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 position and velocity of an object iteratively (repeatedly) to predict its motion

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:

1. Setup statements
2. Definitions of constants (if needed)
3. Creation of objects and specification of initial conditions
4. 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 hard drive. Make sure to add ".py" to the file name.

Enter the following statement in the VIDLE editor window:

```python
from visual import *
```

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

3a. Creating objects
Create a box object to represent the track:

```python
track = box(pos=vector(0,-.05, 0), size=(2.0, 0.05, .10), color=color.white)
```

Run the program by pressing F5. Arrange your windows so the VPython Shell window is always visible. Kill the program by closing the graphic display window.

Now create a second box object, named "cart", with some color other than white. Give this object a position (pos) of (0, 0, 0) and a size of (0.1, 0.04, 0.06).

Run the program by pressing F5. Zoom (both mouse buttons down) and rotate (right mouse button down) to examine the scene. The cart should be floating just above the track. Is it?

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:
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 (Drop-down Help menu, choose VPython).

3b. Initial conditions
Any object that moves needs two vector quantities declared before the loop begins:
1. initial position; and
2. initial velocity.

You’ve already given the cart an initial position at the left end of the track. Now you need to give it an initial velocity. If you push the cart with your hand, the initial velocity is the velocity of the cart just after it leaves your hand.

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

```python
mcart = 0.80
vcart = vector(0.5, 0, 0)
print('cart velocity =', vcart)
```

We have made up a new variable named “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 vector variable vcart, which is the velocity of the cart. We assigned it the initial value of (0.5,0,0) m/s.

- Run the program. Look at the VPython Shell window. Is the correct value of the vector vcart printed there? From what is printed, how can you tell it is a vector?

Note: There are no “built in” physics attributes v or m for objects like there are built-in geometrical attributes pos or radius. However, VPython allows us to create new attributes for objects. We could have called the velocity cart.v, or the mass cart.m, instead of vcart or mcart. It can be helpful to create attributes like mass or velocity associated with objects so we can easily tell apart the masses and velocities of different objects in a complex program.

3c. Time step and total elapsed time
To make the cart move we will use the position update equation \( \mathbf{r}_f = \mathbf{r}_i + \mathbf{v} \Delta t \) repeatedly in a “loop”. We need to define a variable deltat to stand for the time step \( \Delta t \), and a variable t to stand for the total time elapsed since the motion started. Here we will use the value \( \Delta t = 0.01 \) 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:
a. Create numerical values for constants we might need (none were needed this time)
b. Create 3D objects
c. Give them initial positions and velocities
4. Repeated calculations: 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 starts with a "while" statement. Instructions inside the loop are indented. VIDLE will indent automatically after you type a colon.

- 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 shown below, then run:

```python
while t < 0.2:
    print('the time is now',t)
    t = t + deltat
print('after 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 VPython to read up the old value of t, and add the value of deltat to it. The left hand side of the statement tells VPython to store this new value into the variable t.

- Run the program. Look at the VPython Shell window.

Look at the printed output in the Shell window. Answer the following questions:
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.)

4a. Constant velocity motion
Consider a cart moving with constant velocity. Somebody or something gave the cart some initial velocity. We’re not concerned here with how it got that initial velocity. We’ll predict how the cart will move in the future, after it acquired its initial velocity.

You will use your iterative computational “loop”. Each time the program runs through this loop, it will do two things:
1. Use the cart’s current velocity 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 Δt is given by

\[ r_f = r_i + v \Delta t, \]  

where \( r_f \) is the final position of the object, and \( r_i \) is its initial position.

If the time interval \( \Delta t \) is very short so that the velocity doesn’t change very much, we can use the initial or final velocity to approximate the average velocity.

We can write \( r_f = r_i + v \Delta t \). 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:

\[ \text{cart.pos} = \text{cart.pos} + v_{\text{cart}} \times \text{deltat} \]

Notice how this statement corresponds to the algebraic equation:

\[ r_f = r_i + v \times \Delta t \]

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):

\[ \text{rate(100)} \]

Every time the computer executes the loop, when it reads “rate(100)”, it pauses long enough to ensure the loop will take 1/100\(^{th}\) 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 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.

Answer the following questions:
1. Which statement in your program represents the position update formula?
2. What would you have to change in your program to make the cart start at the right end of the track and move to the left? Do this.

4b. Changing velocity

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 it is accelerating to the right?
An iterative prediction of motion can include the effects of forces (or accelerations) that change the velocity of an object. Here is a translation into VPython:

\[ v_f = v_i + a \Delta t \]

\[ v_{\text{cart}} = v_{\text{cart}} + a \ast \text{deltat} \]

Final velocity  Initial velocity  Acceleration  Time step

- Before the loop, create a new vector variable named \( Fair \), and assign it the value \(< 0.3, 0, 0 >\), using the appropriate VPython syntax (look at how you created a variable to represent the velocity vector).
- Increase the time the loop runs, allowing the cart to move for 20 seconds, in order to explore the following questions:

**Answer the following questions:**

1. Should the acceleration statement be placed before the loop or inside the loop?  
2. How should you write this statement in order to use the acceleration?  Do this.  Does your program work as you predicted it should?  If not, fix it (you may wish to discuss your approach with another group.)  
3. By experimentation, determine a value for the acceleration 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.

**4c. 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 velocity of the cart so that it includes a +y component similar in magnitude to the \( x \) component of the velocity.  What happens?  Explain this.

**5. Playing around (optional)**

What would you need to do to include the effects of gravity?  The gravitational acceleration near the Earth is \((0, -g, 0)\), where \( g = +9.8 \text{ m/s}^2 \).  Give the cart an initial velocity of \((0.7, 3.0, 0) \text{ m/s}\), give the acceleration due to gravity the vector value \((0, -g, 0)\).  Turn on gravity, turn off the horizontal acceleration, use a smaller \( \text{deltat} \) of 0.005 for accuracy, and use an “if” statement to make the cart bounce on the track:

\[
\text{if cart.pos.y} < 0:
    \text{vcart.y} = \text{abs(vcart.y)}  \#\text{ make the cart move upward}
\]

Eventually you will 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.