In [ ]:

```
from vpython import *
# Charges in a conductor reaching equilibrium
# Adapted from "Stars Interacting Gravitationally" by Bruce Sherwood

scene = canvas()
scene.width = scene.height = 600
scene.background = color.black

# Display text below the 3D graphics:
scene.title = "Charges in a Conductor"

scene.caption = """Ctrl-drag to rotate "camera" to view scene. Alt-drag to zoom.
Refresh the web page to re-execute with different (random) initial conditions."""


# Parameter values
N = 200    # Number of individual charges
Q = 5e-6   # Net charge, in Coulombs
m = 1e-3   # Mass of each charge, in kg
R = 0.10   # Radius of conducting sphere, in meters
dt = 0.001 # Time step, in seconds
K = 8.99e9 # Coulomb constant

q = Q/N    # Charge for each individual charge

scene.range = 1.5*R
scene.forward = vec(-1,-1,-1)

sphere(radius = R, color=color.white, opacity = 0.5)

charges = [] # Empty array of charges, to be filled below

# Create charges with random initial positions, initially at rest:
for i in range(N):
    position = R/sqrt(3) * vec.random()
    charge = sphere(pos=position, radius = 0.01*R, color=color.red) # Random position
    charge.velocity = vec(0,0,0)   # Initially at rest
    charges.append( charge )

# Function to compute forces & update velocities
def computeForces():
    global charges
    N = len(charges)
    for i in range(N):
        charge_i = charges[i]
        F_net = vec(0,0,0)
        r_i = charge_i.pos
        for j in range(N):
            if i == j: continue # A charge doesn't interact with itself
```

```
            charge_j = charges[j]

            r_j = charge_j.pos
            r_vector = r_i - r_j
            r = mag(r_vector)
            F = K*q*q/r**2 * (r_vector/r)
            F_net = F_net + F
        a = F_net / m      # Acceleration of charge i
        if mag(a) > 1000:  # In case of a huge acceleration...
            a = 1000 * a / mag(a) # Rescale acceleration to smaller value
        charge_i.velocity = charge_i.velocity + a*dt # Update velocity of charge
i


# (COMPLETED) FUNCTION THAT NEEDS TO BE FILLED IN BY THE STUDENTS:
def computeEfield(P):
    ''' Computes the total electric field at point P, which is a 3D vector.
    YOU WILL NEED TO COMPLETE THIS FUNCTION!! '''

    global charges
    N = len(charges)

    # E_net will be computed from a summation, so it is first set to zero
    E_net = vec(0,0,0)

    # Loop through all charges in order to compute the net E field
    for charge in charges:
        r_vector = P - charge.pos # vector between charge & point P
        r = mag(r_vector)         # "r" is the magnitude of the r vector
        E = 1e-6 * K*q/r**2 * (r_vector/r) # The E field from this ONE charge
        E_net = E_net + E         # Computes the running sum, E_net
    return E_net # This sends the computed value back to the main loop

P = vec(0.5, 0, 0)

t = 0 # Start the timer at t = 0

while True:
    rate(100) # Sets maximum frame rate to 100 frames per second

    # Compute all forces on all charges & update velocities
    computeForces()

    # Having updated all velocities, now update all positions
    for charge in charges:
        charge.pos = charge.pos + dt * charge.velocity
        d = mag(charge.pos) # Distance from center of sphere to charge
        if d > R: # If charge would have LEFT the conductor
            charge.pos = charge.pos * R / d # Bring back to edge

    t = t + dt              # Update the value of time
    E_net = computeEfield(P) # After updating positions, compute E using your fu
nction
```

```
    # Print the numerical value of |E| in microCoulombs
    print('At P =', P, 'meters,  |E| =', mag(E_net), 'N/uC')
```

In [ ]:

```

```