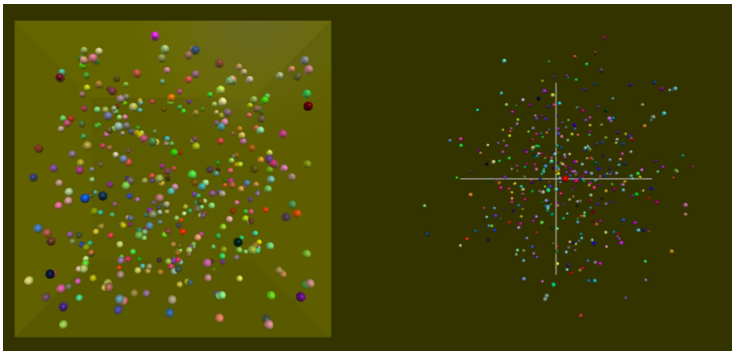
VP8 Drift Velocity

In the program, *N*=400 charged particles, each with mass *m* and charge *q*, are in a cubic box of length 2*L*. The velocity of the particles has a speed distribution with root mean square = *vrms*, but in random directions. With the periodic boundary condition, each particle hitting a wall will disappear and reappear on the opposite wall with the same velocity. The two different displays, the position display (called "real space") and velocity display (called "velocity space"), together form the "phase space" of the entire system. In this "phase space", the states (position and velocity) of all particles are completely represented. In the program, the drift velocity *vd,* which is the velocity averaged over all particles over all time, is found and shown in the velocity display as the large red ball.

Homework:

Within each time step *dt*, each particle has a probability (*prob*) to encounter a "collision" (Note: not a real collision against any other charged particle, but just a collision against some virtual particles). After the collision, the velocity of the particle is randomly distributed with the same root mean square = *vrms*, and in random directions. There is also an electric field *E* which accelerates all particles. For every 2000*dt, print the collision time: *tau* = (total time *t*N*) / (total collision number of all particles of all time), the simulated drift velocity *vd*, the theoretical drift velocity *q*E*tau/m*. Are these two values close?



```
from vpython import *
import numpy as np

prob = 0.005
N, L = 400, 7E-9/2.0
E = 1000000
q, m, size = 1.6E-19, 1E-6/6E23, 0.1E-9        #artificial charge particle
t, dt, vrms = 0, 1E-16, 10000.0
atoms, atoms_v = [],[]

#initialization
scene = canvas(width=600, height=600,align = 'left', background=vector(0.2,0.2,0))
scenev = canvas(width=600, height=600, align = 'left', range = 4E4, background=vector(0.2, 0.2,0))
container = box(canvas=scene, length = 2*L, height = 2*L, width = 2*L, opacity=0.2, color = color.yellow )

pos_array = -L + 2*L*np.random.rand(N,3)
X, Y, Z = np.random.normal(0, vrms, N), np.random.normal(0, vrms, N), np.random.normal(0, vrms, N)
v_array = np.transpose([X, Y, Z])

def a_to_v(a):    # array to vector
    return vector(a[0], a[1], a[2])

for i in range(N):
    atom = sphere(canvas=scene, pos=a_to_v(pos_array[i]), radius = size, color=a_to_v(np.random.rand(3,1)))
    atoms.append(atom)
    atoms_v.append(sphere(canvas=scenev,pos=a_to_v(v_array[i]), radius = vrms/30, color=a_to_v(np.random.rand(3,1))))

# the average velocity and two axes in velocity space
vd_ball = sphere(canvas=scenev,pos=vec(0,0,0),radius = vrms/15, color=color.red)
x_axis = curve(canvas=scenev, pos=[vector(-2*vrms,0,0), vector(2*vrms,0,0)], radius=vrms/100)
y_axis = curve(canvas=scenev, pos=[vector(0,-2*vrms,0), vector(0,2*vrms,0)], radius=vrms/100)
vv = vector(0, 0, 0)                    # for calculating the average velocity
total_c = 0                             # the total number of collisions

while True:
    t += dt
    rate(10000)

    v_array[:,0] += q*E/m*dt
    pos_array += v_array*dt             # calculate new positions for all atoms
    outside = abs(pos_array) >= L
    pos_array[outside] = - pos_array[outside]
```

```
# handle collision here

vv += a_to_v(np.sum(v_array,axis = 0)/N)

if int(t/dt)%2000 == 0:
    tau = 0                    # need to be modified
    print(tau, vv/(t/dt), q*E*tau/m)
vd_ball.pos = vv/(t/dt)

for i in range(N): atoms_v[i].pos, atoms[i].pos = a_to_v(v_array[i]), a_to_v(pos_array[i])
```