the length of the axes to something appropriate.

Initial conditions

Any object that moves needs two vector quantities declared before the loop begins:

1. initial position; and
2. initial momentum.

We’ve already given the cart an initial position of $\langle -0.5, 0, 5.0 \rangle$ m. Now we need to give it an initial momentum. If you push the cart with your hand, the initial momentum is the momentum of the cart just *after* it leaves your hand. Since the definition of momentum at speeds much less than the speed of light, is $\vec{p} \approx m\vec{v}$, we need to tell the computer the cart’s mass and the cart’s initial velocity.

- On a new line type the following:

```
cart.m = 0.80
```

We have made up a new attribute named “m” for the object named “cart”. The symbol **cart.m** now stands for the value 0.80 (a scalar), which represents the mass of the cart in kilograms. Now that we have the mass, multiplying the mass by the initial velocity will give the initial momentum. Let’s give the cart an initial velocity of $\langle 0.5, 0, 0 \rangle$ m/s. Then, the initial momentum would be the mass times this initial velocity vector.

- On a new line type the following:

```
cart.p = cart.m*vector(0.5, 0, 0)
```

This gives the cart an initial momentum value that is $(0.80 \text{ kg}) \langle 0.5, 0, 0 \rangle \text{ m/s} = \langle 0.4, 0, 0 \rangle \text{ kg m/s}$. The symbol **cart.p** will stand for the momentum of the cart (a vector) throughout the program.

Note: There are no “built in” physics attributes **p** or **m** for objects like there are built-in geometrical attributes **pos** or **radius**. We can make any attributes we want. We could have called the momentum just **p**, or the mass just **m**, instead of **cart.p** or **cart.m**. It’s useful to create attributes like mass or momentum associated with objects so we can easily tell apart the masses and momenta of different objects.

Time step and time

To make the cart move we will use the equation $\vec{r}_f = \vec{r}_i + \vec{v}_{\text{avg}} \Delta t$ repeatedly in a loop. We need to define a variable **deltat** to stand for the time step Δt . Here we will use the value $\Delta t = 0.01$ s.

- On a new line type the following:

```
deltat = 0.01
```

Also set a variable **t**, which stands for cumulative time, to start at zero seconds. Each time through the program loop we will write, we will increment the value of **t** by **deltat** (in this case, 0.01 seconds).

- On a new line type the following:

```
t = 0
```

Think about the situation and discuss the following questions with your lab partner:

- How far will the cart move in one time step (one execution of the loop)?
- How many executions of the loop will it take for the cart to move one meter?
- What will the time **t** be after moving one meter?

This completes the first part of the program, which tells the computer to:

- a. Create numerical values for constants we might need
- b. Create 3D objects
- c. Give them initial positions and momenta

2. The “while” loop

Constant momentum motion

Somebody or something gave the cart some initial momentum. We’re not concerned here with how it got that initial momentum. We’ll predict how the cart will move in the future, *after* it acquired its initial momentum.

We will create a “while” loop. Each time the program runs through this loop, it will do two things:

1. Use the cart’s current momentum to calculate the cart’s new position
2. Increment the cumulative time **t** by **deltat**

- On a new line type the following:

```
while t<3.0:
```

- Make sure the statement ends with “:”, then press Enter. Note that the cursor is now indented on the next line. If it is not, place the cursor just after the “:” and press Enter.

This tells the computer to keep executing the loop while the cumulative time is less than 3 seconds.

You know that the new position of an object after a time interval Δt is given by

$$\vec{r}_f = \vec{r}_i + \vec{v}_{\text{avg}} \Delta t$$

where \vec{r}_f is the final position of the object, and \vec{r}_i is its initial position. If the time interval Δt is very short, so the velocity doesn’t change very much, we can use the initial or final velocity to approximate the average velocity.

Since at low speed $p \approx mv$, or $v \approx p/m$, we can write

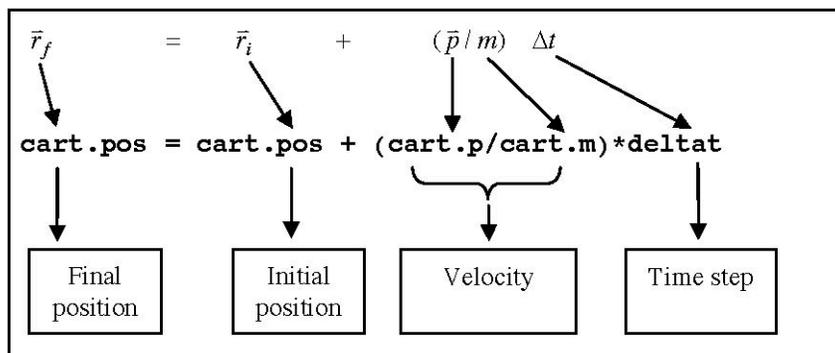
$$\vec{r}_f = \vec{r}_i + \left(\frac{\vec{p}}{m} \right) \Delta t$$

We will use this equation to increment the position of the cart in the program. First, we must translate it so *VPython* can understand it.

- On the indented line after the “while” statement, type the following:

```
    cart.pos = cart.pos + (cart.p/cart.m)*deltat
```

Notice how this statement corresponds to the algebraic equation:



In the program, the final and initial position is written the same way: `cart.pos`. This isn't really an equality, but an assignment statement; `cart.pos` is being assigned a new value equal to its old value plus `(cart.p/cart.m)*deltat`.

Finally we need to increment the cumulative time `t`.

- On the indented line after the “**while**” statement, type the following:

```
t = t + deltat
```

- Now, run the program.

Slowing down the animation

When you run the program, you should see the cart at its final point. (You may need to zoom out to see it.) The program is executed so rapidly that the entire motion occurs faster than we can see. We can slow down the animation rate by adding a “rate” statement.

- Place the cursor at the end of the statement that reads “while t<3.0:”. Press Enter to insert a new, indented statement right before the “cart.pos” statement.
- On this new line, type the following:

```
rate(100)
```

Every time the computer executes the loop, when it reads “**rate(100)**”, it pauses long enough to ensure the loop will take 1/100th of a second. Therefore, the computer will only execute the loop 100 times per second.

- Now, run the program.

You should see the cart travel to the right at a constant velocity, stopping beyond the edge of the track.

Note: The cart going beyond the edge of the track isn't a good simulation of what really happens, but it's what we told the computer to do. There are no “built-in” physical behaviors, like gravitational force, in VPython. Right now, all we've done is tell the program to make the cart move in a straight line. If we wanted the cart to fall off the edge, we would have to enter statements into the program to tell the computer how to do this.

**CHECKPOINT 1: Ask an instructor to check your work for credit.
You can read ahead while you're waiting to be checked off.**

3. Force and momentum principle

Now we will modify our program to include a force on the cart. This means we will have to add statements to the program that tell the computer that there is a force on the cart, and tell it how to use the momentum principle to change the cart's momentum due to the force.

Suppose you push a cart, and after leaving your hand it has an initial momentum to the right, as before. But now, the fan on the cart is opposing this initial momentum, so that there is a net force on the cart to the left. This force is about 0.4 newtons. Let's use our program to model this situation. First let's add a statement that creates a force vector.

- In the loop, insert a new statement right after the “**rate(100)**” statement. Type the following:

```
Fnet = vector(-0.4, 0, 0)
```

Note: This force has a constant (vector) value; it doesn't change each time through the loop, unlike the cart's position. We could have, therefore, written this statement before the loop. But in most of the programs we write, force will be changing with time, so force calculations should go *inside* the loop.

Next, we need to use the momentum principle. The momentum principle says the change in an object's momentum in a short time period is equal to the net force on it, times the time interval (this assumes that the force doesn't change much during the short time interval). In symbols, this is:

$$\Delta\vec{p} = \vec{p}_f - \vec{p}_i = \vec{F}_{\text{net}}\Delta t$$

Where \vec{p} is the cart's momentum, and \vec{F}_{net} is the net force on the cart. We can rewrite this as in the update form:

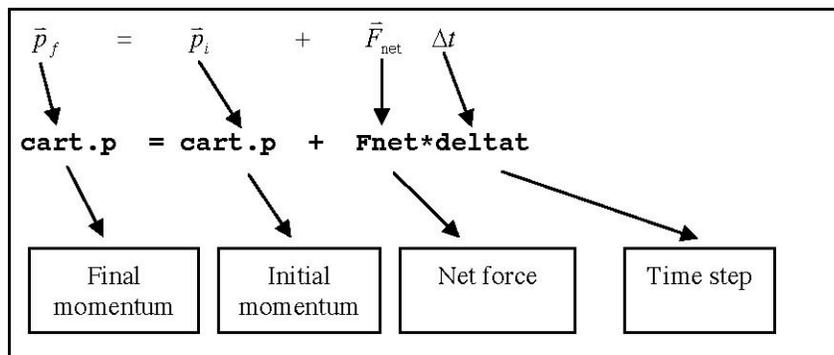
$$\vec{p}_f = \vec{p}_i + \vec{F}_{\text{net}}\Delta t$$

We then use this equation in the program, which tells the computer how to calculate a new momentum for the cart.

- Insert a new statement in the loop, right after the line that reads `"Fnet = vector(-0.4, 0, 0)"`.
- On this new line, type the following:

```
cart.p = cart.p + Fnet*deltat
```

Note the similarity between this statement and the statement that updates the position. Here is how the algebraic equation corresponds to the VPython code:



- Run the program. You should see the cart move to the right a small distance before changing direction and moving to the left.
- Change the initial momentum of the cart so that it moves a greater distance to the right before changing direction: make the initial *speed* of the cart 0.9 m/s, with direction to the right.
- Run the program. You should now see the cart move much closer to the right end of the track before it starts moving to the left.

4. Making graphs of the motion

At the start of your program, add this statement to import the graphing module:

```
from visual.graph import *
```

Before the loop, add the following statement, to create a graphing curve object:

```
mgraph = gcurve(color=color.cyan)
```

Inside the loop, after updating the momentum and position of the cart, and the time, add the following statement:

```
mgraph.plot( pos=(t, cart.p.x) )
```

As you might expect, this plots a point on the graph at a location given by t on the horizontal axis, and the x component of the momentum of the cart on the vertical axis.

Prevent the two windows from overlapping by placing the following statement just after the initial statement “`from visual import *`”: (Note: this step does not yet work on Macs. In class you’ll just need to quickly drag the graph window down out of the way so you can see the cart moving.)

```
scene.y = 400 # move animation window down 400 pixels from top of screen
```

After examining the plot of the x component of the cart’s momentum, change the plot statement to plot the x component of the position of the cart (`cart.pos.x`), then change back to plotting `cart.p.x`.

**CHECKPOINT 2: Ask an instructor to check your work for credit.
You can read ahead while you’re waiting to be checked off.**

5. Motion in two dimensions

Let’s try giving the cart a component of initial momentum in the y -direction (that is, upward). Let’s assume that we are in outer space, where gravitational forces on the cart are negligible. We want to explore a situation where the motion is two-dimensional.

- Change the initial velocity of the cart to $\langle 0.7, 0.1, 0 \rangle$ m/s and run the program.
- Add a trail to the cart, to see its trajectory clearly. To do this, create a curve object before the while loop; for example, `cart.trail = curve(color=color.red)`. Inside the loop, append the moving object’s position to this curve object every time you go through the loop, with `cart.trail.append(pos=cart.pos)`.

The cart now makes a path in the shape of a parabola. Let’s look more carefully at the momentum of the cart during its motion by printing out the momentum vector at each execution of the loop.

- Insert a new statement in the loop, just after the line that reads “`cart.p = cart.p+Fnet*deltat`”. Type the following on this new line:

```
print "t=", t, "cart.p=", cart.p
```

Run the program. The values of time t and momentum `cart.p` are now printed in the text output window at every execution of the loop. Refer to these values when doing the following exercises.

In your notebook, write the answers to the following questions. Your instructor will ask you about them.

- (1) Using the printed list of momentum vectors, calculate $\Delta\vec{p}$. That is, pick one of the vectors to be the final momentum, and subtract from it the vector just above it, which is the initial momentum:
$$\Delta\vec{p} = \vec{p}_f - \vec{p}_i = \langle ?, ?, ? \rangle \frac{\text{kg}\cdot\text{m}}{\text{s}}$$
- (2) Calculate the vector $\vec{F}_{\text{net}}\Delta t$. (In the program, this would be `Fnet*deltat`.)
$$\vec{F}_{\text{net}}\Delta t = \langle ?, ?, ? \rangle \text{N}\cdot\text{s}$$
- (3) Do your calculations show that the change in momentum $\Delta\vec{p}$ is equal to $\vec{F}_{\text{net}}\Delta t$?

- (4) Look at the printed list of momentum vectors again. Why aren't the y or z components of the momentum changing?

CHECKPOINT 3: Ask an instructor to check your work for credit.
You can read ahead while you're waiting to be checked off.

After being checked off, you can submit your final program to WebAssign to be graded. Check carefully in WebAssign to see what is asked for in the submitted program.

6. Playing around

What would you need to do to include the effects of gravity? The gravitational force near the Earth is $\langle 0, -mg, 0 \rangle$, where $g = +9.8$ N/kg. Turn on gravity and turn off the fan force, use a small Δt of 0.001 for accuracy, and use an if statement to make the cart bounce on the track, as you did to make a ball bounce off the walls of a box. Give the cart an initial velocity of $\langle 0.7, 3.0, 0 \rangle$ m/s.

Eventually you'll see the cart bouncing off nothing! Can you think how to change your program to do something more realistic? Hint: You can combine logical tests by using the keywords "and" and "or". For example, the logical expression " $(y < 3) \text{ and } (z > 2)$ " is true only if both of these conditions are true.