Particles System: Snow, Jack Jorgensen, 10372243

VIDEO: https://youtu.be/qz2SatL9low

References:
Snowflakes are hexagonal crystals:
“All snowflakes contain six sides or points owing to the way in which they form. The molecules in ice crystals join to one another in a hexagonal structure, an arrangement which allows water molecules - each with one oxygen and two hydrogen atoms - to form together in the most efficient way. “
https://www.metoffice.gov.uk/weather/learn-about/weather/types-of-weather/snow/snowflake

Efficiency implementation:

The method of implementation that the system uses to store and process the particles are through a particle structure. Within this structure the position of the particle in space is stored as an array of size 3. The velocity of the particle is also stored as an array of size three. The other property of the particle is the time to live stored as a GLuint.

The efficiency of the rendering and also the physics is O(n). The system works using a global acceleration structure to store the acceleration of the system, which is uniform throughout. Within this structure it has 3 GLfloats, wind in the x direction, wind in the z direction and gravity acting in the y vector.

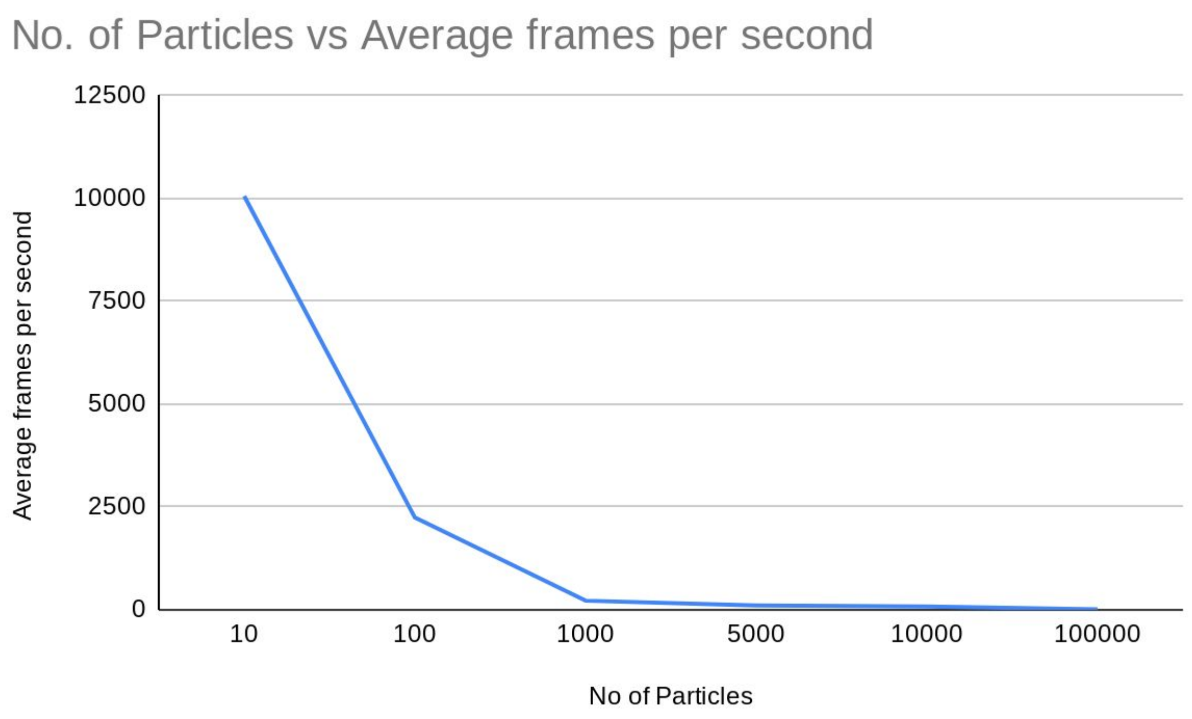
The particles are initialized, updated, killed and reincarnated the same way. A for loop cycles through the Particle array, and updates the velocity of the particle according to the global acceleration vector.

It then loops again updating the position of the particle.

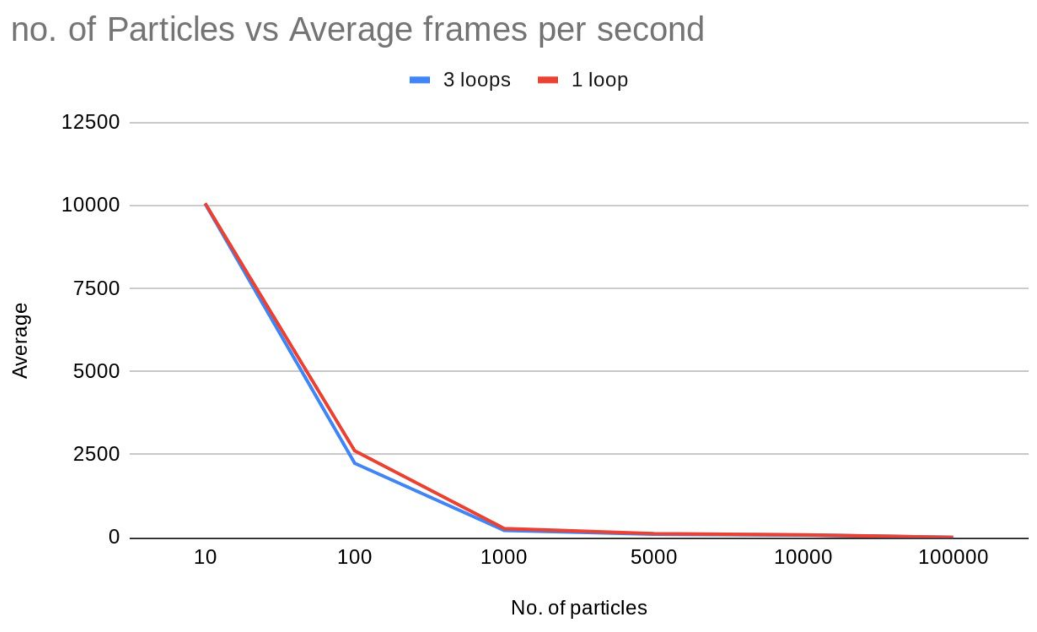
Finally it loops again drawing the particle.

This is a relatively inefficient system. Since for each frame the program loops through the list 3 times separately. For small scales (<6000 particles) this works quite well. The program runs at an average 120 frames per second (recorded using the frame rate program, developed by Toby Howard available on the Manchester CS wiki)

Upon reaching larger sizes, the program struggles and drops to 12 frames per second at 10000 particles

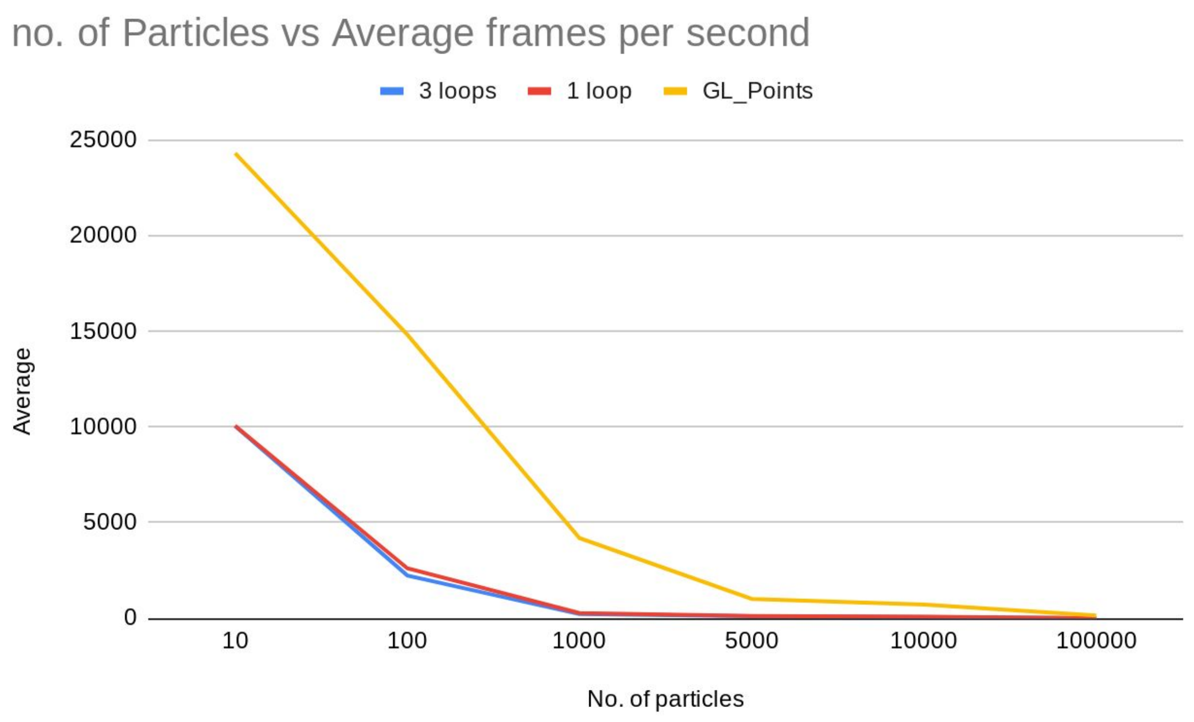


A more efficient method to use would be to update all the particles components in a single loop. I tried implementing this and measured the results



This however changes little as it still is running in linear time. Thus a matter of code layout preference rather than performance. The enabling and disabling of certain physics also did not have an effect of the performance

This leads to the conclusion that it is the implementation of the rendering that is causing the slowness. This is confirmed when we use GL_Points, which are rendered the same and are very lightweight



There are specific ways to measurably improve performance.
One notable suggestion by Tom McReynolds and David Blthye in the book “Advanced Graphics Programming Using OpenGL” (Published 2005, by Elsevier. ISBN 1-55860-659-9, available in Manchester Main Library 006.6633 OPE)
Is the use of Vertex arrays. This allows the particles to be “operated on efficiently” and “avoids data conversion overhead” Also the use of interleaved arrays, possibly using the function DrawArrays() would improve performance.

This could be further improved upon by the use of a VAO and VBO (vertex array/buffer object) For which openGL is optimized for.
The use of these data structures, specifically VAOs and VBOs, allow for a more efficient communication between the CPU and the GPU.

The limiting factor is that for each cycle of my program, the computer must process and update each point, and each variable in that structure. Only once it has processed the whole array is it then sent to the GPU to be rendered.
VAOs are Structures of arrays (SOA) rather than Arrays of Structures (AOS). This is a more efficient implementation to the GPU and is optimized for it. Since the VAO will store one or more VBO, for each of the properties of the particles. This allows the driver to take advantage of the known layout of the buffer.
(summarized from http://ogldev.atspace.co.uk/www/tutorial32/tutorial32.html and https://www.khronos.org/opengl/wiki/Tutorial2:_VAOs,_VBOs,_Vertex_and_Fragment_Shaders_(C_/_SDL))

Implementation of Physics:
The main physical properties implemented here are the wind and gravity.
Snow falls at a maximum of 2m/s. However due to the lack of anything else other than the snow there is no reference frame for the particles.
The speed of the particles is bounded, since snow does have a small terminal velocity (implemented after the video was taken please forgive me, line 99 of the code)
The physics calculations are very crude, the acceleration is added to the velocity and the velocity is added to the position of the particle, which position is a coordinate.
So while it looks like a vector, it isn't strictly coded to be such.

The snow also lies on the ground, as is common and then melts as expected by thermodynamics.

There is a lot that could be added to the system. Including a texture mapping on the ground that increases when the snow land, currents in the wind that causes the snow to 'spiral' (convection currents) rather than a 2 directional (xz plane) wind.

In terms of the actual effect produced, the laws of motion are pretty well represented. Falling objects increase their speed under gravity until they reach a terminal velocity due to air resistance. This effect is also present under the wind. Thus the particle will continue to accelerate until it hits this terminal velocity. Which currently implemented is 2 (omnidirectional as a magnitude)

The formulas for speed and acceleration are very standard:
Speed = initial speed + acceleration*time.
Since velocity and position are calculated each frame, time is 1, reducing the equation to a simple addition.
(lines 99-102 and 110-112 in the code)

This is essentially the same as (and is) a vector representation.

Finally:

One extra feature that was not talked about is that there are two particle sources. When rendering a check is performed. All the particles in the first half of the array are initalized at one point and the other at a separate predefined point (since clouds are mostly static or at least significantly present objects, not magical holes from whence snowflakes appear)
This is acceptable since all particles have a random time to live and so die at different times and so it is not seen to come from one or the other but is random