Report

4. **Analysis of the fidelity of your chosen laws of motion – how well do they**
simulate the behavior you have chosen to model? Fidelity of chosen laws [3 marks].

a. Basic description (1 mark)

Rules that particles follow are first law and second law of newton's law of motion.

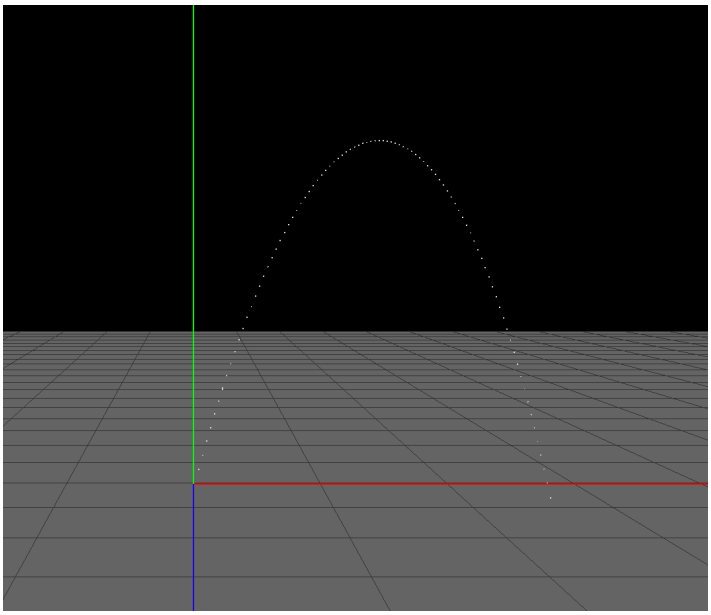
After particles ejected by the emitter, the only force on each particle is gravity which is in negative y direction. Therefore, velocity of particles in x and z direction won't change since there is not any component force in these directions, as indicated by newton first law. Newton's second law of motion states that for external force applied, the change in velocity depends on the mass of the object:

$$F = ma(1)$$

Thanks to external resultant force equivalent to gravity and its magnitude is mg, the velocity of particle changes in negative y direction and change rate of the velocity could be got as -g via the equation (1).

b. describe informally (eg without captured data) how accurately the laws of motion are simulated

From the video, we could see that the trajectory of each particle is a parabola which opens downward. Therefore, there must be a force on the object in negative y axis direction. Besides, the trajectory of each particle is symmetric along the y-axis direction (picture of trajectory of a typical particle shown below.



Therefore, there must not be any component forces on the x-z plane. Otherwise, the symmetry doesn't occur along that exact direction. As a result, movement of particles follow the newton's first law in x or z axis direction and the newton's second law in y axis direction. In conclusion, the laws of motion described in (a) are simulated quite accurately without observing actual data .

c. ability to describe with specific data/examples how accurately the laws of motion are simulated

velocities in x, y, z axis direction for a specific particle got from terminal are listed below:

```
vel x : 0.426064, vel y: 3.555556, vel z: 0.852128
vel x : 0.426064, vel y: 3.402275, vel z: 0.852128
vel x : 0.426064, vel y: 3.248993, vel z: 0.852128
vel x : 0.426064, vel y: 3.095712, vel z: 0.852128
vel x : 0.426064, vel y: 2.942431, vel z: 0.852128
vel x : 0.426064, vel y: 2.789150, vel z: 0.852128
vel x : 0.426064, vel y: 2.635869, vel z: 0.852128
vel x : 0.426064, vel y: 2.482587, vel z: 0.852128
vel x : 0.426064, vel y: 2.329306, vel z: 0.852128
vel x : 0.426064, vel y: 2.176025, vel z: 0.852128
vel x : 0.426064, vel y: 2.022744, vel z: 0.852128
vel x : 0.426064, vel y: 1.869462, vel z: 0.852128
```

We could see that the velocity in x and z direction of the particle doesn't change. Therefore, motion in x or z axis direction follows the newton's first law. The velocity change in y axis direction between frames is approximately 0.153. In general, fps is 62. Therefore, velocity changes per second could be 0,153 * 62 = 9.49 m. We could get that the error of the simulated gravity which is (9.81 – 9.49) / 9.81 = 0.033. That is quite small. So, the first of motion of law in x or z direction is completely simulated, and the second law of motion in y direction is almost simulated.

5. Efficiency of your approach to implementing your laws of motion – how
efficient is your computation of particle position between frames? [3marks]

a. 1 mark for any evidence that efficiency has been thought of;

Implementing Newton's first law of motion is quite easy. Assume that the Px and Pz are the particle's x and z coordinates in current frame. The velocity in X direction is Vx, the velocity in Z direction is Vz. The time between frames is deltaT, the Pnx and Pnz are the particle's x and z coordinates in the next frame.

$$Pnx = Px + Vx * deltaT$$
$$Pnz = Pz + Vz * deltaT$$

However, implementing the newton's second law of motion to calculate particle 's y coordinate is a little bit hard. Assume that Py, Pny refer to current y coordinate and y coordinate for the next frame respectively. Vy, Vny refers to current y velocity and the y velocity for the next frame. Then thanks to the newton's second law $F = ma, F = -mg. a = -g$. For the most accurate way to calculate the next frame y position, $Pny = Py + Vy * deltaT - \frac{1}{2} g * deltaT^2$. However, there may be a better way to implement such a distance formula in a more efficient way.

b. 1 mark if there is evidence of thinking about efficiency and what measures could be taken to improve it.

We could use $Pny = Py + Vy * deltaT$ to simplify the original formula $Pny = Py + Vy * deltaT - \frac{1}{2} g * deltaT^2$to enhance computation efficiency since fewer terms are needed to compute.

Reason: In general FPS is 60. Hence, deltaT is $\frac{1}{60}$. As a result, the term $\frac{1}{2} g * deltaT^2$ could be $-0.5 * 9.81 * \left(\frac{1}{60}\right)^2 \approx 1.36 \times 10^{-3}$ which are so small that we could ignore it. In conclusion, $Pny = Py + Vy * deltaT$ could be used in computation for y position to improve efficiency.

c. 1 mark for demonstrating that specific measures to improve maximize efficiency have been implemented.

The first method to maximize efficiency is by using multi-threading. By using multiple threads coordinates for many particles can be calculated concurrently. So, the computation efficiency is improved.
The second method may be using the vertex shader to make position of vertices calculated in GPU to save a large amount of CPU-GPU data transferring time.

6. Analysis of overall performance / rendering speed, and discussion of efficiencies implemented. [4 marks].

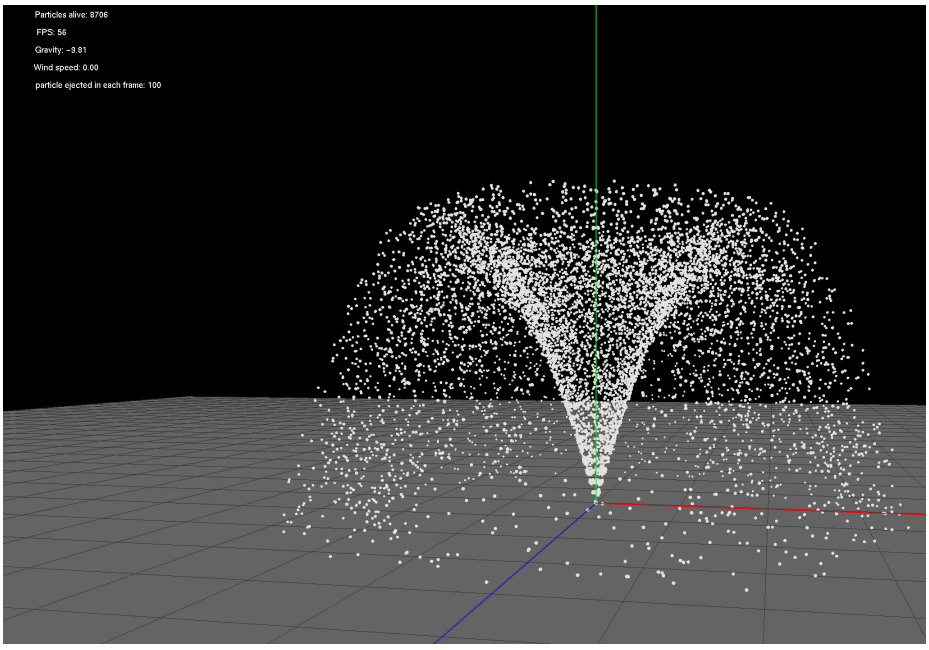
a. 1 mark for sensible discussion of how performance is bound by the limitations of data structures/CPU/use of GPU/transfer of data between CPU-GPU;

Bottleneck occurs when there is a limitation of how much data is being sent for processing or how much data could be processed at the same time. The occurrence of that can be concluded into three major cases. Firstly, the low performance of CPU. If the performance of the CPU is quite low compared with the performance of the GPU, to draw all the particles the GPU must wait for the data output from the CPU, thus making bottleneck appear. Secondly, the performance of the GPU is quite low compared with the performance of the CPU. In this case, even CPU is fast enough to process data for all the

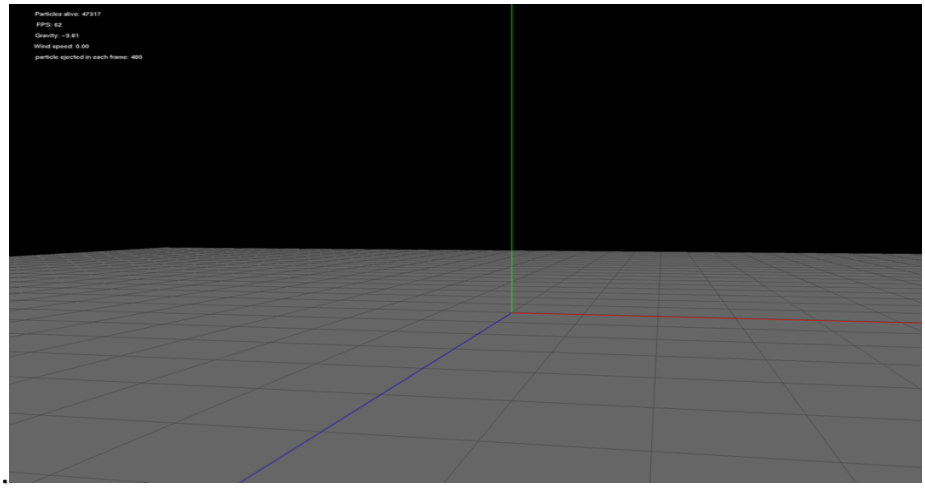
particles needed to draw in each frame, the GPU doesn't have the ability to render so many particles. Hence, bottleneck appears, and sudden FPS drop can be got. Thirdly, the low data transfer from CPU to GPU. Even the CPU and GPU are both fast enough, the transfer from CPU to GPU might be quite slow. GPU has to wait for the data transfer to render particles.
To analyze how performance is bound by these processes, the performance of them needs to be investigated when there is a sudden drop of FPS appearing.

b. 1 mark for evidence of exploring performance/rendering bounds by performing experiments.

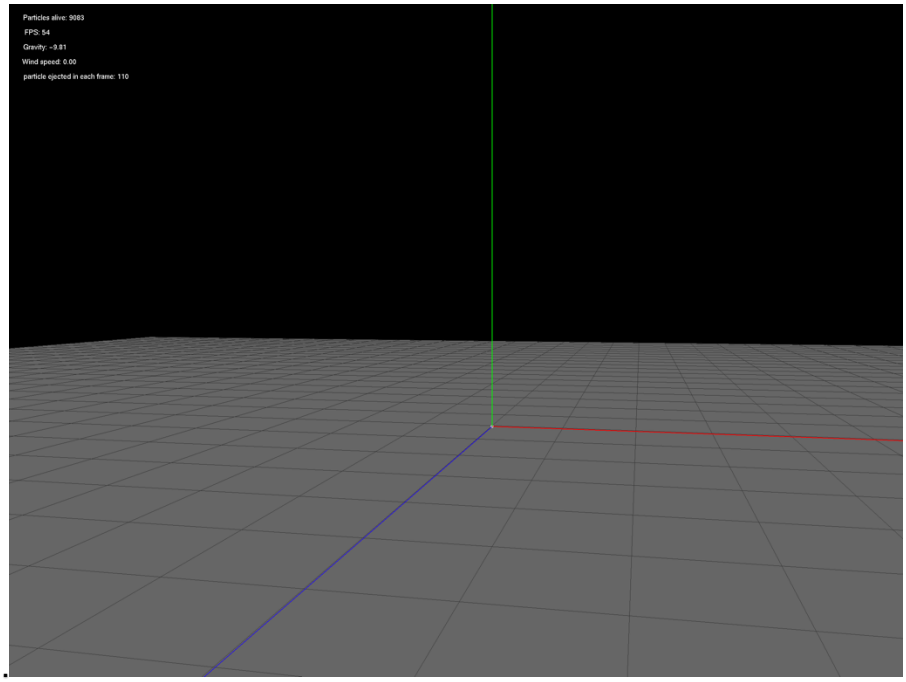
To make this experiment, the point where sudden drop in FPS needs to be found. Process: slow increase in number of points ejected in each frame untilfps drops



After find that point, we can comment out rendering processes of particles. Increase in number of the points to figure out if the bottleneck is caused by CPU



We could investigate that even number of particles ejected each frame is increase to 480, the fps remains. We could infer that the bottleneck is not caused by low performance of CPU.
Furthermore, comment out particles movement calculation(Just keep the rendering process)



We could get when number of particles ejected in each frame is 110, there is a sudden decrease of FPS.

c. 1 mark for analysis/discussion supported by some data.
Follow the steps in part(b). With more experiments and data records, we could get the following tables.
Investigate cpu performance (Comment out rendering process),

Particles ejected in each frame	30	60	90	120	150	180	210	240	270
FPS	62	62	62	62	62	62	62	62	62

Investigate gpu and data transfer from gpu to cpu performance (Comment out movement computation process).

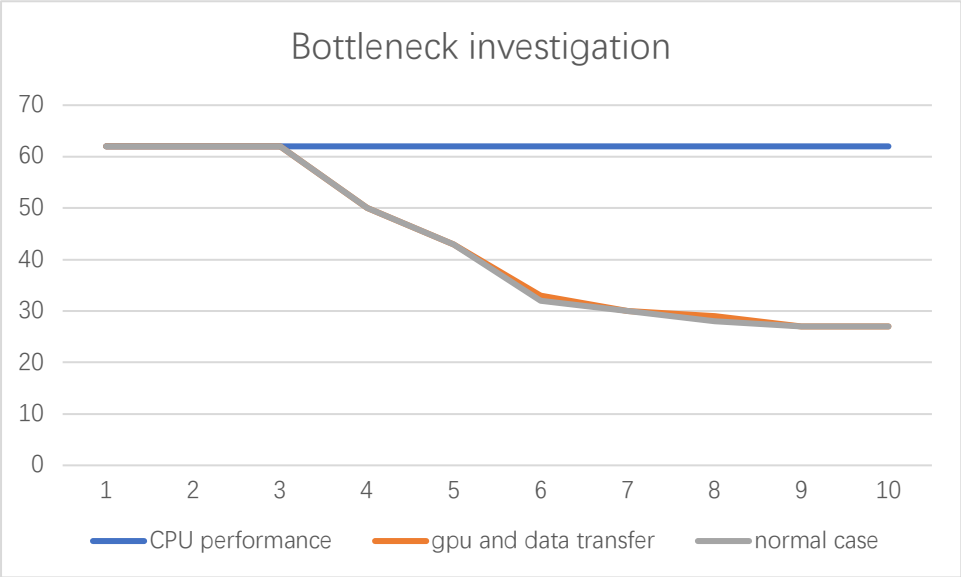
Particles ejected in each frame	30	60	90	120	150	180	210	240	270	300
FPS	62	62	62	50	43	33	30	29	27	27

Investigate the normal process

Particles ejected in each frame	30	60	90	120	150	180	210	240	270	300
FPS	62	62	62	50	43	32	30	28	27	27

From the table above, we could find that the FPS of normal process and gpu investigation process is almost the same. Therefore, we could confirm that the performance is not bound by CPU, it is bound by GPU or data transferring from CPU to GPU.

d. 1 mark for a rigorous performance analysis supported by graphs or other data visualizations.



From the line chart, the normal case and gpu and data transfer case almost overlap. Therefore, the performance is bound by CPU or data transferring from CPU to GPU.
To analyze which case indeed incurs the bottleneck. We could perform experiments for the gpu and data transfer case again with GPU history window open. The picture cut shows below.

We could see the use of gpu increases and almost reaches its maximum as particles emitted increase. In conclusion, the performance might be bound by GPU.