Computational Activity 4: Space, the Final Frontier

Derived from activites VP04 and VP05 in the instructor resources that accompany Ruth Chabay and Bruce Sherwood's Matter and Interactions *textbook.*

In this activity, you will simulate motion with gravitational forces, eventually building up a program that simulates the elliptical orbit of a spacecraft. You have previously written down a formula for the gravitational potential energy, $V_g = GmM/r$. You have also seen that the gravitational force is opposite to the derivative of potential energy. Extending that idea to three dimensions, we obtain that the gravitational force is

$$\vec{F}_g = -G\frac{mM}{r^2}\hat{r}$$

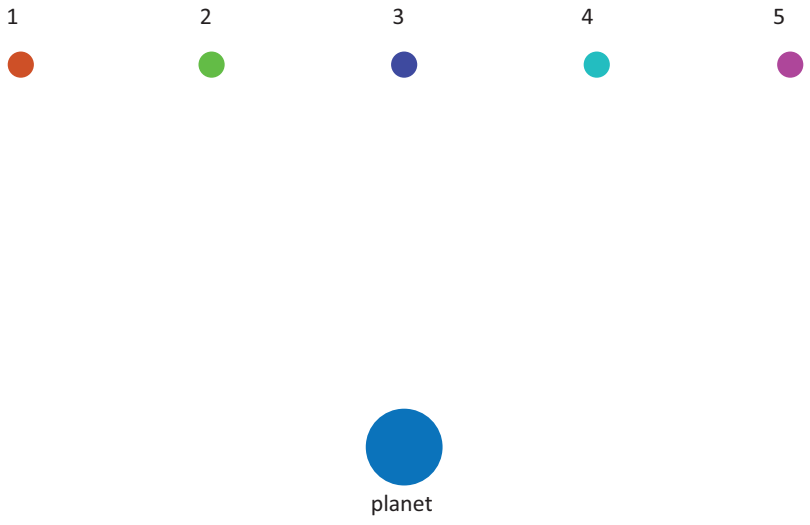
This formula is known as "Newton's Law of Universal Gravitation." The vector $\vec{r}$ is the displacement from the object creating the gravitational force to the object experiencing the gravitational force, with $r = |\vec{r}|$ and $\hat{r} = \vec{r}/r$, Note that the vector $\hat{r}$ is a *unit vector* that points in the radially outward direction, directly away from the other body. By definition, any unit vector has a magntitude of 1 (with no units!), so it contains information only about the direction of the force, not about its magnitude. (VPython hint: In VPython, if you have some vector `some_vector`, then you can find its magnitude with `mag(some_vector)`.)

1 Prediction

⇒ On a whiteboard draw a diagram like the one below. Each numbered dot represents the position of a different spacecraft. (A single spacecraft near a planet would not move in a straight line.)

⇒ At each numbered location draw an arrow representing the gravitational force on a spacecraft at that location, due to the planet.

⇒ Make sure that your diagram makes sense. Is the direction of each arrow correct? Is the length of the arrow proportional to the magnitude of the quantity it represents?



2 Objects and Constants

2.1 Creating Objects

⇒ Start a VPython program by inserting these lines at the beginning:

```
scene.width = 1000
```

⇒ Create a sphere of radius 6.4×10^6 m, located at $\langle 0, -2 \times 10^7, 0 \rangle$ m to represent the planet. Give the sphere an appropriate name.

⇒ Create a second sphere to represent the spacecraft. Give this sphere an appropriate name. You will need to exaggerate its radius to make it visible; try 3×10^6 m. Place it at location $\langle -13 \times 10^7, 4.5 \times 10^7, 0 \rangle$ m, and make its color different from the color of the planet.

2.2 Constants

In a program it is important to use symbolic names for constants. Why? To make it easy to make changes and corrections to the model. For example, if you want to change the mass of an object, it is much easier to change a single line such as :

```
mass = 1.3e7
```

than it is to search for and replace all occurrences of the string `1.3e7`.

⇒ Insert a line of code that assigns the name "G" to the value of the gravitational constant.

⇒ Also create approriately named variables to store the masses of the planet (6×10^{24} kg) and of a spacecraft (1.5×10^4 kg).

3 Calculating Gravitational Force

For the single spacecraft your program has created:

⇒ Add instructions to calculate gravitational force on the spacecraft by the planet. Use symbolic names for every quantity in the program lines you write.

⇒ Print the value you calculated. Does the direction of the force make sense?

3.1 Representing Gravitational Force with an Arrow

⇒ Create an `arrow` object with its tail at the location of the spacecraft.

⇒ Set the `axis` of the `arrow` object to the value of the gravitational force you calculated.

⇒ Run the program. What do you see? Can you explain why?

3.2 Scale Factors

We will frequently want to use `arrow` objects to visualize vector quantities such as forces. However, if the distances between objects in the program is very large compared to the magnitude of the vector, the

arrow may be too small to see. (Alternatively, if the distance between objects is very small compared to the magnitude of the vector, the arrow object may be so huge that no other objects can be seen.)

⇒ Watch the VPython instructional video *VPython Instructional Videos: 5. Scale Factors* http://vpython.org/video05.html to see how to multiply the axis attribute of an arrow by a scalar to adjust its length so all objects in the scene are visible.
⇒ Use an appropriate scale factor to scale your arrow to an appropriate length. Be prepared to explain how you determined this scale factor.

4 Multiple Locations

To calculate and display the gravitational force on several spacecraft at different locations, as in the diagram you drew previously, we can use a loop. Recall that in a previous activity you encountered loops that moved a single object. We could also create loops that create multiple objects at different locations.

⇒ Which of the following loops, Loop 1 or Loop 2, will create multiple different objects?
⇒ Which loop will move a single object across the screen?

Loop 1

```
wall = box(pos=vector(0,0,-2), length=40, height=20, width=1)
x = -20
dx = 2
ball = sphere(pos = vector(x,0,0), color=color.red, radius=0.3)
while x < 21:
    rate(2)
    ball.pos = ball.pos + vector(dx,0,0)
    x = x + dx
```

Loop 2:

```
wall = box(pos=vector(0,0,-2), length=40, height=20, width=1)
x = -20
dx = 2
while x < 21:
    rate(2)
    ball = sphere(pos = vector(x,0,0), color=color.red, radius=0.3)
    x = x + dx
```

4.1 Gravitational Force at Multiple Locations

Modify your program so that it contains a single while loop which does the following:

⇒ Creates five different spheres representing five spacecraft at these locations:
$\langle -13 \times 10^7, 4.5 \times 10^7, 0 \rangle$ m
$\langle -6.5 \times 10^7, 4.5 \times 10^7, 0 \rangle$ m
$\langle 0, 4.5 \times 10^7, 0 \rangle$ m
$\langle 6.5 \times 10^7, 4.5 \times 10^7, 0 \rangle$ m
$\langle 13 \times 10^7, 4.5 \times 10^7, 0 \rangle$ m

⇒ Calculates and prints the gravitational force on each spacecraft by the planet.
⇒ Visualizes the gravitational force on each spacecraft with an `arrow` with its tail on the spacecraft. Adjust the scale factor to make the display comprehensible, but make sure you use the same scale factor for all arrows! To make it easier to compare arrows, set the `shaftwidth` of each `arrow` equal to half the radius of a planet.

5 Forces on the Planet

⇒ Modify your program so that it also displays 5 arrows representing the forces on the planet by the five spacecraft.
⇒ Make sure the directions and magnitudes of these arrows make sense.

---

6 Explain and Predict

Study the following VPython program carefully. Make sure you understand the whole program, but don't run the program yet. Reading and explaining program code is an important part of learning to create and modify computational models.

```
scene.width = 800
scene.height = 800

G = 6.7e-11
mEarth = 6e24
mcraft = 15e3
deltat = 60

Earth = sphere(pos=vector(0,0,0), radius=6.4e6, color=color.cyan)
craft = sphere(pos=vector(-10*Earth.radius, 0,0), radius=1e6,
               color=color.yellow, make_trail=True)
vcraft = vector(0,2e3,0)
pcraft = mcraft*vcraft
t = 0
scene.autoscale = False ##turn off automatic camera zoom

while t < 10*365*24*60*60:
    rate(100)
    craft.pos = craft.pos + (pcraft/mcraft)*deltat
    t = t+deltat
```

Without running the program, answer the following questions:

⇒ What is the physical system being modeled?

⇒ In the real world, how should this system behave? On the left side of your whiteboard or paper, draw a sketch showing how you think the objects should move in the real world.

⇒ Will the program as it is now written accurately model the real system?

⇒ On the right side of the whiteboard or paper, draw a sketch of how the objects created in the program will move on the screen, based on your interpretation of the code.

7 Modify and Extend the Model

⇒ Run the program.

⇒ How did your prediction compare to what you saw? Did something happen that you didn't expect to happen?

⇒ How should the program be changed so that it is a physically reasonable model of the system?

⇒ Modify the program. Run it and compare its behavior to the behavior you expect from the real system.

8 Visualize Momentum

In a previous activity, you used arrows to visualize gravitational force vectors. In this program, you will use an arrow to visualize the momentum of the spacecraft. Note that you want one arrow that moves with the spacecraft–you don't want to create many arrows. Therefore you will create the arrow before the loop, and change its `pos` and `axis` inside the loop, just after updating the `pos` of the spacecraft. You may review the video on scale factors if necessary: *VPython Instructional Videos: 5. Scale Factors* http://vpython.org/video05.html

⇒ Print the initial momentum of the spacecraft.
⇒ Use this value to decide on a value for a scale factor that will make the arrow representing momentum a reasonable length in the display.
⇒ Before the computational loop:
  ⇒ Add a line of code that assigns the symbolic name `sf` to the scale factor.
  ⇒ Create an arrow named `parr` to represent the craft's momentum.
⇒ Inside the loop, add two lines of code (after the position update) like this:

```
parr.pos = craft.pos
parr.axis = pcraft * sf
```

⇒ Once you have seen the entire orbit, you may have to adjust the scale factor.

9 Testing a Value of a Variable

An `if` statement can be used to instruct VPython to do something only in a particular situation. The action to take is indented after the `if` statement. For example, consider the following code fragment:

```
a = 3
if a < 4:
    sphere(color=color.green)
box(pos=vector(3,3,0), color=color.cyan)
```

⇒ Try the code above in a new program window. What does it do?
⇒ What happens when you replace `a = 3` with `a = 8`?

9.1 Detecting a Collision

If your spacecraft collides with the Earth, the program should stop.

⇒ Add code similar to the following inside your loop (using the name you defined for the relative position vector between the spacecraft and the center of the Earth):

```
if mag(r) < Earth.radius:
    break  ## exit from the loop
```

The `break` instruction tells VPython to get out of the loop and go to the first instruction after the loop (if there is one.) Because the `break` instruction is indented after the `if` statement, it will be executed only if the `if` test returns a value of `True`.

10 Direction of Momentum

Record your answers to the following questions:

⇒ For this elliptical orbit, what is the direction of the spacecraft's momentum vector? Tangential? Radial? Something else?
⇒ What happens to the momentum as the spacecraft moves away from the Earth?
⇒ As it moves toward the Earth?
⇒ Why? Explain these changes in momentum in terms of the Momentum Principle.

11 Effect of the Initial Velocity

⇒ Approximately, what minimum initial speed is required so that the spacecraft "escapes" and never comes back? You may have to zoom out to see whether the spacecraft shows signs of coming back. You may also have to extend the time in the while statement.
⇒ What initial speed is required to make a nearly circular orbit around the Earth? You may wish to zoom out to examine the orbit more closely.
⇒ How does increasing the initial speed affect the orbit? Explain this by considering the first few time steps.
⇒ How does decreasing the initial speed affect the orbit? Explain this by considering the first few time steps.

12 Adding an Arrow to Represent Force

⇒ Choose an initial speed that produces an elliptical orbit.
⇒ Add a second, different colored arrow representing the net force on the spacecraft. This arrow should also move with the craft. You'll need a different scale factor for this arrow.
⇒ Are the net force on the craft and the momentum of the craft in the same direction?
⇒ What is the relative direction of these arrows when the craft is slowing down?
⇒ Speeding up?
⇒ Draw a diagram showing the directions of the craft's momentum and the net force on the craft at 6 different locations along an elliptical orbit. At each location note whether the speed of the craft is increasing, decreasing, or momentarily not changing.
