Week 9(3/3->3/11)

The results from last week, week 8, gave unformatted data to see if the physics involved in the calculation appear reasonable. Originally the data was formatted for easy reading by a human, as in a structure like:

Time: current time

Particle: Particle index
Position: (x, y, z)
Velocity: (x, y, z)
Force: (x, y, z)
.
.
.

In order for a program like matlab to read in the data, the format needed to be changed to something like:

Time	X	Y	Z	X	Y ...
0.0	0.5	0.5	0.0	-0.5	-0.5...
0.1	0.49	0.49	0.0	-0.49	-0.49...

This is fairly basic C++ I/O operation so I won't go over this, but the major issue is when and where to output the data to a file. This has to be done after the calculation, but before the display. This means that the arrays are dumped every calculation cycle which is not efficient but it works for this test.

The simulation setup I chose to go with is one of the most simple while still requiring gravity. This involves two particles equal distance apart as well as height and velocity. The setup for this simulation is added to a case statement in the reset() function in particlesystem.cpp as shown below:

```
case CONFIG_TEST:
{
	int p=0;
	int v=0;
	m_hPos[p++] = -0.5f;
	m_hPos[p++] = 0.5f;
	m_hPos[p++] = 0.25f;
	m_hPos[p++] = 1.0f; // radius

	m_hPos[p++] = 0.5f;
	m_hPos[p++] = 0.5f;
	m_hPos[p++] = 0.25f;
	m_hPos[p++] = 1.0f; // radius

	m_hVel[v++] = 0.01f;
	m_hVel[v++] = 0.0f;
	m_hVel[v++] = 0.0f;
	m_hVel[v++] = 0.0f;
```

```
        m_hVel[v++] = -0.01f;
        m_hVel[v++] = 0.0f;
        m_hVel[v++] = 0.0f;
        m_hVel[v++] = 0.0f;
}
```

The coordinate system for this program has its origin at the center of the floor of the box. The 'y' direction is height while the 'z' direction is depth from the original viewing position. For the test all constants are left as the default values except the time-step is set to 0.01.

```
float timestep = 0.1f;
float damping = 1.0f;
float gravity = 0.0003f;

float collideSpring = 0.5f;;
float collideDamping = 0.02f;;
float collideShear = 0.1f;
float collideAttraction = 0.0f;
```

The path of the particles is shown in figure 1, 2, 3, and 4.

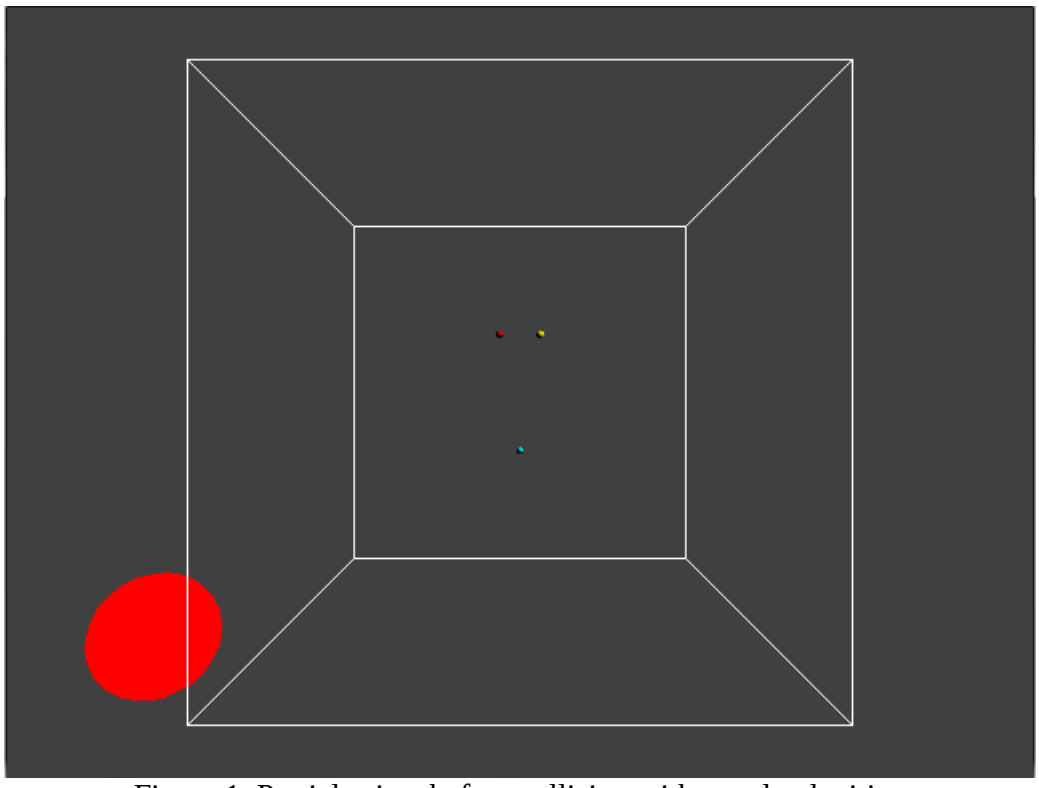

Figure 1, Particles just before collision with equal velocities

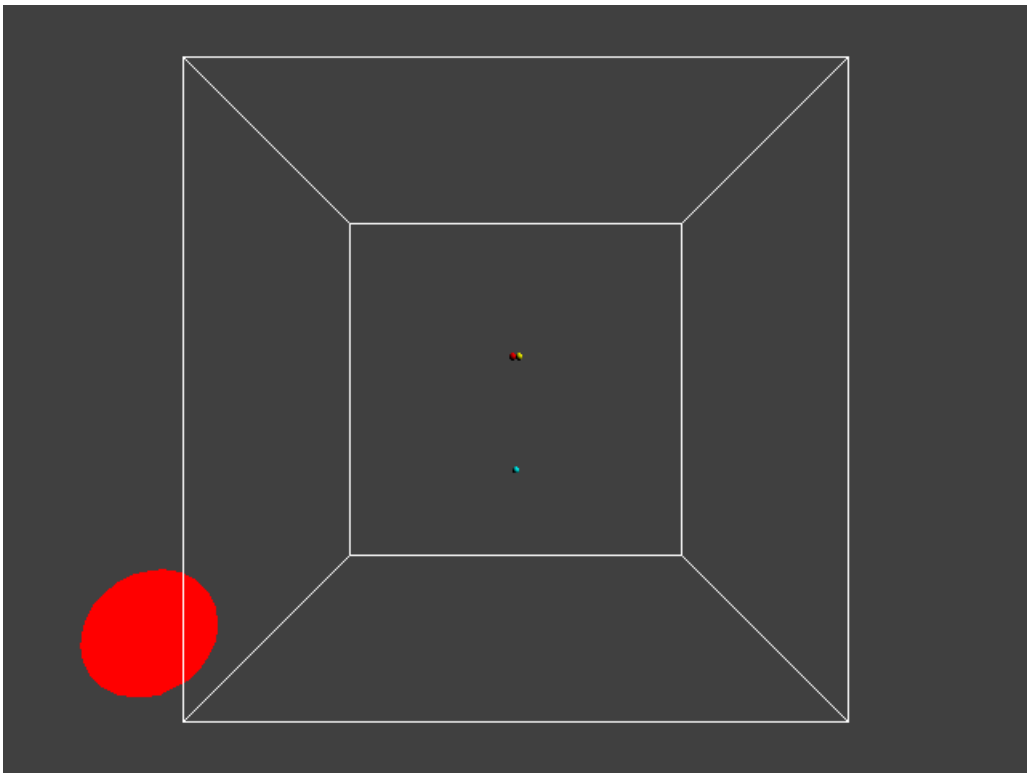

Figure 2, Particles at collision point

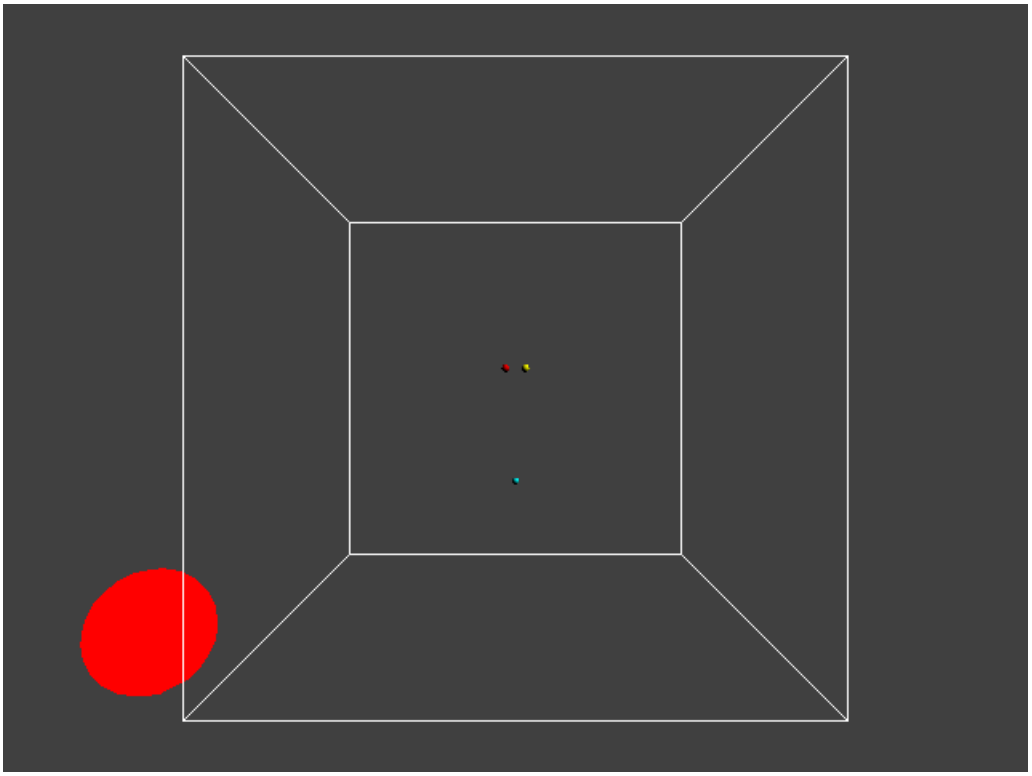

Figure 3, Particles just after collision at which the simulation ends

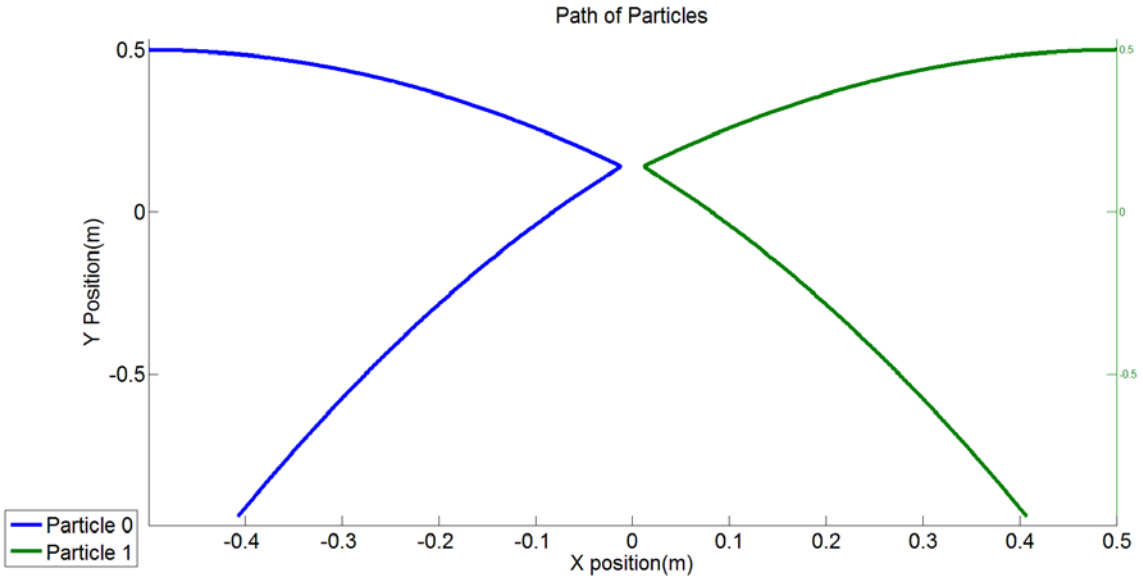

Figure 4, data retrieved from simulation showing the path of the particles plotted in MATLAB

The next thing I looked at was the velocity and force data which were also dumped at the same time as the position data. I separated these data sets into three files each containing data for each particle. The result of the velocity in x are shown in figure 5, velocity in y shown in figure 6, and the total force on particle 0 are shown in figure 7.

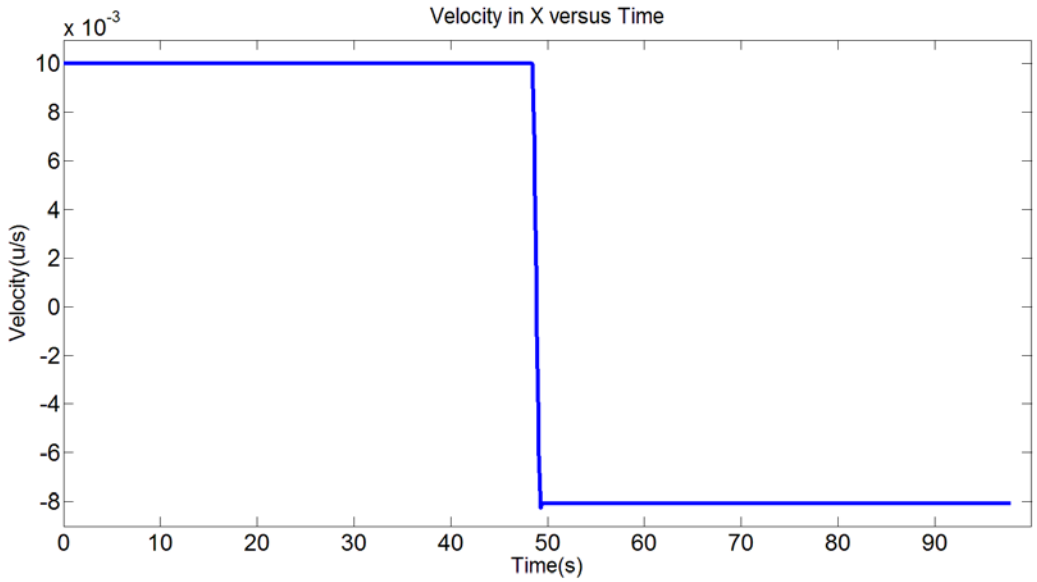

Figure 5, Velocity in the x direction reverses and drops in magnitude due to collision damping

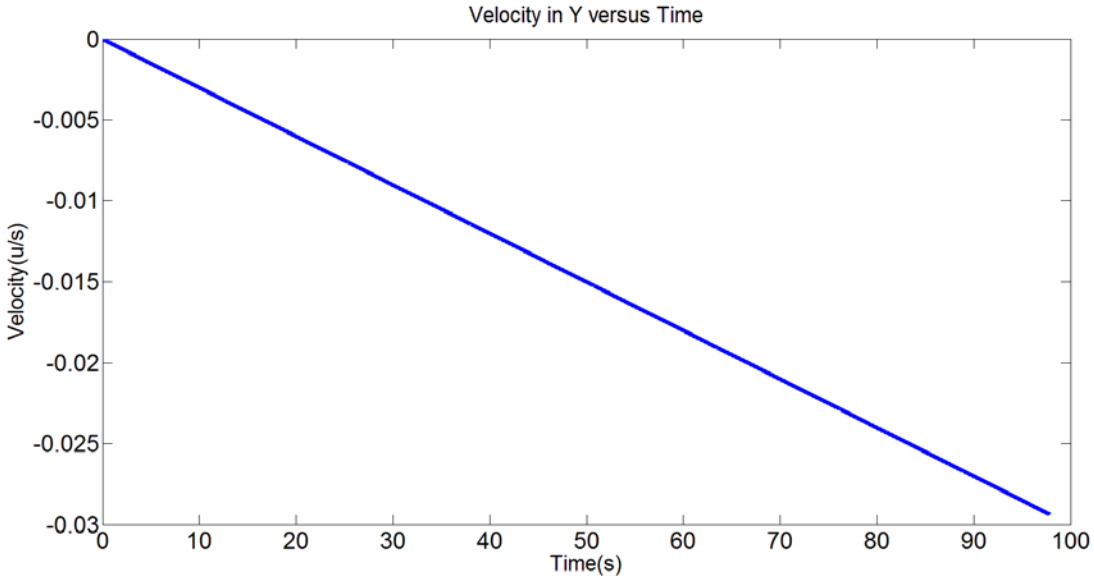

Figure 6, velocity in the vertical direction increases linearly as expected

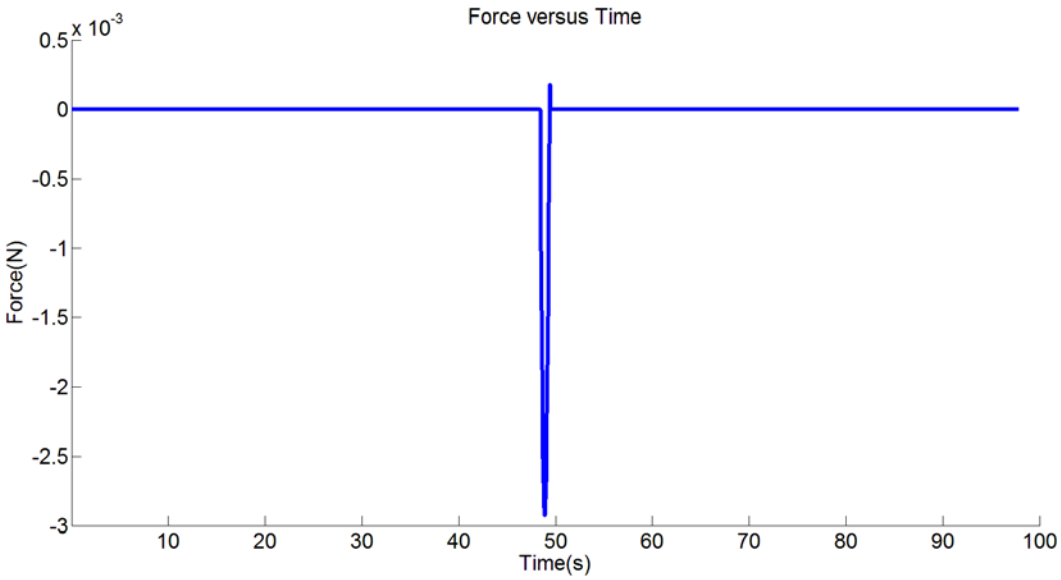

Figure 7, collision force on particle 0 showing dip only during collision

What's Next?

There are three options that I can think of at this point:

1. Continue with modification of this program
2. Make more changes to this as a sort of test then move onto writing a program from scratch
3. Start working on a new program with what I've learned, using this program as a launch-pad

I like the third option better because I can write something and better document the program compared to starting with someone else code. I wouldn't be starting off like I would have at the start of the this quarter. I would be able to create a GUI for easier use of the program, so if you want to test a specific shape you could import this, or set specific properties of the polymer. We can work out exactly what you think should be done.

$$dist = length(relPos)$$
$$\overrightarrow{\boldsymbol{relpos}} = \overrightarrow{\boldsymbol{posB}} - \overrightarrow{\boldsymbol{posA}}$$

$$\overrightarrow{\boldsymbol{norm}} = \frac{\overrightarrow{\boldsymbol{relPos}}}{dist}$$

$$\overrightarrow{\boldsymbol{relVel}} = \overrightarrow{\boldsymbol{velB}} - \overrightarrow{\boldsymbol{velA}}$$
$$\overrightarrow{\boldsymbol{tanVel}} = \overrightarrow{\boldsymbol{relVel}} - \overrightarrow{\boldsymbol{norm}} \cdot (\overrightarrow{\boldsymbol{relVel}} \bullet \overrightarrow{\boldsymbol{norm}})$$

$$x = collideDist - dist$$
$$\zeta = Damping\ ratio\ (0 \rightarrow 1.0)$$
$$k = spring\ constant\ (0 \rightarrow 1.0)$$
$$\tau = shear\ constant\ (0 \rightarrow 0.1)$$
$$\alpha = attraction\ (0 \rightarrow 0.1)$$
$$g = gravity\ constant\ (0.001 \rightarrow 0.003)$$

$$\overrightarrow{\boldsymbol{force_{i\ on\ 0}}} = -kx \cdot \overrightarrow{\boldsymbol{norm}} + \zeta \cdot \overrightarrow{\boldsymbol{relVel}} + \tau \cdot \overrightarrow{\boldsymbol{tanVel}} + \alpha \cdot \overrightarrow{\boldsymbol{relPos}}$$

$$Total\ Force\ on\ Particle_0 = \sum_{i=1}^{N} \boldsymbol{force_{i\ on\ 0}}$$

$$\vec{\boldsymbol{v}}_{\boldsymbol{t+\Delta t}} = GD(\vec{\boldsymbol{v}}_{\boldsymbol{t}} + Gdt) + \sum_{i=1}^{N} \boldsymbol{force_{i\ on\ 0}}$$
$$\overrightarrow{\boldsymbol{pos_{t+\Delta t}}} = \overrightarrow{\boldsymbol{pos_t}} + GD(\overrightarrow{\boldsymbol{v_t}} \cdot \Delta t) + GD(G \cdot \Delta t^2)$$