

Calculating gravitational force with VPython

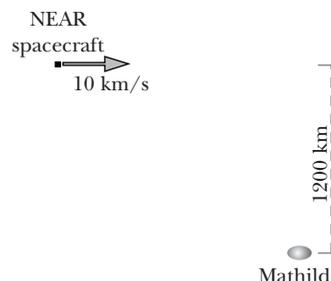
Every program that models the motion of physical objects has two main parts:

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 (the subject of this lab activity)*
 - 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

In your fan cart program you studied motion with a constant force. But in the orbit of a planet around a star, or two stars orbiting each other (a “binary star”), the gravitational forces on the objects may change continually in magnitude and direction.

In this lab activity you will learn how to make VPython calculate gravitational forces, as vectors. Knowing how to do this will enable you to write programs that predict the motion of objects in space, for if you can calculate the net force on an object, you can update its momentum just as you did in the fan cart program. Knowing the momentum lets you update the position. Repeating these momentum and position updates will trace out a complete trajectory.

1 Creating the objects



The NEAR spacecraft flew past the asteroid Mathilde (for details, see Problem 2.11, p. 70 of the textbook). Create spheres representing Mathilde and the spacecraft:

```
from visual import *
craft = sphere(pos=vector(-1.2e6,0.6e6,0), radius=1e4, color=color.cyan)
Mathilde = sphere(pos=vector(0,-0.6e6,0), radius=5e4, color=color.white)
```

- Run the program to see Mathilde, with the spacecraft at the upper left. The size of Mathilde is about right, but the size of the spacecraft has been greatly exaggerated so that you can see it.

2 Giving the objects physical attributes

In addition to geometrical attributes such as **pos** and **radius**, you can give objects physical attributes such as mass. The mass of the NEAR spacecraft was 805 kg; add the following statement to your program:

```
craft.m = 805
```

The mass of Mathilde was estimated from its size as seen in photos taken by the spacecraft, and the density of typical rocks. The density of rocks is about 3000 kg/m^3 , and Mathilde appeared to be about 70 km by 50 km by 50 km in the photos ($7e4 \text{ m}$ by $5e4 \text{ m}$ by $5e4 \text{ m}$). Add a statement that calculates the estimated mass of Mathilde in kilograms. *Write out the calculation in the program and let Python do the calculation for you; don't use your calculator.* This lets you see in the program how the mass was calculated.

```
Mathilde.m = ?
```

3 Calculating the gravitational force on the spacecraft

Two spheres with masses m_1 and m_2 attract each other with a force given by $\vec{F}_{\text{grav}} = G \frac{m_1 m_2}{|\vec{r}_{12}|^2} \hat{r}$, where

$$G = 6.7 \times 10^{-11} \text{ N} \cdot \text{m}^2 / \text{kg}^2$$

\vec{r}_{12} is a relative position vector pointing from one object to the other

\hat{r} is a unit vector pointing from one object toward the other

In the following sections you will calculate the gravitational force that the Mathilde asteroid exerts on the NEAR spacecraft, by

- calculating the relative position vector \vec{r}_{12} that points from one object to the other
- calculating its magnitude $|\vec{r}_{12}|$
- using $|\vec{r}_{12}|$ to calculate the magnitude of the gravitational force $|\vec{F}_{\text{grav}}| = G \frac{m_1 m_2}{|\vec{r}_{12}|^2}$
- calculating the unit vector \hat{r} , which gives the direction of the gravitational force
- calculating the vector gravitational force, the product of its magnitude and direction: $\vec{F}_{\text{grav}} = |\vec{F}_{\text{grav}}| \hat{r}$

3.1 Calculate the relative position vector

You know the vector positions of the two objects, which are `craft.pos` and `Mathilde.pos`.

- Add a statement to your program to calculate a vector \mathbf{r} that points from Mathilde to the spacecraft. Think about what you know about calculating relative position vectors between two objects. *Don't use any numbers, just symbols.*

$\mathbf{r} = ?$

3.2 Calculate the magnitude of the relative position vector

In order to calculate the magnitude $|\vec{r}_{12}|$ of the relative position vector, you need to know the following:

- The components of a vector in VPython are given by its x, y, and z attributes. For example, `craft.pos.x` is the x component of the position of the spacecraft, and `r.y` is the y component of the vector \mathbf{r} that you created.
- To calculate a square of a quantity, use two asterisks. For example, `3**2` gives 9.
- To calculate the square root of a quantity, use the function `sqrt`. For example, `sqrt(9)` is 3.

Knowing these features of VPython,

- Add a statement to your program to calculate the magnitude $|\vec{r}_{12}|$ of the relative position vector:

`rmag = ?`

3.3 Calculate the magnitude of the gravitational force

Using the quantity $|\vec{r}_{12}|$ that you just calculated (`rmag`), and the masses `craft.m` and `Mathilde.m`,

- Add a statement to your program to calculate the magnitude $|\vec{F}_{\text{grav}}|$ of the gravitational force. You should define `G` near the start of your program and use it here.

`Fmag = ?`

3.4 Calculate the direction of the gravitational force

You have calculated a vector that points from Mathilde to the spacecraft. The direction of this vector is along the line of the gravitational forces between the two objects, so if we knew the direction of \vec{r}_{12} in the form of its unit vector \hat{r} , we could use it to get the direction of the gravitational force.

- You know both the vector \vec{r}_{12} (\mathbf{r}) and its magnitude ($\mathbf{r}\mathbf{mag}$). Use these quantities to add a statement to your program to calculate the unit vector \hat{r} (pronounced “r-hat”):

rhat = ?

3.5 Calculate the gravitational force as a vector

Now you have everything you need to be able to calculate the gravitational force that the Mathilde asteroid exerts on the spacecraft. Remember that any vector can be factored into a product of its magnitude times its direction. You know the magnitude of the gravitational force, so to get the vector force you just need to multiply by the unit vector for the force. You already have the unit vector for the relative position vector, which lies along the same line as the force vector.

Think carefully about whether you want to multiply the magnitude of the gravitational force times \hat{r} , or rather by $-\hat{r}$. That is, is the force that acts on the spacecraft in the same direction as the **rhat** unit vector (or \mathbf{r} vector), or in the opposite direction?

- Add a statement to your program to calculate the net vector force acting on the spacecraft. We’ll assume that Mathilde is the only object near enough to have a significant effect on the spacecraft.

Fnet = ?

- Add a print statement to your program to show the components of the net force:

```
print "Fnet =", Fnet
```

- Run the program, and ask yourself whether the signs of the force components make sense (then make changes to your program if necessary).

4 Visualizing the force vector with an arrow

Having calculated the gravitational force vector, we want to visualize it by displaying an arrow representing the vector.

- Add a statement to your program to create an arrow on the screen that represents the gravitational force acting on the spacecraft. Choose **pos** and **axis** attributes so that the tail of the arrow is on the spacecraft and the arrow points toward Mathilde. *Do NOT use numbers!* Write the values for the **pos** and **axis** attributes symbolically, in terms of the quantities you have already calculated.

```
arr1 = arrow(pos=?, axis=?, color=color.yellow)
```

If you have calculated the gravitational force correctly, you probably don’t see an arrow! You may remember that in your ball in a box program the velocity arrow was huge, and you had to scale it down. Here the force is so small you have to scale it up to be able to see it. How do you pick a scale factor?

Look at the output of your **print** statement to get an idea of about how big the force is. Your graphics display is scaled to be about 2000 km wide (2e6 m). If for example the magnitude of the force were 100 newtons, you would need to scale the force up by a factor of about $1e6/100 = 1e4$ in order to see anything.

- Add a statement near the start of your program to calculate an appropriate scale factor **Fscale**, and use it to scale the axis of your arrow statement, so that the arrow is visible. Single numbers, positive or negative, are called “scalars,” and they can be used to scale vectors.

Fscale = ?

5 The force acting on Mathilde

Now that you have displayed the force that the Mathilde asteroid exerts on the spacecraft, also display the force that the spacecraft exerts on Mathilde. Think carefully about how this force would be calculated. You should realize that you don’t need to do any new calculations! If you find yourself doing new calculations, you’ve missed an important property of the gravitational force law.

- Add a statement to your program to display an arrow representing the force that the spacecraft exerts on Mathilde. Put the tail of the arrow on Mathilde. *Don't add any new calculations, and don't use any numbers, just symbols.*

```
arr2 = arrow(pos=?, axis=?, color=color.yellow)
```

A question to ponder: In a ten second time interval, how would the change in the *momentum* of the spacecraft compare to the change in the *momentum* of Mathilde? How would the change in the *velocity* of the spacecraft compare to the change in the *velocity* of Mathilde? How would the *total momentum* (the momentum of the spacecraft plus the momentum of Mathilde) change during this time interval? Your instructor will ask you about these issues.

CHECKPOINT G1: Ask an instructor to check your work for credit.
You can read ahead while you're waiting to be checked off.

6 The force at other positions

In order to see how the force on the spacecraft would vary at different positions of the spacecraft relative to Mathilde, calculate and display the force on the spacecraft (and on Mathilde) for four other locations.

- Add the following statement to your program, to place the craft at a new position:

```
craft.pos = vector(-0.6e6, 0.6e6, 0)
```

- Copy and paste your previous calculations at the end of the program, and run the program. You should see arrows representing force vectors for two different locations of the spacecraft.
- Once you have seen the two pairs of force vector arrows displayed correctly, repeat these additions to the program for the positions $\langle 0, 0.6e6, 0 \rangle$, $\langle 0.6e6, 0.6e6, 0 \rangle$, and $\langle 1.2e6, 0.6e6, 0 \rangle$.
- If the arrows get too big, reduce your scalar **Fscale** so that the arrows don't run into each other. You must use the *same* **Fscale** with all of your arrows, so that all the force vectors are consistent with each other.

CHECKPOINT G2: Ask an instructor to check your work for credit.
You can read ahead while you're waiting to be checked off.

7 Playing around

You might like to display the force arrows for some additional positions of the spacecraft.

In order to emphasize the physics rather than programming techniques, we had you make multiple copies of the calculational statements, which is clunky, and not feasible if there are hundreds of positions involved. It is not difficult to have just one copy of the calculations in a while loop where you increment the x component of the position of the spacecraft each time through the loop. Try it!

An advanced topic is to modify your program in such a way that you can drag the spacecraft around with the mouse, displaying the forces as you move, in which case you don't want to create new arrows all the time, just update the attributes of the two existing arrows. To see how to handle mouse interactions, study the VPython demos that use the mouse, and see the "Visual" documentation on the IDLE help menu. Choose "Reference manual", then "Mouse Interactions". You may want to make the spacecraft bigger to make it easier to select and drag it.

Another advanced topic is to define a function ("def" in Python) that calculates the forces and displays them, then "call" that function repeatedly. We won't need this technique in this course, but if you get interested in programming you might like to study the Python documentation available on the IDLE help menu. Python is an important and powerful programming language; we will barely scratch the surface of its capabilities.