Physics 115: Contemporary Physics III

Spring 2013

Programming Assignment 4

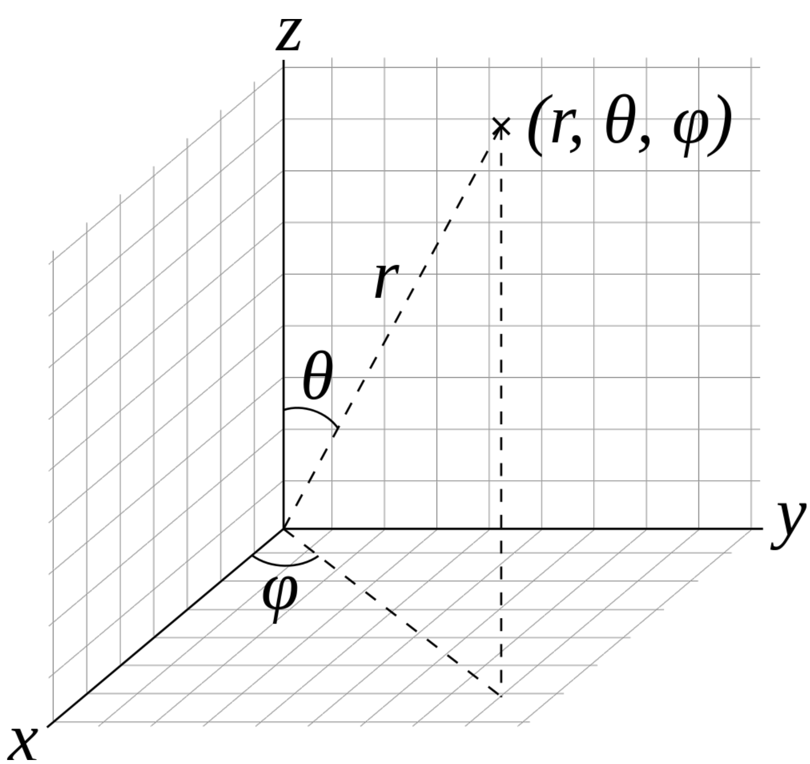
Due June 5, 2013

Electromagnetic Radiation

In this final recitation assignment you will produce electromagnetic radiation! As you remember from class, an accelerating charged particle generates electromagnetic radiation. You will plot the radiation as it propagates outward.

How are you going to do this? Take a charged sphere and give it an acceleration for an brief time, of duration τ, at $t = 0$. The particle doesn't actually need to move, but feel free to add that if you like. Your program will generate pairs of $\mathbf{E}$ and $\mathbf{B}$ vectors all around the particle. As the pulse of radiation from the particle expands into space, the nonzero pairs will lie on a sphere that increases in radius with time—pairs will "light up" as the pulse pases them.

Spherical polar coordinates will be most convenient for our purposes. This is another coordinate system commonly used to describe 3-D space. As illustrated below, instead of the familiar Cartesian coordinates (x, y, z), we use instead (r, θ, ϕ), where r is distance from the origin, θ is angle with the z axis, and ϕ is angle in the $x - y$ plane around the z axis.



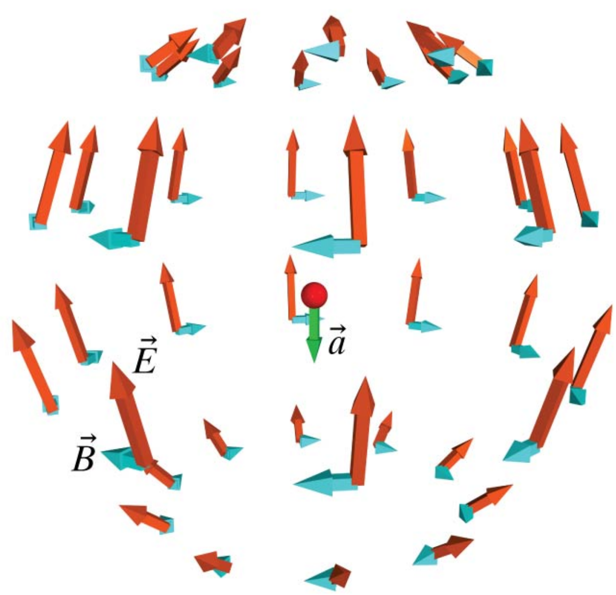
The connection with Cartesian coordinates (needed for vpython) is

$$\begin{aligned} x &= r \sin\theta \cos\phi \\ y &= r \sin\theta \sin\phi \\ z &= r \cos\theta. \end{aligned}$$

Using steps $\delta\theta = \pi/6$ and $\delta\phi = \pi/6$, place pairs of **E** and **B** vectors spherically around the particle. Use a minimum radius of $r = 3 \times 10^{-12}$ m and radial steps of $\delta r = 3 \times 10^{-12}$ m, up to (say) $r = 3 \times 10^{-11}$ m. Experiment to see what looks best! Set the radius of your particle small enough for these values to be practical (10^{-13} m or so). Note that these radii correspond to light travel times of $r/c = 10^{-20}, 2 \times 10^{-20}, 3 \times 10^{-20}$ s, etc.

The magnitudes of the fields at $t = 0$ will all be zero, since the charged particle was not accelerating for $t' < 0$. For $t > 0$ you will set each vectors length to represent the field at that location and time (suitably scaled so we can see what's going on). You'll find the equation for calculating the **E** field of the radiation in Chapter 24 of the text book. The book does not give an explicit expression for the **B** field, but it does tell you how to compute it from **E**. Note that we are not interested in the electric Coulomb field of the stationary charged particle, just the radiative field. Once you give the particle a kick, the radiation field propagates outward at speed c. Since the fields at time t and distance r from the source depend on the acceleration at the "retarded" time $t' = t - r/c$, only those points for which $a(t') \neq 0$ will light up.

(a) Set the acceleration of the particle to be $\mathbf{a} = (0, -2 \times 10^{17}, 0)\,\mathrm{m/s^2}$ for $0 \leq t < \tau$, with $\tau = 1.5 \times 10^{-20}$ s. Display the radiation field for $t = 0, 10^{-20}, 2 \times 10^{-20}, 3 \times 10^{-20}$ s, etc. Note that the width of the pulse is such that only one shell of vectors will be nonzero at any instant. As time increases, it should look as though the sphere of nonzero field vectors is expanding. At any instant, it should look something like the following image from the book:



The color scheme you use can be different, just make sure the E and B fields are easily distinguishable (don't use black and dark grey, for example).

(b) Repeat part (a) for a short sinusiodal pulse of radiation corresponding to $\mathbf{a} = (0, -2 \times 10^{17}, 0)\cos\omega t\ \mathrm{m/s^2}$, with $\omega = 3.1 \times 10^{20}$ rad/s, for $0 \leq t < \tau$, with $\tau = 4 \times 10^{-20}$ s. This corresponds to the electron oscillating at gamma-ray frequencies for approximately two periods. Display the radiation field for $t = 0, 10^{-20}, 2 \times 10^{-20}, 3 \times 10^{-20}$ s, etc. In this case, the width of the pulse will span more than one shell, so several spheres of nonzero field vectors should be seen. [Note that if you have written your program correctly, all you should need to change relative to part (a) are the details of the function computing the acceleration.]

Optional Extra

Python classes offer a particularly convenient way of implementing the creation and management of complex objects such as electric/magnetic vector pairs. For example, you could create a radiation class which consists of two arrows, one for **E** and one for **B**. See

`http://www.freenetpages.co.uk/hp/alan.gauld/tutclass.htm` or

`http://en.wikibooks.org/wiki/A_Beginner's_Python_Tutorial/Classes` for useful introductory tutorials on writing a class. Your class will need to take in r, θ, and ϕ. A member function (a function that exists only in the class) should handle the conversion from spherical to Cartesian coordinates and place the two arrows at the proper location. Your class should look something like the following:

```
def radiation:
    def __init__(self, r, t, p):    # double underscores, note
        self.earrow = arrow(color=color.yellow)
        self.radius = r
    def place(self):
        x = r*cos(theta)*sin(phi)
        self.earrow.pos = vector(x,y,z)
```

`self` is a special variable that exists only inside a class. It refers to the particular instance of the class that you are currently working on. In your main program you will initialize objects of type radiation, and then place them, e.g.

```
    rad_arrow = radiation(r,t,p)
    rad_arrow.place()
```

Whenever you need to access any of those arrows later, say to modify their axes, you can do something like:

```
    rad_arrow.earrow.axis = vector(Ex,Ey,Ez)
```

where `Ex`, `Ey`, `Ez` are the proper values for the electric field at that point. You'll need to add a magnetic vector to the class, too.

You will need to make a large amount of radiation objects, so why not make a list of them?

```
    rad_arrows = []
    for i in range(0,10): # example numbers
        rad_arrows.append(radiation(r, theta, phi))
        rad_arrows.place()
```

Accessing a particular earrow is exactly what you would guess:

```
    rad_arrows[3].earrow.axis = vector(x,y,z)
```

With a little practice, using classes is a great way to simplify both the writing and the maintenance of your programs!