This code shows how to make an array of arrows for ELECTRIC field lines from a point source. The code includes a saturation level to keep arrows from getting overly large and ruining the appearance in the simulation. I also threw in a bonus for loop showing how you can modify every arrow in the list, after the initial creation, very easily.

To verify you truly understand the code, generalize this code to create a 3D set of arrows instead of just 2D. Then modify it to handle a current source (instead of a charge source).

	Electric Field	**Magnetic Field**
Type of Source	Charge	Current
Symbol for Source	dq	$I\ d\vec{s}$
Contribution to Field	$d\vec{E} = \frac{k\ dq\ \vec{r}}{r^3}$	$d\vec{B} = \frac{\mu_0}{4\pi} \cdot \frac{(I\ d\vec{s}) \times \vec{r}}{r^3}$

```
GlowScript 2.7 VPython
k = 8.99e9
q1 = -1.0e-9
x = -5
x_max = 5.01
dx = 1
y = -5
y_min = -5
y_max = 5.01
dy = 1
sat_lev = 1          #max allowable size of an arrow
list_of_arrows = []    #creates empty list for arrows


if q1>0:
  c1=vec(1, 0, 0)
else:
  c1=vec(0, 0.6, 1)
ball1 = sphere(pos=vec(0, 2, 0), radius=0.3, color=c1)

while x<x_max:
  while y<y_max:
    POI = vec(x, y, 0)
    r1 = POI-ball1.pos

    if mag(r1) < 0.1:
      E_field = vec(0, 0, 0)                    #this field would be infinite...ignore it
    else:
      E_field = k*q1*r1/mag(r1)**3

    if mag(E_field)>sat_lev:
      E_field=hat(E_field)*sat_lev

    E_arrow = arrow(pos=POI, axis=E_field, color=vec(1, 1, 1))
    list_of_arrows.append(E_arrow)
    y += dy
  y = y_min
  x += dx

for myarrow in list_of_arrows:
  rate(20)
  myarrow.color = c1
```

This code shows how to use ELECTRIC fields from two point sources acting on a third. It also shows one way to incorporate a saturation level to reduce overly large arrows from ruining the appearance of the simulation. Code continues on next page as well.

```
GlowScript 2.7 VPython
from visual import *
#scale_arrow is a number that will allow you to change the size of force arrows
scale_arrow=1

#saturation level...arrows larger than 5 are capped at 5
sat_lev = 5

#q's are charges in coulombs
q1 = 1.0e-4   #0.1 mC
q2 = -2.0e-4  #0.2 mC
q3 = -2.0e-4

#m's are masses in kilograms
m1 = 1
m2 = 2000
m3 = 2000

#the coulomb constant is k
k = 8.99e9

#r's are initial locations of charges
r1 = vec(10, 0, 0)
r2 = vec(-5, 0, 0)
r3 = vec(5, 0, 0)

#v's are initial momentums of charges
p1 = vec(0, 5.5, 0)
p2 = vec(0, 0, 0)
p3 = vec(0, 0, 0)

#initialize while loop parameters
t = 0
delta_t = 0.01
sim_speed = 0.5
#place a ball of appropriate color at each position
#make positive charges red and negative charges blue-green
#notice the smaller mass, m1, has a smaller radius
if q1>0:
   c1=vec(1, 0, 0)
else:
   c1=vec(0, 0.6, 1)
ball1 = sphere(pos=r1, radius=0.3, color=c1)

if q2>0:
   c2=vec(1, 0, 0)
else:
   c2=vec(0, 0.6, 1)
ball2 = sphere(pos=r2, radius=1, color=c2)

if q3>0:
   c3=vec(1, 0, 0)
else:
   c3=vec(0, 0.6, 1)
ball3 = sphere(pos=r3, radius=1, color=c3)
```

```
#determine the initial Coulomb force q2 exerts ON q1 (the small ball)
#recall F=k*q*Q*r_hat/r^2  is the same thing as F=k*q*Q*r_vector/r^3
r_2to1=ball1.pos-ball2.pos   #final position minus initial position
F_2on1 = k*q1*q2*r_2to1/mag(r_2to1)**3
print(F_2on1)   #this print statement allows you to check the math

#determine the initial Coulomb force q3 exerts ON q1 (the small ball)
r_3to1=ball1.pos-ball3.pos   #final position minus initial position
F_3on1 = k*q1*q3*r_3to1/mag(r_3to1)**3
print(F_3on1)   #this print statement allows you to check the math

#determine initial NET force on q1
F_NETon1 = F_2on1 + F_3on1
print(F_NETon1)   #this print statement allows you to check the math

#draw arrows representing the NET force
#scale_arrow, defined earlier, allows one to change size of all force arrows
F_arrow_NETon1 = arrow(pos=r1, color=0.5*(c2+c3))
print(scale_arrow*mag(F_NETon1))
if scale_arrow*mag(F_NETon1) > sat_lev:
   F_arrow_NETon1.axis=hat(F_NETon1)*sat_lev
else:
   F_arrow_NETon1.axis=F_NETon1*scale_arrow

#loop uses force ON ball1 to update position and momentum of ball1
#for now assume balls 2 & 3 remain motionless as m2=m3 >> m1
#WARNING: code produces unphysical results if ball1 moves inside ball 2
while t<30:
   rate(sim_speed/delta_t)
   p1 = p1 + F_NETon1*delta_t                    #momentum update equation
   ball1.pos = ball1.pos + (p1/m1)*delta_t       #position update equation
   r_2to1=ball1.pos-ball2.pos                    #update vector between 1 & 2
   F_2on1 = k*q1*q2*r_2to1/mag(r_2to1)**3        #update force 2 exerts on 1
   r_3to1=ball1.pos-ball3.pos                    #update vector between 1 & 3
   F_3on1 = k*q1*q3*r_3to1/mag(r_3to1)**3        #update force 3 exerts on 1
   F_NETon1 = F_2on1 + F_3on1                    #update NET force on 1
   if scale_arrow*mag(F_NETon1) > sat_lev:
      F_arrow_NETon1.axis=hat(F_NETon1)*sat_lev
   else:
      F_arrow_NETon1.axis=F_NETon1*scale_arrow
   F_arrow_NETon1.pos = F_arrow_NETon1.pos + (p1/m1)*delta_t      #update F_NETon1 arrow pos
   t = t + delta_t                                               #increment the time
```