Calculating and displaying the electric field of a single charged particle

You have calculated the electric field produced by a single charged particle. The somewhat tedious process of calculating electric field vectors can be automated by programming a computer to do this. In addition, doing the calculations in VPython will allow us to display the electric field in multiple locations, so we can examine the 3D pattern of field created by a charged particle.

*Asterisks mark exercises for which answers are given at the end of the writeup.*

I. Program organization

You should organize a program in the following way, inserting comments (##) to explain what the different sections of the program are doing:

- First, a section giving names to any constants that will be used.
- Second, a section in which visible objects like spheres or arrows are created and named. (Sometimes you will end up creating more objects later.)
- Third, a section in which important variables are named and given initial values, if any.
- Fourth, the calculations.

II. A VPython program to calculate the electric due to a single charged particle.

II.1. Creating a new program file

Start a new VPython program. See the VPython Reference Sheet for instructions on how to start IDLE and begin editing your file. Remember to leave the Python Shell window visible at all times. Do not close this window, because it gives you error messages; end the program by closing the graphical display window.

The first two lines at the very beginning of your program should be:
```
from __future__ import division
from visual import *
```

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

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

II.2. Constants

Constants should be defined at the beginning of your program. After the two lines you inserted in section III.1 above, type:
```
## constants
oofpez = 9e9
qproton = 1.6e-19
```

The first line gives the name “oofpez” (which stands for One Over Four Pi Epsilon-Zero), to the number \(9\times10^9\). The second line gives the name “qproton” to the charge of a proton.

- To make sure you have typed everything correctly, add the statement
  ```python
  print oofpez
  ```
- Run your program by pressing F5 or by using the Run menu. Does the correct value of the constant print out in the Python Shell window?

II.3. Creating the charged particle

In your program add this comment line:
```
## objects
```
Now create a sphere to represent a charged particle.  
The sphere should be located at the origin.  
The name of the sphere should be “particle”  
The radius of the sphere should be 1e-11 m. (This is much larger than the radius of a proton, which is about 1e-15 meters, but we will exaggerate the size of the particle in order to make it easily visible in our display.)

Run the program. You should see a sphere in the center of your display window. If you don’t see anything, first check the Python Shell window for error messages, then try again.

II.4 Initial values of variables

In your program add this comment line:

```python
## initial values
```

Now we’ll assign a symbolic name to the observation location. We need a symbolic name for this location, because we will eventually want to instruct VPython to calculate the electric field at many different observation locations. Since there is no object at the observation location, we must create a vector variable to represent the observation location. We’ll call it “obslocation”.

Type the statement:

```python
obslocation = vector(2.1e-10, 2.1e-10, 0)
```

This defines a vector named “obslocation” to represent the location where we want to find the electric field. Later we will have the program calculate the electric field at other locations, too. A vector is not a displayable graphical object, so it doesn’t have a “pos” attribute. We refer to this vector simply by its name obslocation. The position attribute of the displayable sphere object is a vector which is referred to as particle.pos.

III Calculations

III.1 Instructing VPython to calculate and display the relative position vector \( \vec{r} \)

When calculating the electric field at an observation location, the relative position vector always points from the source particle (the initial location) to the observation location (the final location).

- In your program, write a line of code that calculates the relative position vector \( \vec{r} \). Call this vector \( \vec{z} \).
- Add a line of code to print the relative position vector from source charge to observation location:

```python
print "relative position vector is", r
```

Note that anything in quotes inside a print statement is printed as text, while anything not in quotes is assumed to be a symbolic name, and its value is printed; separate quantities to be printed with commas as shown.

- Now create a green arrow named "\( \vec{ra} \)" whose tail is at the center of the source charge, and whose axis is \( \vec{r} \), the relative position vector you just calculated.
- Run the program. In the display window you should see a green arrow from the source charge to the observation location, and in the Python Shell window you should see the printed value of \( \vec{r} \).

III.2 Telling VPython how to calculate the magnitude of \( \vec{r} \)

We need to translate the equation \( |\vec{r}| = \sqrt{(r_x)^2 + (r_y)^2 + (r_z)^2} \) into VPython.

Referring to vector components

The components of a vector are attributes of the vector, so they can be referred to with “.” syntax. The components of a vector named “\( \vec{r} \)” are \( r_x, r_y, \) and \( r_z \)

Exponentiation operator

In VPython, use the operator ** to raise a quantity to a power. \( b^3 \) should be written \( b**3 \)

Square root operator

In VPython the function \( \text{sqrt()} \) is used to compute a square root. \( \sqrt{3} \) should be written \( \text{sqrt}(3) \)
• In your program, write a line of code to calculate the magnitude of the relative position vector \( \mathbf{r} \). Call this quantity \( r_{\text{mag}} \).
• Add a line of code to print the value of \( r_{\text{mag}} \):
  ```python
  print "magnitude of r is", rmag
  ```
• Run your program. Compare your printed value to the answer on the last page.*

III.3 Calculating the unit vector \( \mathbf{\hat{r}} \)
• In your program, calculate the unit vector \( \mathbf{\hat{r}} \) in the direction of \( \mathbf{r} \). Call this vector “\( \mathbf{r}\hat{\mathbf{r}} \)”.
• Add a line of code to print the value of \( \mathbf{r}\hat{\mathbf{r}} \):
  ```python
  print "unit vector \( \mathbf{r}\hat{\mathbf{r}} \) is", \( \mathbf{r}\hat{\mathbf{r}} \)
  ```
• Run your program. Compare your printed values to the answer on the last page.*

III.4 Calculating and displaying the electric field at the observation location
You have now calculated \( \mathbf{r} \), \( r_{\text{mag}} \), and \( \mathbf{r}\hat{\mathbf{r}} \). Finally, we need to use all of these pieces to calculate the electric field, as a vector, at the observation location. Write down the symbolic algebraic equation used to calculate the electric field vector, then convert it to VPython. Use the name “\( \mathbf{E} \)” for the electric field vector you will calculate.

• To check your work, add a print statement after the calculation of the electric field:
  ```python
  print "Electric field vector is", \( \mathbf{E} \)
  ```
• Run your program. Compare the answer computed by your program to the value given on the last page.*

III.5 Drawing an arrow to represent the electric field vector
We want to represent the electric field vector we calculated by an arrow. Create an orange arrow (color.orange) and give the arrow the name “\( \mathbf{ea} \)”.
• Remember that the tail of the arrow must be placed at the observation location!
• Think about what the axis of the arrow should be, and write the appropriate code.
• Run the program.

You should see an arrow pointing in the appropriate direction. However, you can’t see the sphere representing the positively charged particle, or the arrow representing the position vector! The arrow is so big that when VPython positions the “camera” so we can see the arrow, the sphere is too small to see.

III.6 Scaling the arrow to a reasonable size
At the moment, the axis of the arrow is a vector equal to the electric field at the observation location. We need to “scale down” the arrow so it is not gigantic compared to the sphere representing the charged particle. To do this, we need to multiply that vector by a scalar, thus changing the magnitude of the vector without changing its direction.

Visually, the display would look sensible if the length of the arrow were about 2e-10 m, which is the same as the distance from the source charge to the observation location. Currently, the length of your arrow is the same as the magnitude of the electric field.

We need to multiply \( \mathbf{E} \) by a scalar factor to decrease its magnitude, while keeping its direction the same.

III.6.1 Estimating a value for the scale factor.
We can estimate a scale factor “\( \text{scalefactor} \)” by noting that the distance between the source charge and the observation location is 3e-10 m, so a reasonable length for the arrow in this case would be something like 3e-10 m (of course, it might be different in a different program). So we can estimate a good value by noting that if we want \( |\mathbf{E}| \times \text{scalefactor} = 3 \times 10^{-10} \text{ m} \), then \( \text{scalefactor} = 3 \times 10^{-10} \text{ m} / |\mathbf{E}| \). You can either estimate the magnitude of \( \mathbf{E} \) from the printed vector, or calculate it on your calculator, or tell VPython to calculate it and print it out in your program.

Your program should now include the following, to scale down the size of the arrows:

\[
\text{scalefactor} = (\text{your value}) \quad \# \text{ in the constants section of the program}
\]
\[
\mathbf{ea} = \text{arrow}(\text{pos} = (\text{appropriate location}), \text{axis}=\text{scalefactor}\times\mathbf{E}, \text{color}=\text{color.orange})
\]
Run the program, and make sure you can see the sphere, the arrow representing the position vector, and the arrow representing the electric field vector. You may want to change the scalefactor so the display looks better to you.

Check your own work:
  Can you see the green position arrow?
  Does the orange arrow point in the correct direction?
  Is the tail of the orange arrow at the observation location?

IV. Add 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 many locations, all the same distance from the source charge.

One way to do this is to copy the code you have written to calculate and display the electric field, and paste it in multiple times, typing new values for the observation location each time. This is the way we’ll do it this time; later we will learn a more flexible way to do this.

• In your program, copy code so as to calculate and display the electric field at 6 more locations.
• Do not copy the code for the green arrow representing the position vector.
• Each location should be 3e-10 m from the origin, and should be located on one of the axes. So, for example, you should have one location at <3e-10, 0, 0> m and another at <3e-10, 0, 0> m, and so on:

Your program should display 7 orange arrows, representing the electric field at 7 locations. 6 of these orange arrows should be on the +x, -x, +y, -y, +z, and -z axes.
There should 1 green position arrow from the source to the first observation location.
The arrows should point in the correct directions!

V. Label a position arrow and a field arrow

On the Help menu in IDLE, choose “Visual” and then “Reference manual” to learn how to put a text label on your graphics display. Or go to the Documentation section of http://vpython.org. At the center of your first green position arrow (halfway along its length), place a label “r”, and at the center (halfway along its length) of the orange arrow representing the electric field at that location, place a label “E”. Note that on the IDLE Help menu you can also learn more about the Python programming language upon which VPython is based.

VI. Turn in your program in WebAssign

Read the instructions in WebAssign carefully. You may be asked to change some values, and to report some values, before you turn it in. An instructor will run your program and give it a grade. You will get full credit if your program correctly calculates and displays the 1 position vector and 7 electric field vectors specified in the WebAssign assignment, and has appropriate labels on the r vector and the first E vector.

** Answers to starred questions: magnitude is approx. 3e-10 m; unit vector is approx. <0.707, 0.707, 0>;
field is approx. <1.15e+010, 1.15e+010, 0> N/C.