Parallel Programming: Position-based Fluid

Yiyi Cai

February 9, 2023

Abstract

Position Based Fluids (PBF)[MM13] is a fluid simulation method using position based dynamics framework. We see the fluid consists of many small particles. For each step, we update all the particles in the constraints of constant density. The whole process can be summarized as the following psuedo code.

Algorithm 1 Simulation Loop

```
1: for all particles $i$ do
2:     apply forces $\mathbf{v}_i \Leftarrow \mathbf{v}_i + \Delta t \mathbf{f}_{ext}(\mathbf{x}_i)$
3:     predict position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
4: end for
5: for all particles $i$ do
6:     find neighboring particles $N_i(\mathbf{x}_i^*)$
7: end for
8: while $iter < solverIterations$ do
9:     for all particles $i$ do
10:        calculate $\lambda_i$
11:    end for
12:    for all particles $i$ do
13:        calculate $\Delta\mathbf{p}_i$
14:        perform collision detection and response
15:    end for
16:    for all particles $i$ do
17:        update position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i^* + \Delta\mathbf{p}_i$
18:    end for
19: end while
20: for all particles $i$ do
21:     update velocity $\mathbf{v}_i \Leftarrow \frac{1}{\Delta t}\left(\mathbf{x}_i^* - \mathbf{x}_i\right)$
22:     apply vorticity confinement and XSPH viscosity
23:     update position $\mathbf{x}_i \Leftarrow \mathbf{x}_i^*$
24: end for
```

Figure 1: Algorithm of PBF

In this report, the following parts will be included:

- Describe the application or a high-level goal.
- Describe what it does in terms of computation.
- Describe how we are going to apply parallelization/vectorization or other things to make it more high-performance.

1 Describe the application or a high-level goal

Position Based Fluids (PBF) is a fluid simulation method using position based dynamics framework. We see the fluid consists of many small particles. For each step, we update all the particles in the constraints of constant density.

Given an initial state, our goal is to find what the fluid will be like as time goes on (Shown in Figure 2. The left one is the initial state and the right one is a snapshot during the simulation). It can be divided into three main parts: Predict, Revise, Update.

”Predict” represents 1-4 in Figure 1. We guess the position of each particle by its velocity and the gravity force.

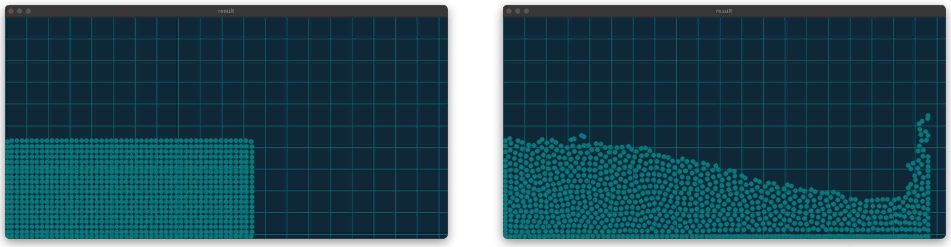
(a) init (b) result

Figure 2: Initial state

”Revise” represents 5-19 in Figure 1. We revise the position by the constraints of constant density so that all the particles will not shrink together.

”Update” represents 20-24 in Figure 1. After the revision, we update the velocity of each particle and one simulation loop ends.

2 Describe what it does in terms of computation

In “Predict” and “Update”, the computation is just naive fmas for each particle. It can be optimized by simd.

$$v_i = v_i + f(x_i)$$
$$x_i = x_i + v_i\Delta t$$
$$(Revise)$$
$$v_i = (x_i - oldx_i)/\Delta t$$

In “Revise”, we should first find all the neighbours of each particles $N_i(x_i)$. Then we loop over the neighbours of each particle to get the revision Δp_i

$$\lambda_i = \sum_{x_j \in N_i(x_i)} \Lambda(x_i, x_j)$$
$$(Synchronize)$$
$$\Delta p_i = \sum_{x_j \in N_i(x_i)} g(x_i, x_j, \lambda_i, \lambda_j)$$
$$(Synchronize)$$
$$x_i = x_i + \Delta p_i$$

The “Revise” takes most of the time of the whole process, so we mainly focus on optimizing this part.

3 Describe how we are going to apply

3.1 OpenMP(CPU)

We can parallel the “Revise” process using the OpenMP. Because the number of neighbours is different for each particle, the computing time is different for each thread in the “for” loop. We use the scheduling(dynamic) to get better performance. The performance acceleration should be the same as the number of cores.

In the real-world test of 30000 particles, the naive approach takes 29904ms. The OpenMP version of 16 threads takes about 2557ms. The acceleration rate is about 12 times.

It is not suitable to use simd for the number of neighbours is different for each particle.

3.2 CUDA(GPU)

We can parallel the "Revise" process using the CUDA. Between the calculation of λ_i and Δp_i, synchronization should be performed, so as between the calculation of Δp_i and the final x_i.

The cuda version takes about 768ms with 64 threads/block * 469 blocks. The acceleration rate is about 39 times.

3.3 Finding neighbours

The problem of moving this algorithm to GPU is that each step the number of neighbours of each particles may change. But the GPU is not good at handling array that can change its length. So in the implementation, I use the prefix sum table and a fixed-length array to store all the particles in grids.

To be specify, because the total number of the particles is fixed, we can make a fixed-length array with the length of the number of the particles. Then for every particles, we calculate the particle number in each grid. By calculating the prefix sum, we can divide the array according to the grid. Finally, we loop over all the particles again to store the particles index in the array according to the grid it belongs to.

In this way, we store the grid information into a foxed-length array which is useful in the GPU implementation. We can get the neighbours of each particles by searching its neighbouring grid, which is much faster than looping over all the particles.

4 Code

Code of this project can be found on the GitHub: Tiny-PBF tauleg000

References

[MM13] Miles Macklin and Matthias Müller. Position based fluids. *ACM Trans. Graph.*, 32(4), jul 2013.