

## Calculating and displaying the magnetic field of a single moving charged particle.

In this computer lab, you will compute and display the magnetic field at a single location in space, due to a moving charged particle. As the particle moves, the magnetic field will of course change. Your finished program will show a particle, initially located at  $\langle 4 \times 10^{-10}, 0, 0 \rangle$  m with velocity  $\langle -4 \times 10^4, 0, 0 \rangle$  m/s moving across the screen, and magnetic field vectors at four observation locations, changing dynamically as the particle moves.

This handout is organized into three parts:

- 1: Calculating and displaying the magnetic field due to a single particle at one instant
- 2: Animating the motion of a particle across the screen
- 3: Repetitively calculating and updating the display of the magnetic field at the observation location, due to the moving particle.

### 1. Calculating and displaying the magnetic field due to a moving particle at one instant

As usual, the first two lines at the very beginning of your program should be:

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

Remember that the second line should be typed exactly as follows:

```
from (space) (underscore) (underscore) future (underscore) (underscore) (space) import (space) division
```

Also remember that your program should be organized into four sections:

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.

Fourth, the calculations, which will probably be repeated inside a loop.

#### *Calculating a vector cross product in VPython*

You can calculate a vector cross product in VPython by calculating each component of the result, the way you would do it by hand. For example, if  $\vec{C} = \vec{A} \times \vec{B}$ , then you can write the following:

```
A = vector(3, 0, 0)
B = vector(0, 2, 0)
C = vector((A.y*B.z-A.z*B.y), (A.z*B.x-A.x*B.z), (A.x*B.y-A.y*B.x))
```

- Try the preceding lines of code, and print the answer to make sure it is correct.

Since VPython can do vector operations, you can also ask VPython to do the cross product:

```
D = cross(A, B)
```

- Try this, and print the answer to see if you get the same result.
- Comment out these lines of code, but leave them at the end of your program.

### Calculating a magnetic field in VPython

1. Define the necessary constants.
2. In your program, the moving particle will be a proton. Create a red sphere representing the particle at the initial location specified in the problem statement above. Give the sphere a radius of  $1e-11$  m (this is of course larger than the real radius of a proton; we need to be able to see the sphere. Name the sphere so you can refer to it later.
3. Create a vector quantity velocity representing the velocity of the proton specified above.
4. Create an arrow at the observation location  $\langle -6 \times 10^{-11}, 0, 6 \times 10^{-11} \rangle$  m with `axis=vector(0, 0, 0)`. Name the arrow `barrow1` so you can refer to it later. Note that the tail of the arrow is at the observation location, so you can simply refer to the `pos` of the arrow when you want to refer to the observation location.
5. Run the program. You may need to zoom out a little in order to see the particle.
6. Why don't you see the arrow? (Be prepared to explain this to an instructor.)
7. Write symbolic VPython statements to calculate the magnetic field at the observation location, due to the "moving" proton. (Think about what quantities you need to know in order to do this calculation).
8. Print the value of the magnetic field (which should be a vector).
9. Is the direction of the magnetic field you calculated correct? Check with the right-hand rule.

### Estimating a value for the scale factor

To make a reasonable-looking arrow representing the magnetic field vector, you need a scale factor. To get started, suppose you want the arrow to be small but visible when the proton is in its current location. In the distance units of the display, how long should the arrow be? (What is the distance between the proton and the observation location? How do you want the length of the arrow to compare to that distance?)

**Remember that scalefactor = (max. length you want the arrow to have)/(max. magnitude of the field it represents)**

At the moment you don't know the maximum magnitude of the magnetic field, so you may need to adjust the scale factor later.

10. Since you already created an arrow, you don't need to do it again. All you need to do is change its axis. Set the axis of the arrow `barrow` to be the magnetic field you calculated, multiplied by an appropriate scale factor to make the size of the arrow appropriate for this display.

```
barrow.axis = (variable representing magnetic field)*scalefactor
```

11. Run the program. Do you in fact see a cyan arrow at the observation location, pointing in the appropriate direction?

**CHECKPOINT VP1: Have an instructor check your work, for credit.**

## 2. Animating the motion of a particle

A computer animation of the motion of an object uses the same principle as a movie or a flip book does. You display the object at successive locations, each at a slightly later time; to your eye, the motion can look continuous. In a program, you calculate the position of the object, and it is displayed in that position. You calculate where the object will be a very short time  $\Delta t$  later, and change its position to the new location, so it is displayed in the new location.

To keep the camera from zooming in and out again, insert this line **after the line creating the sphere**:

```
scene.autoscale = 0
```

You will use the relation between velocity (a vector) and position (a vector) to calculate the position of a moving particle at successive times, separated by a short time step  $\Delta t$ . Remember that, in vector terms:

$$\vec{r}_f = \vec{r}_i + \vec{v}\Delta t \text{ (because } \vec{v} = \frac{\Delta\vec{r}}{\Delta t}\text{)}$$

In VPython, this translates to a statement like:

```
particle.pos = particle.pos + velocity*deltat
```

where `velocity` is the vector quantity which you have initialized earlier in the program.

This statement may look odd to you if you have not had much programming experience. The equals sign here has the meaning of "assignment" rather than equality. This instruction tells VPython to read up the old position of the particle, add to it the quantity, and store the new position as the current position of the particle. This will automatically move the particle to the new position.

For this program you will use a time step of  $5 \times 10^{-20}$  seconds:

```
deltat = 5e-20
```

Stop the loop when the particle has gone off the screen (i.e. when the y-component of the particle's position becomes greater than  $5 \times 10^{-10}$  m). Your loop might start like this:

```
while particle.y > -5e-10:  
    particle.pos = particle.pos + velocity*deltat
```

12. Run your program. Does the particle move down the screen?

**CHECKPOINT VP2: Have an instructor check your work, for credit.**

### 3. Calculating and updating the magnetic field as the particle moves

When the particle moves, the magnetic field due to the particle changes, so you need to recalculate it each time you move the particle.

13. Think about the Biot-Savart law (the equation you use to calculate the magnetic field of a moving charged particle). Which of the quantities in this equation change when the position of the particle changes? Which quantities do not change?
14. Move all the lines of code involving changing quantities so they are inside the loop, after the line of code updating the particle's position.
15. **Do NOT move the line of code that creates the arrow! This line should remain before the loop. You do not want to create a new arrow each time through the loop - this would make thousands of arrows, all on top of each other!** If your program runs very slowly, you may have made this error.
16. **Do NOT put a print statement inside the loop.** This makes the program run very slowly.
17. *Do* move the line of code that *changes* the arrow, so the arrow will continually display the changing magnetic field due to the moving particle.
18. Run the program. You will need to adjust the scalefactor.

**CHECKPOINT VP3: Have an instructor check your work, for credit.**

### 4. Adding observation locations

Add three additional observation locations, in locations with the same x-coordinate, but different y and z coordinates, so that the four locations are at the corners of a square surrounding the path of the particle. Your finished program should display the magnetic field vectors simultaneously at all these locations as the particle moves.

Set the scalefactor to a value that makes the arrows representing magnetic field large, but not so large that they extend offscreen.

### 5. Turn in your program in WebAssign.

Read the instructions in WebAssign carefully. You may be asked to change some values before turning in your program.