

Calculating and displaying the electric field of a dipole

1 Problem Statement

Write a VPython program which does an exact calculation of the electric field of a dipole, and displays the field at 22 locations as specified in this handout, including locations not on the standard dipole axes.

The dipole consists of:

- a particle of charge $+e$ located at $\langle 4 \times 10^{-11}, 0, 0 \rangle$ m, represented by a red sphere, and
- a particle of charge $-e$ located at $\langle -4 \times 10^{-11}, 0, 0 \rangle$ m, represented by a blue sphere.

The observation locations should be:

- 12 locations evenly spaced on a circle of radius 3×10^{-10} m in the xy plane, centered on the dipole
- 12 locations evenly spaced on a circle of radius 3×10^{-10} m in the xz plane, centered on the dipole.

At each location you will display an arrow representing the net electric field at that location, due to both of the charges in the dipole. By writing a program you can determine the electric field at any location in space, not just locations along the axis of the dipole, or along a line perpendicular to the dipole axis.

2 Diagram and planning

On paper or a whiteboard, make a careful 2D diagram of the system, in the xy plane.

Each particle, position, and vector should be labeled with the name you will use for it in the program.

Your diagram should include

- both charged particles
- a single observation location (not on an axis)
- **all relative position vectors you will need** (think about what the “source” charges are)
- the individual electric field vectors you will calculate (just estimate their directions, and draw arrows)
- the net electric field vector at the observation location (estimate this and draw an arrow)

**Make sure you have labeled particles, positions, and vectors on your diagram.
Then discuss your diagram with a neighboring group.**

3 Write the program

Your program should be organized with these sections:

```
## setup and constants
## objects
## initial values
## calculations
```

3.1 Setup and constants

Start with these two statements:

```
from __future__ import division
from visual import *
```

In the first line note the two underscores before and two after the word “future”.

This makes Python treat $(1/2)$ as a floating point number (0.5) instead of an integer (0).

After typing the first two lines to invoke VPython, enter the values of the physical constants you will need. You will also eventually need a scale factor for representing electric field vectors with arrows, so you can go ahead and create it now, temporarily giving it a value of 1.0 (you will change this later).

```
## constants
oofpez = 9e9
qproton = 1.6e-19
scalefactor = 1.0 # temporary value
d = 3e-10 # distance from origin to observation locations
```

The first line gives the name “oofpez” (which stands for One Over Four Pi Epsilon-Zero), to the number 9e9. The second line gives the name “qproton” to the charge of a proton.

3.2 Objects

In your program add this comment line:

```
## objects
```

You will need two spheres to represent the two charged particles. Give each one a radius of 1e-11 m. Make the negative charge blue (`color=color.blue`) and the positive charge red. If you don't remember how to create a sphere, check the Visual reference manual available on the Help menu.

3.3 Initial values

To get started, you will calculate the net electric field at one observation location. Later, you will modify this, and write a loop to calculate and display the electric field at many observation locations. You will need to have a variable that contains the value of the observation location. In your program add this:

```
## initial values
obslocation = vector(d,0,0) # d is a previously defined constant
```

3.4 Calculations

You can't use the “analytical” formulas derived in the textbook for the electric field of a dipole in these VPython calculations, because you are going to calculate the field at many locations not on the axis or perpendicular to the axis, where these formulas don't apply. You must go back to the superposition principle. At each observation location, you must calculate the electric field due to the positive charge, the electric field due to the negative charge, and the net electric field, which is the sum of these two.

VERY IMPORTANT

Refer to your diagram to see what quantities you need to tell the computer to calculate.

- Calculate the net electric field at the location given by “obslocation”. Remember that for each of the two charges you need to calculate
 - a vector from the source charge to the observation location,
 - the magnitude of that vector, and
 - the corresponding unit vector,in order to be able to calculate the electric field vector contributed by that charge. Then you need to add the two contributions to get the net field. Print the net electric field (“`print Enet`”).
- To check your calculation, compare your answer to the answer you get using the approximate formula for the electric field on the axis of a dipole (use VPython to evaluate and print this result).

The magnitude of a vector \mathbf{r} can be calculated as `sqrt(r.x**2 + r.y**2 + r.z**2)`, or more easily as `mag(r)`.

3.5 Arrow representing electric field at one location

Given the size of the display (in meters), a reasonable length for an arrow representing the electric field vector would be something like 1e-10 m, so you need to scale the electric field to this approximate length. You want $|\vec{E}| * \text{scalefactor}$ to be about 1e-10, so write `scalefactor=1e-10/(typical E value)`.

You already printed a typical E value near the dipole, so use value that to calculate `scalefactor` in the constants section of your program. You may want to adjust `scalefactor` later.

- Create an orange arrow representing the electric field at the observation location, and scale it appropriately by setting the arrow's axis to the scalefactor times the electric field. The charges and the arrow must be clearly visible. If you don't remember how to create an arrow, check the Visual reference manual available on the Help menu. Note that the axis of the arrow is relative to the tail of the arrow.

**Make sure everyone in the group agrees that the display looks reasonable.
Then discuss your display with a neighboring group.**

4 Adding more observation locations

To see the pattern of electric field around a charged particle, you will extend your program to calculate the electric field at 12 locations, all lying on a circle of radius d in the xy plane, centered on the dipole.

Instead of copying and pasting code 12 times, you will put your existing calculations in a loop.

NOTE: An easy way to do this is to select your code and on the Format menu choose "Indent Region". The coordinates of a location on a circle of radius d in the xy plane are $d \cdot \langle \cos(\theta), \sin(\theta), 0 \rangle$.

4.1 Calculating electric field at multiple locations on a circle surrounding the dipole

The organization of your program should look like this:

```
from __future__ import division
from visual import *
## constants
... [your constants here, including scalefactor]
## objects
... [your particles here]
## initial values
theta = 0
## calculations
while theta < 2*pi:
    ## calculate new vector value for obslocation, using theta
    obslocation = d*vector(cos(theta), sin(theta), 0)
    ## use superposition principle to calculate Enet at obslocation
    [your code here]
    ## print values of obslocation, and Enet vector
    ## create an orange arrow to display Enet at obslocation, and scale it
    [your code here]
    ## calculate new value of theta in radians: add (2*pi/12) to theta
```

- Run your program. It should calculate and display the electric field as arrows at 12 locations on a circle surrounding the particle. Do not display the relative position vectors – there are too many of them, and they will clutter up the display.
- Your program should print the value of each observation location, and E_{net} at that location.
- Look at your display. Does it make sense? Do the magnitudes and directions of the 12 orange arrows make sense?

**Make sure everyone in the group agrees that the display looks like the field of a dipole!
Then discuss your display with a neighboring group.**

5 Adding more locations in a different plane

Most of your program is done, and you are nearly ready to turn it in.

- Add code to compute and display the electric field at 12 evenly spaced locations lying in a circle of radius d in the xz plane.
- Note that two of the locations (on the x axis) will be the same as two of the previous locations. This is okay. The electric field arrows at these two locations should overlap completely – it should look as if there is only one arrow there.
- Rotate the “camera” with the mouse to get a feeling for the nature of the 3D pattern of electric field.

**Make sure everyone in the group agrees that the display looks right.
Discuss your display with a neighboring group.
Finally, show your display to your instructor.
Then the Recorder turns in the group’s program to WebAssign.**

You may be asked to make some changes to the program, and answer some questions about it.

7 Playing around

Here are some suggestions of things you might like to try after turning in your program.

- If there are red-cyan stereo glasses available (red lens goes on left eye), you can display true stereo by adding the following statement near the start of your program (for more about this, see the section “Controlling Windows” in the Visual reference manual available from the Help menu):

```
scene.stereo = 'redcyan'
```

- Add a circle of observation locations in the xz plane.
- Modify a copy of your program so that you place a proton at rest at a location on the +y axis, then release it, and animate its motion by calculating the electric force, then applying the momentum principle $\vec{p}_f = \vec{p}_i + \vec{F}_{\text{net}}\Delta t$ to update the momentum, then update the position, in a loop. Note that in order to calculate the force on the proton you always have to calculate the electric field of the dipole at the current location of the proton. Leave a trail, so you can study the trajectory:

```
trail = curve(color=color.yellow) # before the loop
...
## in the loop, after updating the proton's position:
    trail.append(pos=proton.pos)
```

Try a Δt of around $1\text{e-}17$ s, to start with. The trajectory is quite surprising. Think about whether this trajectory is consistent with the Energy Principle.

- In the Visual reference manual available on the Help menu, in the section on “Mouse Interactions”, find out how to determine the current position of the mouse. Display the electric field at the mouse location, either where you click the mouse, or continuously as you drag the mouse. It’s interesting to display the electric fields contributed by each charge, as well as their sum, the net electric field.
- Make a “quadrupole” out of two opposed dipoles, and display the electric field. The dipoles can be aligned on one axis (+- - +) or they can be placed one above the other, with opposite orientations:

$$\begin{array}{c} + - \\ - + \end{array}$$

You’ll need to increase `scalefactor`, because the two dipoles nearly cancel each other’s fields.

- Make a stack of dipoles, and display the electric field.