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 Rectangular 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 = 8       # Number of individual charges
Q = 5e-6    # Net charge, in Coulombs
L = 0.10    # Length of conducting cube, 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*L
scene.forward = vec(-1,-1,-1)

box(pos=vector(0,0,0), axis=vector(1,0,0), size=vector(L,L,L), 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 = L/2 * vec.random()
    charge = sphere(pos=position, radius = 0.01*L, color=color.red) # Random pos
ition
    charge.velocity = vec(0,0,0)   # Initially at rest
    charges.append(charge)

# Function to compute forces & update positions
def computeForces():
    global charges
    N = len(charges)
    for i in range(N):
        charge_i = charges[i]
        F_net = vec(0,0,0) # Will sum up force. First set to zero.
        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

        # Will use the limit of large friction, where force --> displacement
        displacement =  F_net
        if mag(displacement) > L/100:  # Don't allow a huge displacement:
            displacement = (L/100) * displacement / mag(displacement)

        # Update the position of the charge using the displacement above:
        charge_i.pos = charge_i.pos + displacement


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

    E_net = vec(0, 0, 0)

    # PUT YOUR LINES OF CODE HERE TO COMPUTE THE E-FIELD

    return E_net # This sends the computed value back to the main loop


P = vec(1, 0, 0) # UPDATE THIS POSITION VECTOR #

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 positions
    computeForces()

    # Don't let the charges leave the conductor
    for charge in charges:

        if charge.pos.x < -L/2:
            charge.pos.x = -L/2
        if charge.pos.x > +L/2:
            charge.pos.x = +L/2

        if charge.pos.y < -L/2:
            charge.pos.y = -L/2
        if charge.pos.y > +L/2:
            charge.pos.y = +L/2
```

```
        if charge.pos.z < -L/2:

            charge.pos.z = -L/2
        if charge.pos.z > +L/2:
            charge.pos.z = +L/2

    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')
```