Implementation of the Unified Particle Physics with a focus on Fluids (liquids, gases, and fluid-solid coupling) in 2D

MOHANNA SHAHRAD, McGill University

This project presents a unified dynamics framework and particle solver with a specific focus on modelling fluids (liquids and gases) and fluid-solid coupling based on the work in Macklin et al. [2014]. Multiple types of constraints and particle phases are defined with the motivation to treat their contact and collision in the same unifies way. The proposed solver is primarily based on position-based dynamics [Müller et al. 2007].

Code: https://github.com/mohannashahrad/COMP559_FinalProject
Video: https://youtu.be/d7wpPL1Bw60 or https://tinyurl.com/MohannaShahradCOMP559Project

Additional Key Words and Phrases: fluid, simulation, unified solver, fluid-solid coupling

ACM Reference Format:
Mohanna Shahrad. 2023. Implementation of the Unified Particle Physics with a focus on Fluids (liquids, gases, and fluid-solid coupling) in 2D. 1, 1 (April 2023), 4 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Unified solvers capable of simulating different types of objects have become popular visualization tools mainly because having a single unified dynamics framework results in less code generation, maintenance and optimization cost reductions Macklin et al. [2014]. Furthermore, unified solvers make the fully coupled interaction between different types of objects more feasible. This project explores and implements the unified simulation approach introduced in Macklin et al. [2014] primarily for liquids and gasses. The key that enables us to treat contact and collisions in a unified manner is to use particle representations connected by different categories of constraints as the primary building block. In *METHODS* section, we go through implementation details and in *RESULTS*, we explore the performance and scalability of the proposed solver.

2 RELATED WORK

This project is based on the work of Macklin et al. [2014] that presents a unified dynamics framework for real-time visual effects. Specifically, this work focuses primarily on section 7 of Macklin et al. [2014], which describes the details of simulating fluids (liquids and gasses) in the unified framework. Algorithms and approaches from Macklin and Müller [2013] and Müller et al. [2003] are heavily used to implement the position-based dynamics and constraints in the proposed framework.

Author's address: Mohanna Shahrad, McGill University, mohanna.shahrad@mail.mcgill.ca.

2023. XXXX-XXXX/2023/4-ART $15.00
https://doi.org/0000001.0000001

3 METHODS

3.1 Particle Representation

In this project, all object types are represented by particles. The simplicity of such a representation brings flexibility regarding the wide range of categories of objects to be implemented. The core properties of particles are as follows. We use two global parameters `PARTICLE_RAD` and `PARTICLE_DIAMETER` (with fixed values per scene) to control the displayed size of particles. A property called *phase* is added to the particles to categorize them into multiple groups, each with its own set of properties, functions, and constraints.

```
enum Phase {
    FLUID,
    GAS,
    RIGID,
    NUM_PHASES
};
```

3.2 Constraint Groups

Similar to the way `Phase` organizes particles into multiple groups, we introduce `ConstraintGroup` to differentiate between different constraints that need to be maintained and solved in the simulation step. This work does not focus on rigid and deformable bodies, so their constraints are not considered below. However, having `ConstraintGroup` concept make the framework flexible and scalable with other types of constraints that the end user wants to include.

```
enum ConstraintGroup {
    CONTACT,
    GENERAL,
    NUM_CONSTRAINT_GROUPS
};
```

3.3 Smoothing Kernels

Before going in-depth into the implementation details, mentioning the smoothing kernels used in simulating fluids and gases in this project is noteworthy. Similar to density equations in Müller et al. [2003], the proposed solver uses *Poly6* kernel for fluid density estimation and *spiky Kernel* for gradient calculation that are called from the constraints *project* routines.

$$W_{poly6}(r,h) = \begin{cases} \frac{315}{64\pi h^9}(h^2 - r^2)^3, & 0 \le r \le h \\ 0, & \text{otherwise} \end{cases}$$

$$W_{spiky}(r,h) = \begin{cases} \frac{15}{\pi h^6}(h - r)^3, & 0 \le r \le h \\ 0, & \text{otherwise} \end{cases}$$

Algorithm 1 Projection Alg. in Fluid Density Constraints

1: Clear the old λ_i values
2: Compute updated λ_i values with $\lambda_i = -\frac{C_i(p_1,..,p_n)}{\sum_k |\nabla_{p_k} C_i|^2+\epsilon}$ where $\{p_1, .., p_n\}$ are the neighboring particle positions, ρ_0 is the rest density, $\rho_i = \sum_j \frac{\text{Poly6}(p_i - p_j, h)}{M_j^{-1}}$ is the density estimator of the i'th particle, $C_i(p_1, .., p_n) = \frac{\rho_i}{\rho_0} - 1$ is the density constraint on the i'th particle, and ϵ is a constant relaxation parameter set to 0.01.
3: Compute the total position update Δp_i values where $\Delta p_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j + \text{correctionTerm}) \nabla \text{SpikyGrad}(p_i - p_j, h)$, and correctionTerm $= -0.1(\frac{\text{Poly6}((p_i - p_j, h)}{0.04h^2})^4$ using the suggested constant parameter values in Macklin and Müller [2013].
4: For each particle, update position using the derived position update from the previous step.

3.4 Two-phase liquid Simulation

One of the proposed simulation scenes demonstrates a two-phase liquid with a density ratio of r showing the Rayleigh-Taylor instability. A new fluid constraint is created and added to the system for each fluid with a different density. Later, the *project* subroutine of fluid constraints will be called in the main simulation loop to adjust the particles' velocities and positions. This *project* subroutine is based on Algorithm 1 of Macklin and Müller [2013] with slight simplifications. Algorithm 1 shows the high-level performed steps.

3.5 Gas Simulation

Macklin et al. [2014] proposed a fully Lagrangian approach for simulating gases based on position-based fluids. This project has two separate scenes for simulating gas/smoke particles, one with a closed boundary and the other with an open boundary.

3.5.1 Closed Boundaries. In this case, the closed environment is first filled by FLUID particles with a constant mass of 1. Then the GAS particles get created from a *Gas Injector* at a fixed position, and gravity is reduced for them by GRAVITY_REDUCTION_ALPHA factor. The gas constraint in the proposed solver is a unilateral density constraint to keep the rest density ρ_0 fixed.

The velocity of GAS particles are derived using a weighted average over FLUID particles as shown in *Equation (28)* of Macklin et al. [2014].

$$v_{p_s} = \frac{\sum_j v_j \text{poly6}(p_s - p_j)}{\sum_j \text{poly6}(p_s - p_j)}$$

where v_{p_s} is the velocity of the GAS particle at position p_s and v_{p_j} is the velocity of the FLUID particle at position p_j.

3.5.2 Open Boundaries. The base of GAS and FLUID particles in open boundaries is similar to what we covered in closed environments. However, in this case, the injector emits both GAS and FLUID particles with the new FLUID particles having a higher velocity than those already existing in the environment. For the best visual effects, it should always be the case that GAS particles are surrounded by a thick layer of FLUID particles around their emission point. As Equation (29) of Macklin et al. [2014] suggests, a drag force is used to simulate the effect of fast-moving interior particles with their surroundings.

$$f_{drag_i} = -k(v_i)(1 - \frac{\rho_i}{\rho_0})$$

where k is a parameter to be tuned.

As also mentioned in Macklin et al. [2014], fluid particles can be removed once they have no smoke particles within their kernel radius or after a pre-defined lifetime. This project assumes that the user will interact with the simulator to stop the simulation once the FLUID particles escape the environment. One last note on gases in open boundaries is that there should be a sufficient number of sampling of fluid particles to enable the simulator to accurately estimates velocity for smoke advection.

Algorithm 2 Main Simulation Loop

1: For all particles:
- Update forces and velocity based on any external forces
- For GAS particles, consider reducing gravity by GRAVITY_REDUCTION_ALPHA factor.
- Predict position based on the computed velocity and store it into p->ep
- Apply *Mass Scaling* as explained in Macklin et al. [2014]'s equation (29).

2: For all particles:
- Find the neighboring particles N_i
- Find if there is any solid contact in N_i.
- Check particle's position and if needed add a *Boundary Constraint* to the local copy of the system's constraints.

3: For all constraint groups:
- Solve them and update counter parameter n. (We call the projection function in each constraint SOLVER_ITRS times)

4: For all particles:
- Update velocity and position based on the results of solving constraints.
- Add any internal forces such as f_vortocity or f_drag.
- Update position based on the changes made to velocity.

5: For all injectors in the system:
- Call their simulation loop to create new smoke or FLUID particles.
- Run boundary checks for the new particles one more time.

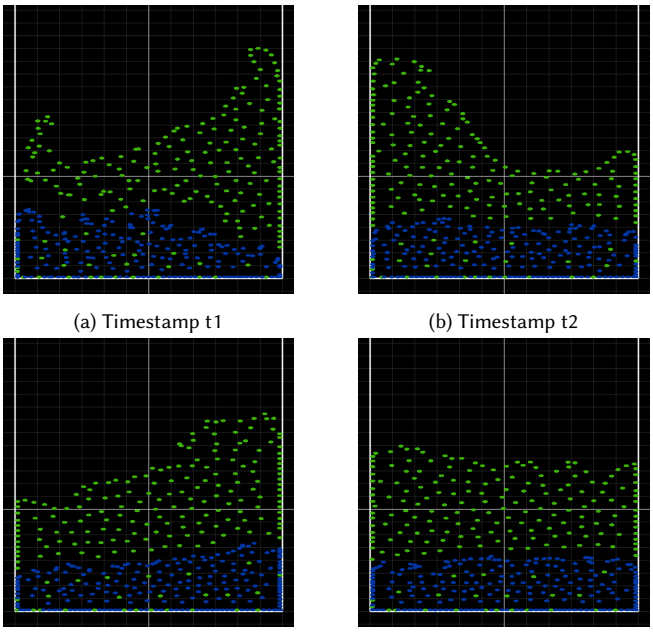
3.6 Simulation Loop Algorithm

Algorithm 2 shows the high level of the steps done in the main simulation loop, which is based on Algorithm 1 of Macklin et al. [2014]. (Note that even though the work in this project is mainly on fluids, gasses, and coupling, some of the rigid body's contact steps in the algorithm is also included to show the end user the possibility of adding them to the framework in future)

4 RESULTS

Figure 1 and 2 show simulator results for the two-phase liquid experiment (showing Rayleigh-Taylor instability) with different density ratios. Figure 3 demonstrates four different scenes in simulating gas in an open-boundary environment. It is interesting to observe the effect of fluid density, injection speed, and the number of surrounding |FLUID particles on the velocity update of the system's particle.

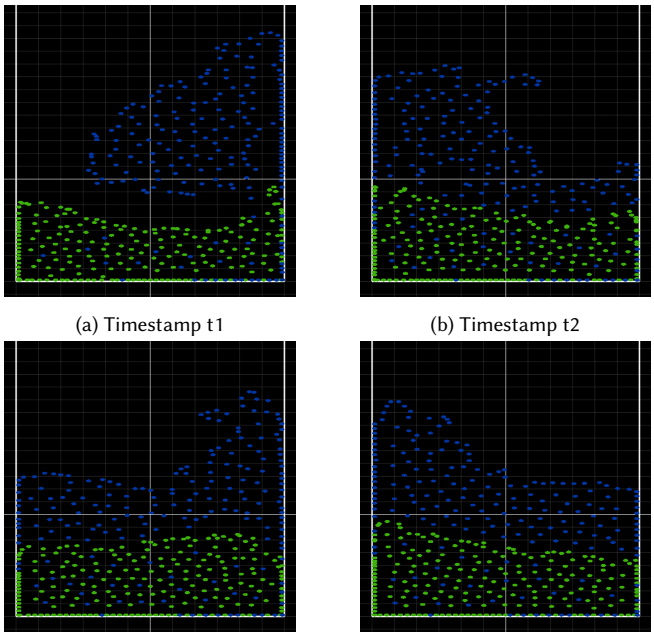
Similarly, figure 4 demonstrates four scenes simulating gas in a closed environment. An interesting observation is that the more



(a) Timestamp t1 (b) Timestamp t2

(c) Timestamp t3 (d) Timestamp t4

Fig. 1. Two-phase liquid example. Density_Green = 1 and Density_Blue = 2



(a) Timestamp t1 (b) Timestamp t2

(c) Timestamp t3 (d) Timestamp t4

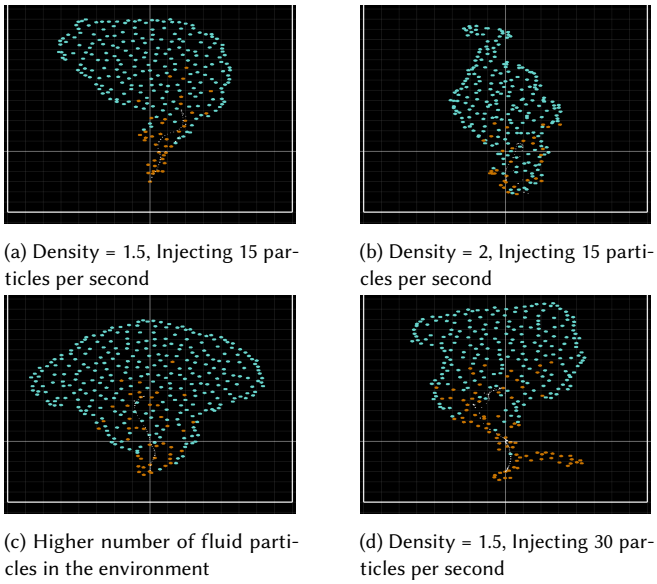
Fig. 2. Two-phase liquid example. Density_Green = 1.6 and Density_Blue = 1

the number of FLUID particles around the injection point of smoke (small white particles), the better the velocity update of smoke particles would be. You can also observe the effect of the FLUID density in the environment on how the velocity field of smoke particles would change.

Aside from the visual results of the simulator, figure 5 represents the average execution time of the main simulation loop 2 with the motivation of assessing performance. As shown in figure 5, with the growing number of particles in the simulation, the average time needed for each main loop iteration gets larger. However, it should be mentioned that the algorithms and approaches introduced in Macklin et al. [2014] should ideally run on GPU to leverage the parallel solver speedup. Given that this project was tested on CPU, it was expected that performance would drop with the growing size of the system.

5 CONCLUSIONS

This project proposes a unified particle solver based on the work done in Macklin et al. [2014] to simulate liquids, gases, and fluid-solid coupling. As discussed before, modelling objects with particles connected by different types of constraints is the key to achieving the unified behaviour of the framework. It was also observed that the performance of the gas simulation (in open and closed) boundaries is tightly connected to the number of FLUID particles around them. There should be a sufficient sampling of FLUID particles to provide accurate velocity estimates for smoke particles' advection. Therefore, this approach might not produce good visual results if used for long-range effects.



(a) Density = 1.5, Injecting 15 particles per second (b) Density = 2, Injecting 15 particles per second

(c) Higher number of fluid particles in the environment (d) Density = 1.5, Injecting 30 particles per second

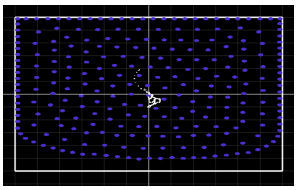
Fig. 3. Simulating gas particles in open boundaries

REFERENCES

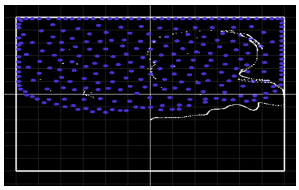
M. Macklin and M. Müller. 2013. Position based fluids. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.

M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim. 2014. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–12.

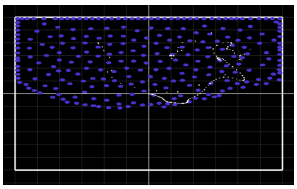
M. Müller, D. Charypar, and M. Gross. 2003. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium*



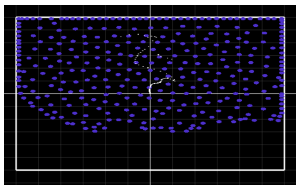
(a) Density = 1, Injecting 15 particles per second



(b) Density = 1.5, Injecting 15 particles per second

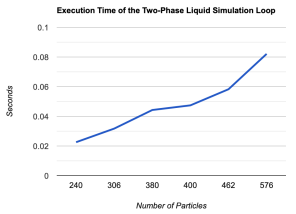


(c) Density = 1.6, with 256 GAS particles around smoke

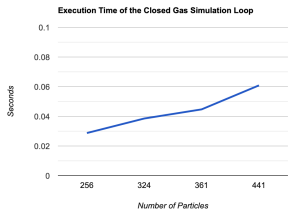


(d) Density = 1.6, with 324 GAS particles around smoke

Fig. 4. Simulating gas particles in open boundaries



(a) Average execution time of Two-Phase fluid main simulation's loop per number of particles



(b) Average execution time of closed-boundary gas simulation's loop per number of particles

Fig. 5. Performance Analysis

on Computer animation. Citeseer, 154–159.

M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation* 18, 2 (2007), 109–118.