ADAPT RAIN SIMULATION IN GAMES

歐愷翔
National Chiao Tung University
Daxuelu 1001, HsinChu, Taiwan
patrickolsen@myw.dk

鄭文皓
National Chiao Tung University
Daxuelu 1001, HsinChu, Taiwan
lobst3rd@gmail.com

丁立澤
National Chiao Tung University
Daxuelu 1001, HsinChu, Taiwan
zingcs86@gmail.com

ABSTRACT

As the size and complexity of today's games, a new technique is designed to modify the execution efficiency of the rains in games. For this project, we parallel the rain simulation program by the use of OpenMP (an API for multi-platform shared-memory parallel programming in C/C++).

1. INTRODUCTION

1.1 Motivation

In real time interactive 3D game, it usually use a cover of foggy texture to make scenes of rain. It makes the experience of game bad. We wish to simulate the rain for each drop it has, such that the rains come to look more natural and gets fully adapted into the game. Rain drops looks simple and ordinary, but to simulate fluid is complex and difficult. Moreover, most of all, the simulation is time-consuming in the real-time interactive 3D game. We want a solution of simulation that is reliable in games.

1.2 Background

Instead of covering texture of rain to a scene, we use particle-based system to simulate fluid. The particle-based system use a method called Smoothed Particles Hydrodynamics (SPH) to simulate fluids with free surfaces. SPH is an interpolation method for particle systems. With SPH, field quantities that are only defined at discrete particle locations can be evaluated anywhere in space.

When it comes to simulating fluids, lots of forces must be considered. For example: gravity, pressure, viscosity, surface tension and collisions to other objects. With SPH, We can compute all forces particles get, and then distribute quantities in a local neighborhood of each particle using radial symmetrical smoothing kernels.

2. Particle-based Fluid Simulation

In particle system, you must consider two laws, law of conservation of mass and law of conservation of momentum. Since the particles in the system will never disappear, we don't need to care about law of conservation of mass. And fluids can be described by Navier-Stokes equation, which conserves momentum.

$$\mathbf{f} = -\nabla p + \rho \mathbf{g} + \mu \nabla^2 \mathbf{v}$$

Velocity field v, density field ρ, pressure field p, external force density field g and μ the viscosity of the fluid.

For the acceleration of particle i we get:

$$\mathbf{a}_i = \frac{d\mathbf{v}_i}{dt} = \frac{\mathbf{f}_i}{\rho_i}$$

For forces:

$$\vec{F}_{gravity} = m\vec{g}$$

$$\vec{F}_{pressure} = -V\nabla p \quad , \quad \mathbf{f}_i^{pressure} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h)$$

$$\vec{F}_{viscosity} = V\mu\nabla^2\vec{u} \quad , \quad \mathbf{f}_i^{viscosity} = \mu \sum_j m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h)$$

3. PROPOSED SOLUTIONS

3.1 Problem statements

The algorithm of simulation contain three parts: initialization, simulation and rendering. Initialization is to initialize the scene including object, viewport…. For rendering, we can call OpenGL build-in function to deal with it. The most complex and time-consuming thing is simulation.

In simulation, particles interact with each other. We need to obtain all forces that every particles get. In order to compute force for a particle, it need to check every affected neighbor particles.

$$\mathbf{f}_i^{pressure} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h)$$

$$\mathbf{f}_i^{viscosity} = \mu \sum_j m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h)$$

To get forces of pressure and viscosity, we computer sum of position difference of particles multiplied by coefficients. The complexity is O(n^2).

3.2 Solutions

We divide the container into small grids which have several particles. So we don't need go through every particles, instead, we just check every particles in neighbor grids. And then we do parallelism on each grid and calculate the forces of the particle with its neighbor particles.

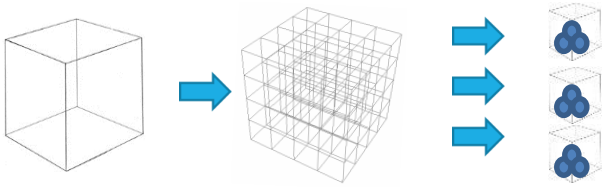
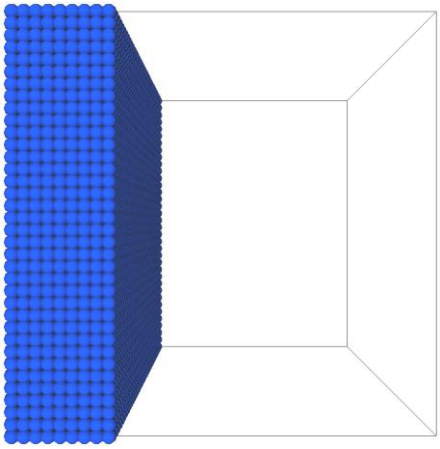


Figure 1 Illustration of dividing a big space, e.g. container, into smaller cubes/grids

4. EXPERIMENTAL METHODOLOGY

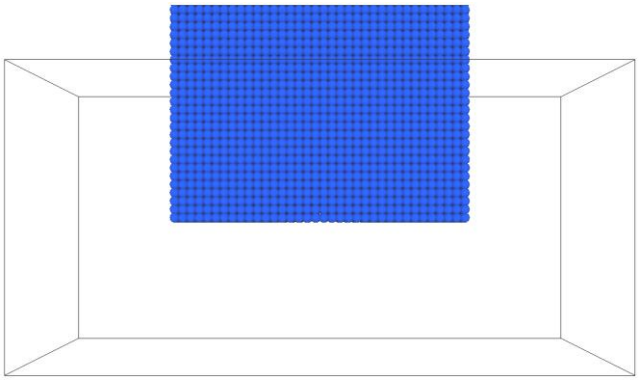
4.1 Tests

To test our code, we take use of three different scenarios. First testing scenario is a scenario of all the particles stacked up onto one side of a container. Later all the particles will fall down and simulate how the particles would act if the container were to be moved from the right to the left. Hereafter the container stops, and only the internal forces acts upon the particles.



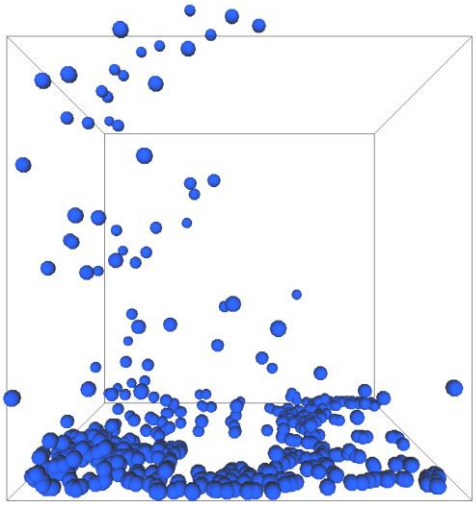
Scenario 1 All particles stacked up onto one side of a container.

The second scenario is a test of a container that is upside down, filled with particles, and hovering over another container. The container having all the particles disappears and all the particles will fall down into the empty container. This scenario is to simulate what would happen, if the container were to be turned upside down or a container filled with water breaks.



Scenario 2 A container filled with raindrops / particles hovering over another container

The third scenario acts like raindrops falling into a container and slowly it builds up the water level.



Scenario 3 Raindrops falling into a container

4.2 Input sets

There are used 11664 particles in scenario 1 and 2 on every test. For scenario 3 we let the program run (rain) till it has summed up about 4000 particles.

4.3 Environment

The environment for the experimental tests are done on: CPU i7 4710HQ @2.50 GHz base, turbo boost @3.5 GHz on Windows 10 64bit.

NOTE *The turbo boost is not deactivated. Which means the results can be misleading for single core and dual core mode. At single core mode the CPU may increase its operating hertz up to 3.5 GHz and dual core up to 3.0 GHz, whereas in quad core mode it will typically run at 2.5 GHz. Thus the results for single core and dual core mode are to be better than if tested at 2.5GHz*

4.4 Record of the OpenMP test

Below here, one will see links to the different tests. The links will send one to youtube, where one can get a demonstration video of the performance on single, dual and quad core mode.

4.4.1 Single core mode

Scenario 1 [LINK] | Scenario 2 [LINK] | Scenario 3 [LINK]

4.4.2 Dual core mode

Scenario 1 [LINK] | Scenario 2 [LINK] | Scenario 3 [LINK]

4.4.3 Quad core mode

Scenario 1 [LINK] | Scenario 2 [LINK] | Scenario 3 [LINK]

4.4.4 Quad core + concurrently mode

Scenario 1 [LINK] | Scenario 2 [LINK] | Scenario 3 [LINK]

5. EXPERIMENTAL RESULTS

5.1 Quantitative data

Below here, one can see the results of the rain simulation, scenario 3.

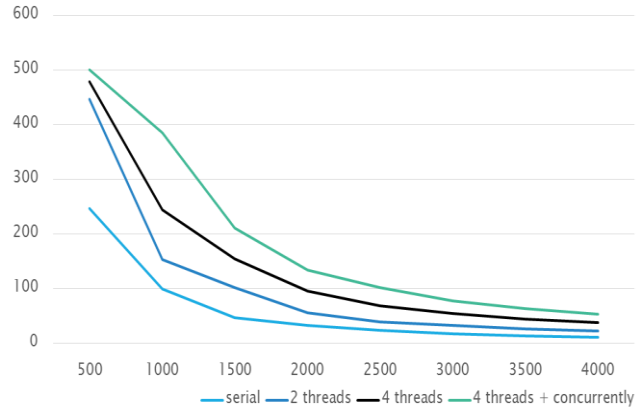


Figure 2 FPS rate in the different modes as the particles increases

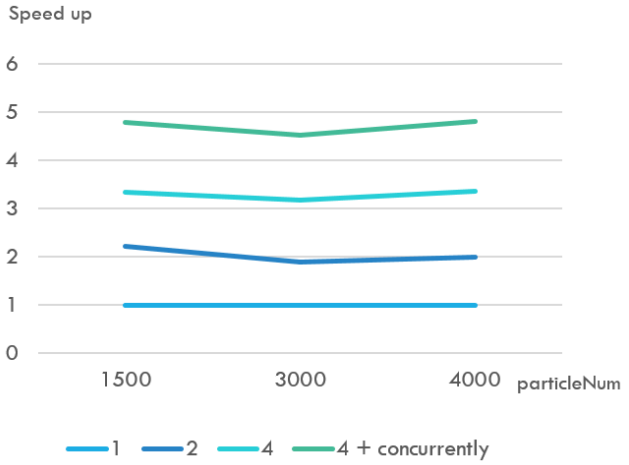


Figure 3 The speed up of the different modes compared to single core mode as particles increases

5.2 Analysis

From the results we can see that with dual core it increases 2 times the core, and with 4 cores it increases by 3.4 times. So currently, the performance boost is about 1.9x per additional cores. If we take in the fact that a single core operates at a higher speed than a dual and quad core does in this environment, one can expect the performance to increase by the number of cores added by approximately 1.98x[1]. Additionally, if the CPU where to have concurrently function such as Intel® Hyper-Threading. Then we can assume an additional 1.5x boost from our results. Thus the code + openMP is quite optimized and efficient for parallelism on CPU.

[1] 1.98x per additional core is an approximation of what the performance boost one could expect. This is a rough guess since we calculate it from the 3.5 GHz to 2.5GHz percentagewise.

6. RELATED WORK

[1] Example of rain filling a container https://www.youtube.com/watch?v=eg0lZm1az1o.

[2] Smoothed Particle Hydrodynamics (SPH) to fluid simulation. https://github.com/saeedmahani/SPH-3D-Fluid-Simulation.

7. CONCLUSIONS

From our experiment with parallelism on CPU, we can conclude that the raining simulation is not quite suited for the CPU. The CPU is too weak to handle all the particles. Moreover, if it also were to handle other game related things, then the performance is just too grave and poorly to be suitable for games. Though we managed to obtain a great performance boost per additional core, it is just not sufficient for games to have a real time rain simulation.

We've done OpenCL version to parallel the part of computing all force and integration. But result is bad because we need to read back positions of particles for each frames, it is a big I/O overhead.

If we were to use the GPU to handle all the forces, collision and rendering etc., then we believe it should be quite possible. However, the CPU is out of the question.

8. REFERENCES

[1] Matthias Müller, David Charypar and Markus Gross. 2003. Particle-Based Fluid Simulation for Interactive Applications.

[2] Micky Kelager. 2006. Lagrangian Fluid Dynamics Using Smoothed Particle