Computer Models of Motion: Iterative Calculations

**OBJECTIVES**

In this activity you will learn how to:

• Create 3D box objects

• Update the position of an object iteratively (repeatedly) to animate its motion

• Update the momentum and position of an object iteratively (repeatedly) to predict its motion

**TIME**

You should plan to finish this activity in 65 minutes or less.

**COMPUTER PROGRAM ORGANIZATION**

A computer program consists of a sequence of instructions.

The computer carries out the instructions 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:**

**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 to Python all of the features available in the visual module''.

Both Python and VPython are undergoing continuous improvement. For example, before Python version 3.0, 1/2 was truncated to zero, but beginning with Python 3.0, 1/2 means 0.5. If you are using a version of Python earlier than 3.0, you should place the following statement as the first statement in your program, before the import of visual:

**from \_\_future\_\_ import division**

This statement (from *space* **underscore underscore** future **underscore underscore** *space* division) tells the Python language to treat 1/2 as 0.5. You don't need this statement if you are using Python 3.0 or later, but it doesn't hurt, because it is simply ignored by later versions of Python.

**2** **Constants**

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

**3** **Creating an object**

 **Create a box object to represent a track:**

**track = box(pos=vector(0, -0.025, 0), size=(2.0, 0.05, 0.10), color=color.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, 0.2, 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 floating 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:**

1. **Where is the ``pos'' of a box object? The left end? The right end? The center?**
2. **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've 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 , 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:**

**mcart = 0.80**

**pcart = mcart\*vector(0.5, 0, 0)**

**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 kg) 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?**

**3.2** **Time step and total elapsed time**

To make the cart move we will use the position update equation  repeatedly in a ``loop''. We need to define a variable **deltat** to stand for the time step , and a variable **t** to stand for the total time elapsed since the motion started. Here we will use the value  s.

• **Type the following new lines at the end of your program:**

**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**

Watch **VPython Instructional Videos: 3. Beginning Loops**

http://www.youtube.com/VPythonVideos

Complete the following task:

• **Create a loop that prints the variable t from 0.0 to 0.19 in increments of deltat (section 3.2)**

• **If you're stuck, watch the video again.**

• **Add a print command after the loop to print the text `End of the loop.'**

• **Run the program. Look at the Python Shell window.**

**5** **Loops and Animation**

Watch **VPython Instructional Videos: 4. Loops and Animation**

<http://www.youtube.com/VPythonVideos> to see how using loops can animate 3D objects. We will use loops and physics principles to build models of motion that are consistent with the natural world. The next section introduces how to translate physics principles into syntax understood by the computer program.

**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 **deltat**

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



where  is the final position of the object, and  is its initial position. If the time interval  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 , or , we can write



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:**

**cart.pos = cart.pos + (pcart/mcart)\*deltat**

Notice how this statement corresponds to the algebraic equation:



 **Think about the situation and answer the following question:**

**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?**

***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):**

**rate(100)**

Every time the computer executes the loop, when it reads ``**rate(100)**'', it pauses long enough to ensure the loop will take  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, 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:**

**Which statement in your program represents the position update formula?**

C**hange your program so the cart starts at the right end of the track and moves to the left.**

When you have succeeded, compare your program to that of another group.

**5.2** **Changing momentum**

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

• **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.**

As discussed in Class, 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: .

 • Update the position: .

 • Repeat.

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



• **Inside the loop, create a new vector variable named F\_air, and assign it the value** **, 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.**

By experimentation, determine a value for F\_air, the force by the air on the fancart, that produces the following motion:

**The cart starts at the right end of the track and moves to the left, gradually slowing down, and coming to a stop near the left end of the track; it turns around and moves to the right, speeding up.** (Note that you need to let the loop run longer in order to see this behavior.)

**6** **Final Challenge**

Explore changing something about the fancart program to achieve some new kind of motion. So far in this program, we've been restricting the cart's motion to 1D. Can you think of something else to try?

 **CHECKPOINT: Ask the Instructor to look over your program and visual output.**

**7** **Document your work.**

Post screen shots of your code and the resulting display windows to your blog for your cart moving back and forth. Investigate using camstudio or other screen capture tools to create a video of the animation of your cart for your blog.