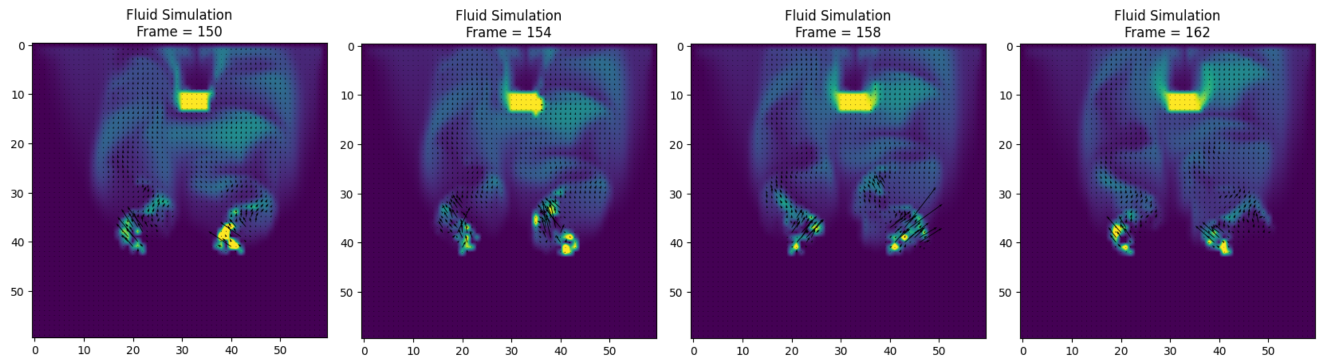
# Fluid Simulation: An Extended Solver

Mariana Ávalos Arce
Universidad Panamericana



## 1 Introduction

Based on the legendary publication of *Real-Time Fluid Dynamics For Games* by Jos Stam[1], the following document presents an extended version of Stam's solver, meaning that the fluid simulation based on Navier-Stokes Equations and a density and velocity solver is used as Stam once suggested in his pioneering paper, but also some extensions are offered to the reader of this document. These extensions include the capability of the solver to simulate the presence of solid objects inside the fluid and different behaviours of the velocity/density emitters based on classic mathematical models.

## 2 Objects In The Fluid

An important extension to the existing solver is the presence of arbitrary objects inside the fluid. These are assumed to be solid and heavy enough to not be carried along the fluid, but the latter should interact with the *walls* of each object.

In order to solve this task, we began by defining how the liquid will interact whenever it collides with the walls of an object. The way any force, in this case the **velocity** of the water stream, should *bounce* against the walls of the object needs to be just like a ball would bounce off a wall when thrown towards it.

We start by identifying the incoming force of velocity at any point in the fluid grid. For that, the solver has a function $set_boundaries(table)$, which is called several times throughout the existing solver in order to handle the bounding box of the fluid. We will add some code in this function, because objects in the fluid means to handle objects as **internal boundaries**. The function receives *table*, which is a grid that contains the velocity bi-dimensional array, where each cell is a 2D vector (array of size 2) as well.

After the existing manipulation of the bounding box bounds, we will add a for loop where we will cover each object in the JSON config file. In each iteration of an object $k$ that has top-left position as *pos* cell coordinates, we loop through the object's height and width, calling them $i$ and $j$ respectively in the loop, but we will start the loops in -1 and finish them in len(obj) + 1, for height, and len(obj[0]) + 1, for width. In this way, the object will be looped with an extra layer for its boundary cells. Next, whenever we are either on the top, bottom, left or right boundary, which we can know with some if conditions, we will gather the top, bottom, left or right neighbour of the current cell $table[pos[0]+i, pos[1]+j]$ by adding or substracting 1 depending on which object wall we are checking. This neighbour cell value is the **incoming velocity** that collides with the wall of the object, which we will call $v$.

Now with $v$ at hand, we will perform a simple **vector reflection**. The input of our function $reflect()$ will be $v$ and $N$, which is the normal vector of the wall we are checking: for the top wall, it would be $\langle 0, 1\rangle$; for the bottom, $\langle 0, -1\rangle$; for the left wall, $\langle -1, 0\rangle$; and for the right wall, $\langle 1, 0\rangle$. Inside the function we will need to compute the incoming force angle $\theta$, which can be calculated with a **dot product**,

$$\theta = \arccos(v \cdot n), \tag{1}$$

where $v$ and $n$ are *normalized* vectors. By having the angle with respect to $N$, we can define the **reflected vector**, which we will call $v'$, using trigonometric relationships shown best through the diagram in Figure 2.

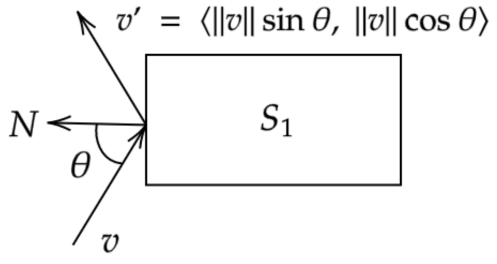


Fig 2. Incoming Force Reflection

Once we have $v'$ vector as the return value of $reflect()$, we can set this vector as our current cell vector in $table[pos[0] + i, pos[1] + j]$, so that we now have the reflected or colliding force direction updated. Then, we simply need to set the density as the average of the values in he current cell $table[pos[0]+i, pos[1]+j]$ in its x and y and we will have the simulation of objects in the fluid. A resulting sequence in the simulation is shown best in Figure 1.

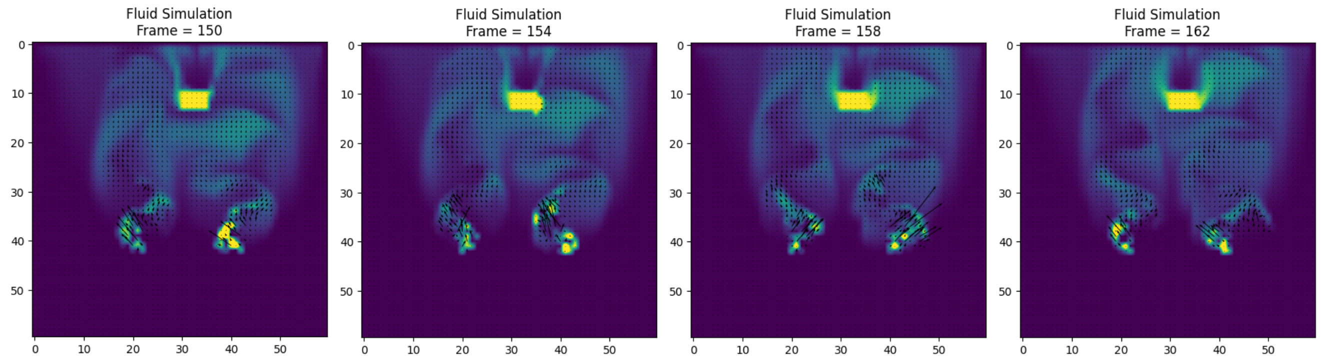


Figure 1: Objects In The Fluid

## References

[1] J. Stam. "Real-Time Fluid Dynamics for Games". In: *Game Developer Conference* (2003). DOI: `https://www.dgp.toronto.edu/public_user/stam/reality/index.html`.