Models of Motion: Iterative Calculations - Fancart

OBJECTIVES

In this activity you will learn how to:

- Create 3D box objects
- Update the momentum and position of an object iteratively (repeatedly) to animate its motion

TIME

You should plan to finish this activity in 50 minutes or less.

COMPUTER PROGRAM ORGANIZATION

- A computer program consists of a sequence of instructions.
- The computer carries out the instruction one by one, in the order in which they appear, and stops when it reaches the end.
- Each instruction must be entered exactly correctly (as if it were an instruction to your calculator).
- If the computer encounters an error in an instruction (such as a typing error), it will stop running and print a red error message.

A typical program has four sections:

- Setup statements
- Definitions of constants (if needed)
- Creation of objects and specification of initial conditions
- Calculations to predict motion or move objects (done repetitively in a loop)

1 Setup statements

Using VIDLE for VPython, create a new file and save it to your own space. Make sure to add “.py” to the file name.

- Enter the following statement in the editor window:

```python
from visual import *
```

Every VPython program begins with this setup statement. It tells the program to use the 3D module (called “visual”). The asterisk means “Add all of the features available in the visual module to Python”.

** If you are using a version of VPython earlier that 3.0, you should place the following statement as the first statement in your program, before the import of visual:

```python
from __future__ import division, print_function
```

(from space underscore underscore future underscore underscore space import space division, print_function)
2  Constants

Following the setup section of the program you would define physics constants. We will talk about this in later projects.

3  Creating an object

- Create a box object to represent a track:

  \[
  \text{track} = \text{box} (\text{pos} = \text{vector}(0.95, -0.025, 0), \text{size} = (2.0, 0.05, 0.10), \text{color} = \text{color}.\text{white})
  \]

Run the program by pressing F5 (this may be fn-F5 on a Macintosh, depending on how you have set your preferences).

Arrange your windows so the Python Shell window is always visible.

Kill the program by closing the graphic display window.

- Create a second box object to represent a cart:

  Name this object “cart”, with some color other than white. Give this object a position (pos) of vector (0.95, 0.02, 0) and a size of (0.1, 0.04, 0.06).

Run the program by pressing F5. Zoom (both mouse buttons down; hold down Options key on Macintosh) and rotate (right mouse button down; hold down Apple Command key on Macintosh) to examine the scene. The cart should be sitting just above the track. Is it? If you don’t see two objects, you skipped something.

Reposition the cart so its left end is aligned with the left end of the track.

To do this you will have to answer the following questions on WebAssign:

Where is the “pos” of a box object? The left end? The right end? The center?
Do the numbers in the “size” of a box refer to the total length, or the distance from the center to one edge?
You can answer these by experimentation, or by looking in the online reference manual (Help menu, choose Visual).

3.1  Initial conditions

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

1. Initial position; and
2. Initial momentum.

You have already given the cart an initial position at the left end of the track. Now you 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. At speeds much less than the speed of light the momentum is \( \vec{p} \approx m\vec{v} \), and we need to tell the computer the cart’s mass and the cart’s initial velocity.

- Below the existing lines of code, type the following new lines:

```python
mcart = 0.80
pcart = mcart*vector(0.5, 0, 0)
print("cart momentum =", pcart)
```

If using a version before VPython 3.0 the print statement should look like:

```python
print "cart momentum=", pcart
```

We have made up a new variable name “mcart.” The symbol “mcart” now stands for the value 0.80 (a scalar), which represents the mass of the cart in kilograms. We have also created a new variable “pcart” to represent the momentum of the cart. We assigned it the initial value of \((0.80 \text{ kg})\langle0.5,0,0\rangle\text{ m/s}\).

- Run the program. Look at the Python Shell window. Is the correct value of the vector pcart printed there? From what is printed, how can you tell it is a vector? Answer on WebAssign.

3.2 Time step and total elapsed time

To make the cart move we will use the position update equation \( \vec{r}_f = \vec{r}_i + \vec{v}\Delta t \) repeatedly in a “loop”. We need to define a variable `deltat` to stand for a time step \( \Delta t \), and a variable `t` to stand for a total time elapsed since the motion started. Here we will use the value \( \Delta t = 0.01 \text{ s} \).

- Type the following new lines at the end of your program:

```python
deltat = 0.01
t = 0
```

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

1. Create numerical values for constants we might need (none were needed this time)
2. Create 3D objects
3. Give them initial positions and momenta

4 Beginning Loops

In a computer program a sequence of instructions that are to be repeated is called a loop. The kind of loop we will use in VPython is called a “while” statement. Instructions inside the loop are indented. IDLE will indent automatically after you type a colon.

For help with loops watch VPython Instructional Videos: 3. Beginning Loops

http://www.youtube.com/VPythonVideos
• To write a simple loop, type the following new lines at the end of your program.
• Be sure to type a colon (:) at the end of the while statement.
• Make sure the indenting is correct, as show below, then run:

```python
while t < 0.2:
    print("the time is now", t)
    t = t + deltat
print("End of the loop")
```

The statement:

```python
t = t + deltat
```

may look like a mathematical error. However, in a program, the “=” sign has a different meaning than in a mathematical equation.
The right hand side of the statement tells Python to read up the old value of t, and add the value of deltat to it.
The left hand side of the statement tells Python to store this new value into the variable t.

• Run the program. Look at the Python Shell window

Look at the printed output in the Shell window. Answer the following questions on WebAssign:
What makes the loop stop? Why is the first printed time 0? Why is the last time 0.19 and not 0.2? How can you get the program to print values from 0 through 0.3? (Try it.)

5 Loops and Animation

5.1 Cart with constant momentum

Consider a cart moving with constant momentum. 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.

You will use your iterative calculation “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 \( \text{deltat} \)

You know that the new position of an object after a time interval \( \Delta t \) is given by

\[
\vec{r}_f = \vec{r}_i + \vec{v}_{avg}\Delta t
\]

Where \( \vec{r}_f \) is the final position of the object, and \( \vec{r}_i \) is its initial position. If the time interval \( \Delta t \) 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 $\vec{p} \approx m\vec{v}$, or $\vec{v} \approx \vec{p}/m$, we can write

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

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.

- Delete or comment out the line inside your loop that prints the value of $t$.
- On the indented line after the “while” statement, and before the statement updating $t$, type the following:

$$\text{cart.pos} = \text{cart.pos} + (\text{pcart}/\text{mcart}) \times \text{deltat}$$

Notice how this statement corresponds to the algebraic equation:

Think about the situation and answer the following question on WebAssign:

What will the elapsed time $t$ be after moving two meters?

- Change the while statement so the program runs just long enough for the cart to travel 2 meters.
- Now, run the program. What do you see? Answer on WebAssign.

*Slowing down the animation*

When you run the program, you should see the cart at its final point. The program is executed so rapidly that the entire motion occurs faster than we can see, because a “virtual time” in the program elapses much faster than real time does. We can slow down the animation rate by adding a “rate” statement.

- Add the following line inside your loop (indented):

```python
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. Since your time interval $\text{deltat}$ is 0.01 seconds, and your rate statement forces each iteration to take 0.01 seconds, the cart will move in “real” time.
• **Now run the program.**

You should see the cart travel to the right at a constant velocity, ending up 2 meters from its starting location.

**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 told the computer 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.

**Answer the following question on WebAssign:**

Which statement in your program represents the position update formula?

### 5.2 Changing momentum

Your running program should now have a model of a cart moving at constant velocity from left to right along the track.

• What should happen to the motion of the cart if you apply a constant force to the left? Discuss this among your group and write down your prediction on WebAssign.

As discussed in the textbook, an iterative prediction of motion can include the effects of forces that change the momentum of an object:

- Calculate the (vector) forces acting on the system.
- Update the momentum of the system, using the Momentum Principle: \( \vec{p}_f = \vec{p}_i + \vec{F}_{\text{net}} \Delta t \).
- Update the position: \( \vec{r}_f = \vec{r}_i + \vec{v} \Delta t \).
- Repeat.

Here is how the Momentum Principle can be translated into VPython:

- **Inside the loop, create a new vector variable named** \( \vec{F}_{\text{air}} \), and assign it the value \((-0.1, 0, 0)\), using the appropriate VPython syntax. (Look at how you created a variable to represent the momentum vector.)
- After calculating the net force, use the Momentum Principle to update the momentum.
- After updating the momentum, use the new momentum to update the position.
Allow the cart to move for 10 seconds in order to explore the following question on WebAssign:
Should the Momentum Principle statement be placed before the loop or inside the loop?

How should you write this statement in order to use the force $F_{\text{air}}$ (the constant force by the air on the fancart) as the net force in this statement? Do this, updating momentum before updating position. Does your program work as you predicted it should? If not, fix it (you may wish to discuss your approach with another group.)

- What should happen to the motion of the cart if you apply a constant force to the right? Discuss this among your group and write down your prediction on WebAssign. Then create a VPython program to model this motion.

5.3 2D motion

In a computer program you can model behavior that would be difficult to observe in the real world.

Do the following:
Change the initial momentum of the fancart so that it includes a $+y$ component similar in magnitude to the $x$ component of the momentum. What happens? Explain this, then compare your explanation to that of a neighboring group.

6 Making graphs in VPython

To show you understand VPython and the physics of this situation, construct a graph of $x$-position vs. time and a graph of velocity vs. time. Be sure to include these graphs in your submitted program.

Here are some VPython tips:

Place: from visual.graph import * underneath “from visual import *”

Place: myDisplay = gdisplay()
  myGraph = gcurve(color=color.???) above the loop

Place: myGraph.plot(pos=(t,cart.x)) above $t = t + \Delta t$.

Do this for each graph. (If you want multiple plots on the same display, only declare gdisplay() once. For more help see the VPython documentation.)

To have a better view of the graphs (instead of dragging them around every time), modify the gdisplay() code to properly locate the position and velocity graphs, here are example codes:

\begin{verbatim}
gdisplay(x=scene.width)
gdisplay(y=scene.height)
\end{verbatim}
7  Turn in your program to WebAssign

Make sure everyone in the group agrees that the program is correct. Check with a neighboring group.

When you turn in a program to WebAssign, be sure to follow the instructions given there, which may sometimes ask you to change some of the parameters in your program.