IGR202
Supervised by UM Kiwon and DEV PARAKKAT Amal

Versatile Rigid-Fluid Coupling for Incompressible SPH

Pizzini Marie, Institut Polytechnique de Paris France

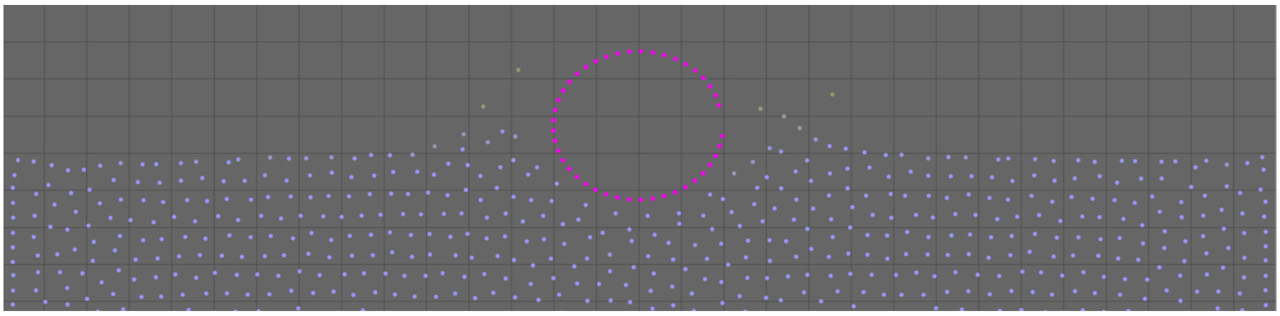


Fig1. Result of the implementation

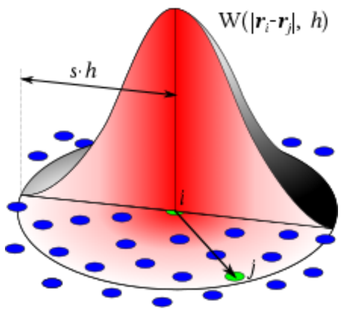
1- Introduction

In computer graphics, fluid simulation is one of the hardest things to render realistically, respecting the laws of physics. The most popular method is using the particle-based Lagrangian approach. This approach is based on the Navier Stokes equation which describes the motion of fluid particles:

$$\frac{D_v}{Dt} = g - \frac{1}{\rho}\nabla p + \nu\nabla^2 v$$

Usually, the number of particles in a simulation is huge and depends on the computation of the neighbors of each particle which can be greedy. To reduce the computational time, a method called Smooth Particle Hydrodynamics (SPH) is based on the use of a smoothing kernel applied to the particles to only consider the neighboring particles that are inside a given radius. It reduces the computational time while preserving continuity. [1]

Fig 2.



Schematic view of SPH convolution

Src : Wikipedia

In a SPH simulation there are basic steps performed by the SPH-based fluid Solver : neighborhood search (usually accelerated by spatial access structure like a grid), density computation, pressure and other forces (like viscous forces) computation and time integration.

One of the challenges of fluid simulation is how to handle interactions between the fluid particles and a rigid boundary that can be moving or static.
There are a few problems that can arise when introducing a rigid boundary. First, it creates a discontinuity and can become a problem when computing the kernel. This density discontinuity can introduce sticking fluid particles to the boundary. [2] The tricky part is therefore to compute the interaction of the boundary on the fluid without introducing discontinuities and without slowing down the computation process (by calculating the pressure for the solid particles for example).
Different methods have been implemented to correctly handle the boundary particles such as using constrained fluid particles, ghost particles, normalizing conditions, forces...

1.1 Literature Review

The first solution to solve the boundary problem is applying a penalty force to the fluid particles. [3] A radial force ((6) and (7)) is applied on the fluid particles depending on their distance to the boundary. The force diminishes when the fluid particle gets further from the boundary. The use of those forces restricts the time step and it can create sticking problems.

$$f_x = \sum_{j=-\infty}^{\infty} \frac{(x_a - x_j)}{r_{aj}} \Phi(r_{aj}, h) \quad \textbf{(6)}$$

$$f_y = \sum_{j=-\infty}^{\infty} \frac{y_a}{r_{aj}} \Phi(r_{aj}, h) \quad \textbf{(7)}$$

Force applied on a particle a at a distance of $y_a$to the boundary. Φ magnitude of the force

Another similar idea is to compute a function that will be added in the density computation of the fluid particles. [4] Like the previous force, this function (8) is based on the distance to the border of the particle. This method helps reduce the sticking problem encountered in the previous method.

$$\rho_i(r_i) = \sum_{j \in fluid} m_j W(r_{ij}) + Z_{wall}^{rho}(|r_{iw}|) \quad \textbf{(8)}$$

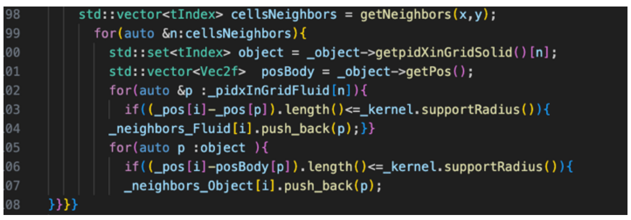
Density of a fluid particle I at a distance r_{iw} to the border. Z_{wall}^{rho} is the penalty function

A different approach can also be used to avoid the sticking problem. It is based on the use of ghost or frozen particles [5]. In this approach, we consider that the boundary particles are like fluid particles. They have the same physical properties except that their velocity (for a moving boundary) is the opposite of the velocity of the fluid particles. This method can introduce penetration problems because the boundary is too thin. Multiple layers of those ghost particles can be used to avoid this problem.

Those approach mainly consider the impact of the boundary on the fluid but not the impact of the fluid on the boundary. Two-way fluid-rigid coupling methods help simulating interactions between the two types of particles. The article Versatile Rigid-Fluid Coupling for incompressible SPH written by N. Akinci, M. Ihmsen, G. Akinci, B. Solenthaler, M.Tescher focuses on how to implement this two-way coupling approach without introducing sticking to the boundary or even penetration of the boundaries.

2- Methodology

This method relies on the fact that boundary particles should contribute to the reconstruction of the field variables (such as the density and the forces computation). To correctly implement this method, it was necessary to create two different solvers as they would be interdependent (the forces calculated by the SphSolver will be passed to the RigidSolver). For each solver, I reused the function I already coded in the practical on Sph Solvers. The neighbor cells are computed and then each particle in those cells within the kernel radius is considered as neighbor of the current particle. It is necessary to build those neighbors vectors at the beginning of the updating loop.



```cpp
    std::vector<tIndex> cellsNeighbors = getNeighbors(x,y);
      for(auto &n:cellsNeighbors){
        std::set<tIndex> object = _object->getpidXinGridSolid()[n];
        std::vector<Vec2f>  posBody = _object->getPos();
        for(auto &p :_pidxInGridFluid[n]){
          if((_pos[i]-_pos[p]).length()<=_kernel.supportRadius()){
        _neighbors_Fluid[i].push_back(p);}}
        for(auto p :object ){
          if((_pos[i]-posBody[p]).length()<=_kernel.supportRadius()){
          _neighbors_Object[i].push_back(p);
}}}}
```

Fig 3. Code Snippet of the computation of neighbors inside the updating function of the SphSolver

After the computation of the neighbor vectors, we compute the density and forces applied on the fluid particles. The particularity of this method is that we consider that the force applied by the boundary on the fluid is equal to the symmetric of the force applied by the fluid on the boundary. The hypothesis is used later in the algorithm to avoid computing the pressure for each boundary particle.

2.1- Density Computation

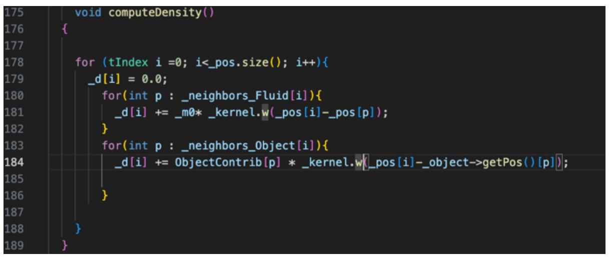
It is important to consider the contribution of boundary particles when computing the density of a fluid particle. We could you use the following approach to comput it:

$$\rho_{f_i} = m_{f_i} \sum_j W_{ij} + m_{b_i} \sum_k W_{ik}$$

However, this equation doesn't consider the volume of the solid particles.
We therefore compute a new quantity called the boundary contribution $\psi_{b_k}(\rho_0) = \rho_0 V_{b_i}$ which gives us the following equation to compute the density:

$$\boxed{\rho_{f_i} = m_{f_i} \sum_j W_{ij} + \sum_k \psi_{b_k}(\rho_0) W_{ik}}$$

When implementing, I started with adapting the way the density was computed in the practical to fit with the article's approach.



```
void computeDensity()
{

  for (tIndex i =0; i<_pos.size(); i++){
    _d[i] = 0.0;
      for(int p : _neighbors_Fluid[i]){
        _d[i] += _m0* _kernel.w(_pos[i]-_pos[p]);
      }
      for(int p : _neighbors_Object[i]){
        _d[i] += ObjectContrib[p] * _kernel.w(_pos[i]-_object->getPos()[p]);

      }

  }
}
```

Fig 4. Code Snippet of the density computation

The Boundary contribution function is coded inside the RigidSolver and is called during the initialization of the rigid Object. This parameter is later passed to the SphSolver to be used in the computation of the density. It only needs to be computed once because the object is rigid and the neighbors inside the object never change after initialization.

When running the simulation without any changes to the way the other forces were computed, we can see that the density of the fluid particles augments but the fluid particles can't "compensate" this higher density with a force applied on the solid.

We observe the particles slowly "exploding". (See *Fig 5.*)

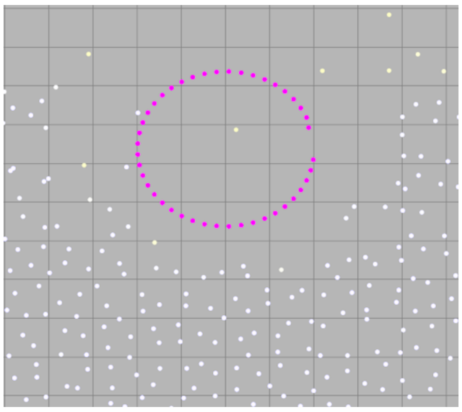


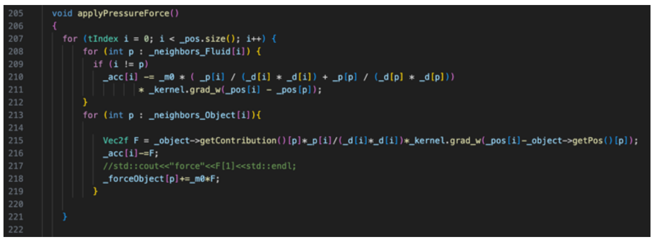
Fig 5. Simulation with only an adapted density and no other forces

2.2- Pressure Forces

We've seen the importance to adapt the forces applied to the fluid particles to take into account the boundary contribution. It is also important because, thanks to force symmetry, we will be able to compute the force applied on the rigid object.

The pressure force applied by a fluid particle on another fluid particle stays the same as in the one I coded in the practical. However, we add another force:

$$\boxed{F^p_{f_i \leftarrow b_j} = -m_{f_i} \psi_{b_j}(\rho_0) \left(\frac{p_{f_i}}{p^2_{f_i}} \right) \nabla W_{ij}}$$



```
void applyPressureForce()
{
  for (tIndex i = 0; i < _pos.size(); i++) {
      for (int p : _neighbors_Fluid[i]) {
        if (i != p)
          _acc[i] -= _m0 * ( _p[i] / (_d[i] * _d[i]) + _p[p] / (_d[p] * _d[p]))
                      * _kernel.grad_w(_pos[i] - _pos[p]);
      }
      for (int p : _neighbors_Object[i]){

          Vec2f F = _object->getContribution()[p]*_p[i]/(_d[i]*_d[i])*_kernel.grad_w(_pos[i]-_object->getPos()[p]);
          _acc[i]-=F;
          //std::cout<<"force"<<F[1]<<std::endl;
          _forceObject[p]+=_m0*F;
        }

  }
```

By the symmetric force $-F^p_{f_i \leftarrow b_j}$ we can get the pressure force applied on the solid by the fluid. This force is added to a vector storing the force applied to a boundary particle j.

Fig 6. Code Snippet of the pressure computation with the force applied by the boundary particles

Because the density increases when a fluid particle gets closer to the boundary, so does their pressure. The magnitude of the pressure force increases when the fluid and the rigid object enter in contact. This creates realistic interaction between the fluid and the rigid object (see *Fig 7.*)

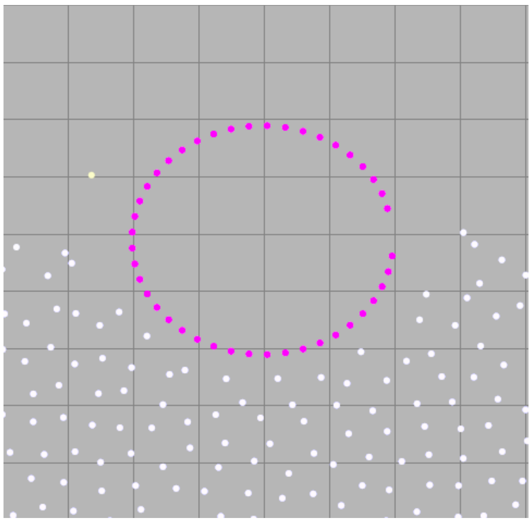


Fig 7. Simulation of the interaction between the object and the fluid when a pressure force is applied by the solid on the fluid and by the fluid on the solid

However, we can see that there is no viscosity interaction between the solid and the fluid because the influence of the solid is not considered in the initial computation of the viscosity force.

2.3- Friction Force

Similarly to the pressure force, the friction (or viscous) force needs to consider the influence of the solid in the computation. The viscous force between fluid particles is the same as the one I coded during practical but we also add the force applied by the solid to the fluid particles.

$$\boxed{F^{p}_{f_i \leftarrow b_j} = -m_{f_i}\psi_{b_j}(\rho_0)\Pi_{ij}\nabla W_{ij}}$$

Similarly, we can compute the force applied by the fluid on the solid.
In the end, the result is quite realistic and the interaction between the fluid and the solid is really a two-way interaction.

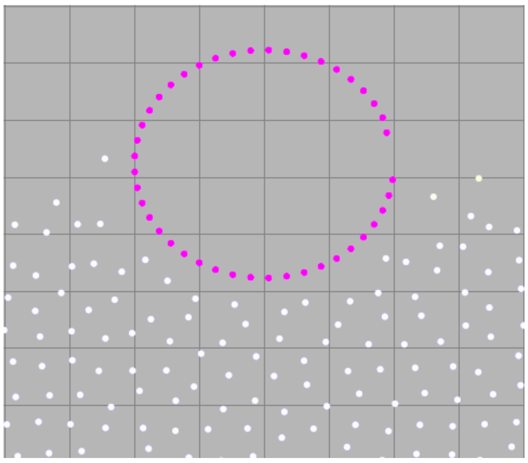


Fig 8. Final result of the simulation with all the forces both between the fluid particles and the solid

3- Result discussion

The computation of the neighborhood is quite expensive during the computation. This leads to a simulation which is really long (approximately 6 minutes for the entire animation). To view the result at a normal speed the video had to be sped up 6 times.
Besides that, the result was quite satisfying as we can see it solved the problems the previous methods had like particle sticking to the boundary or particle penetrating the object boundaries.

4- Personal Insight

The beginning of the project was quite challenging as it was hard for me to understand which forces and variable needed to be computed where and when because the two solvers are really interdependent. It took me some time to understand I could compute the Contribution of the boundary particle directly in the initilalisation of the Rigid Solver as it was not changing during the simulation and was needed in the Sph Solver. However, the algorithm skeleton provided in the article was really helpful and clear and helped me a lot in understanding the order in which each thing needed to be computed..

I also took quite some time to notice an error I made in the practical on RigidSolver when updating the velocity of the rigid object. I was adding the old velocity vector at each step even though I was also adding the old position when updating. The position was added twice each time and the velocity was getting higher and higher at each step. This led to too much force being applied and the fluid particle exploding. (See *Fig.9*) I was convinced it was because of a problem in my implementation of the new pressure force when in fact it came from the RigidSolver :')

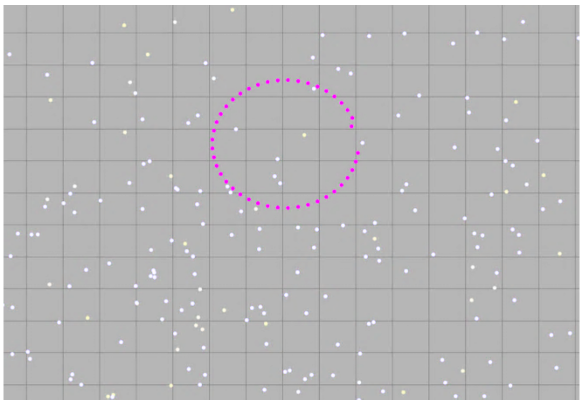


Fig 9. Problems due to the error in the computation of the velocity

I also encountered a problem when rendering the solid particles. The circle wouldn't close. I tried computing the initial position of the boundary particle by turning around a circle with a particle every step. I didn't bother solving it because even though this circle wasn't closed, it didn't influence the interaction with the fluid. The fluid particles were not penetrating the circle boundaries.

The main thing I would do to improve my implementation with more time would be to find a way to reduce a lot the time the animation takes to render. Each updating step takes too long to compute which slows down the rendering. I already tried to reduce this problem by only updating the particles which had changed cells during the step. That helped reduce the number of cells searched at each rendering iteration but it clearly wasn't enough.
This problem could be further improved by reducing the number of loops going through the particles when computing the neighbors.

It would perhaps have been cleverer to group the forces computation and the density calculation in one big loop going over the particles. This would have reduced the number of "for" loops which are present in each force and density computation.

With more time, it could be interesting to explore the simulation of those interactions in 3D. The difference with the 2D is minor and shouldn't be hard implementing.
Other tests would be interesting to see the usefulness of this method. It would have been interesting to run some simulation with different viscosity coefficient to see the impact it has on the fluid-solid interaction.

Simulation with static boundaries could also be interesting. It could be used to define the boundaries of the simulation like a container for exemple. It should not be hard to implement, we can simply constrain the object to a position and have it have a velocity of 0.

Finally, the article suggests that the method works well on articulated objects. It could be interesting to implement this aspect.

Bibliography

[1] M. Ihmsen, J. Orthman, B. Solenthaler, A. Kolb et M. Teschner, «SPH Fluids in Computer Graphics,» Eurographics, 2014.

[2] N. Akinci, M. Ihmsen, G. Akinci, B. Solenthaler et M. Teschner, «Vesratile Rigi-Flui Coupling for Incompressible SPH,» 2012.

[3] J. Monaghan et J. Kajtar, «SPH particle boundary forces for arbitrary boundaries,» Elsevier, School of Mathematics Sciences, Monash University, Clayton 3800, Australia, 2009.

[4] T. Harada, S. Koshizuka et Y. Kawaguchi, «Smoothed Particle Hydrodynamics on GPUs,» The Visual Computer manuscript.

[5] L. Libersky et A. Petschek, «Smooth Particle Hydrodynamics With Strenght of Materials,» Center for explosives technology research and department of physics , 2005.