

Computational Fluid Dynamics With a Paper Airplane

Tasada, Daniel Tse, Nathan

November 25, 2024

Abstract

In this paper, we investigate the relationship between an airplane's shape and its performance. Our results show that ...

Contents

1	Introduction	3
2	Execution	4
2.1	Preliminary: Lagrangian Fluid Simulation	4
2.2	Eulerian Fluid Simulation	5
2.2.1	The Code	5
2.2.2	Methodology	5
2.2.3	The Math	5
3	Results	6
3.1	Conclusion	6

1 Introduction

This thesis covers the simulation of the aerodynamics of an airplane, using our own Computation Fluid Dynamics (or CFD) model.

The goal is to calculate the ideal shape of an airplane for its aerodynamic performance. We aim to do this by creating a Machine Learning model, and feed it the results of the CFD, which should result in our final model.

The thesis questions are the following:

- How do the different aspects of fluid dynamics work and how do we implement it in a computer program?
- How do we dynamically generate 3D models?
- How does an airplane's wing shape influence its performance?

2 Execution

2.1 Preliminary: Lagrangian Fluid Simulation

Our first attempt at a fluid simulator was using a technique called Lagrangian fluid simulation. This involves simulating the fluid as a particle collision simulation. It regards fluid dynamics, as particle physics, where every air molecule is a "particle". We start with a container, and a bunch of air molecules, all of which interact with each other to create a fluid. This interaction is effectively the collisions between particles. It turns out that simulating particle dynamics in two dimensions is pretty straightforward. But when you throw in that Z-axis, it gets a lot harder. The main article we used for this is [Rigid Body Collision Resolution](#) (Hakenberg, 2005).

Here is a set of formulas necessary to perform the calculations that describe the behavior of a pair particle before and after their collision:

The following variables are necessary to perform the calculations:

Inertia tensor I ($kg \cdot m^2$) :	$L = \frac{L}{\omega};$
Angular momentum L ($kg \cdot m^2/s$) :	$L = mvr;$
Angular velocity ω (rad/s) :	$\omega = \frac{\Delta\theta}{\Delta t};$

Collision normal ($n \in \mathbb{R}^3$) in world coordinates away from body 1;
Point of contact ($r_i \in \mathbb{R}^3$) in world coordinates with respect to p_i ;
Orientation ($R_i \in SO(3)$) transforming from object to world coordinates;

Where i represents one of two particles in a given collision:

Velocity after collision	$\tilde{v}_i,$
Angular velocity after collision	$\tilde{\omega}_i,$
Constant	$\lambda,$

The following formulas represent the relation between particles:

$$\begin{aligned}\tilde{v}_1 &= v_1 - \frac{\lambda}{m_1}n; \\ \tilde{v}_2 &= v_2 + \frac{\lambda}{m_2}n; \\ \tilde{\omega}_1 &= \omega_1 - \Delta q_1; \\ \tilde{\omega}_2 &= \omega_2 + \Delta q_2; \\ \text{where } q_i &:= I_i^{-1} \cdot R_i^{-1} \cdot (r_i \times n), \\ \text{and } \lambda &= 2 \frac{nv_1 - nv_2 + \omega_1 I_1 q_1 - \omega_2 I_2 q_2}{(\frac{1}{m_1} + \frac{1}{m_2})n^2 + q_1 I_1 q_1 + q_2 I_2}\end{aligned}$$

After lots of trial and error, we were able to successfully implement the math. We did this using Go and raylib, and the code is available at github.com/dtasada/paper at the `lagrangian-go` branch. But we encountered a simple issue where the particles would phase into each other, rendering the particle simulation worthless. This could be because of a too small time step, or a logic error in the collision detection. This should be fixable by adding more simulation steps, which means that each particles solves its collisions more than once per collision, but we weren't able to successfully implement this. In the end, we ended up scrapping the Lagrangian simulation model.

2.2 Eulerian Fluid Simulation

Next we tried a method that we'd had our eyes on for a while. This was had originally been our first choice, but we switched to Lagrangian due to problems I'll explain shortly. It's called Eulerian fluid simulation, and the way this model works is that it sees fluid as a grid of cells that react to each other. Eulerian fluid simulation is an application of cellular automata, which is a computation model used in physics, biology, and many other applications.

Our fluid simulation involves a two or three-dimensional grid of cells, each of which have velocity and density fields. Each frame, the cells interact with each other according to the [Navier-Stokes equations](#).

We're not pretending to understand the math involved in the Navier-Stokes equations, which are difficult to understand without a good background in physics and differential equations. We have humbly borrowed most of the math from Mike Ash's [Fluid Simulation for Dummies](#) (Ash, 2006), which is in turn based on Jos Stam's brilliant work on [Real-Time Fluid Dynamics for Games](#) (Stam, 2012).

2.2.1 The Code

Our code is structured as follows:

```
paper
├── simulation ..... Contains the physics simulation code
│   ├── include ..... Our own engine headers
│   ├── lib ..... External libraries
│   ├── resources ..... Resources like images, fonts, shaders
│   └── src ..... Actual engine source
├── neural ..... Contains ML engine
└── Makefile
```

The code is written in C++, and uses [raylib](#) library for rendering.

The Structure

2.2.2 Methodology

2.2.3 The Math

In Eulerian fluid dynamics we represent fluids with a velocity vector field. This means we assign a velocity vector to every point in space. The Navier-Stokes equations show us how these velocity vectors evolve over time with an infinitely small timestep.

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u + \nu \nabla^2 u + f$$

$$\frac{\partial \rho}{\partial t} = -(u \cdot \nabla)\rho + k \nabla^2 \rho + S$$

The first equation shows the change in velocity in a compact vector notation. Unlike the Lagrangian fluid simulation, in Eulerian fluid simulation the fluid is not represented by individual particles. That's why, instead, fluid density is used, which tells us the amount of particles present in a point in space. The second equation represents the change of this density.

3 Results

Present your experimental or theoretical results in this section. Use tables and figures to illustrate important points.

3.1 Conclusion

Summarize the main findings of the paper. Mention potential future work or research directions.

References

- Ash, M. (2006). Fluid simulation for dummies. www.mikeash.com/pyblog/fluid-simulation-for-dummies.html
- Gradiance. (2024). *Teaching myself c so i can build a particle simulation*. www.youtube.be/NorXFOobehY
- Hakenberg. (2005). Rigid body collision resolution. www.hakenberg.de/diffgeo/collision/rigid_body_collision_resolution.pdf
- Lague, S. (2023). *Coding adventure: Simulating fluids*. www.youtube.be/rSKMYc1CQHE
- Shiffman, D. (2019). *Coding challenge #132 fluid simulation*. www.youtube.be/alhpH6ECFvQ
- Stam, J. (2012). Real-time fluid dynamics for games. www.dgp.toronto.edu/public_user/stam/reality/Research/pdf/GDC03.pdf
- Unknown. (2012). Lattice-boltzmann fluid dynamics. www.physics.weber.edu/schroeder/javacourse/LatticeBoltzmann.pdf
- Wikipedia. (2024). Navier-stokes equations. https://en.wikipedia.org/wiki/Navier%E2%80%93Stokes_equations
- Work, P. (2023). *Writing a physics engine from scratch - collision detection optimization*. www.youtube.be/9IULfQH7E90