Fluid Simulation in C++

Simulation Assignment

21/04/2021

1 Introduction

When tasked with researching a simulation technique in computer graphics, my mind jumped straight to fluid simulations as they are visually appealing and have interesting-but-not-to-complex mathematics driving them.

For this project I have implemented a grid-based fluid simulation using C++ and the NCCA Graphics Library (NGL) to render the simulation in OpenGL.

I briefly review the different technologies used, the implementation details of the program, and finally, critically analyse the approach.

2 Literature Review

2.1 Vector-Fields

In fluid simulations, vector fields are often used to store the data of a fluid simulation providing a vector at each point in a 2D or 3D space that can be used to represent the velocity of the fluid at this particular point, Galbis & Maestre (2012).

2.2 Navier-Stokes & The Grid-Based Method by Jos Stam

Often seen as the pinnacle of fluid simulations, Stam's simple grid-based fluid simulation provides reliable, real-time fluid simulations for incompressible fluids, Stam (2003). This method works by making rough estimates to the Navier-Stokes (see Temam (2001)) equation, using Gauss-Seidel relaxation, and then applies this to a vector field of velocities.

This method is "good enough" for computer graphics where only the visual output needs to be realistic, and due to it running efficiently and being simple to program, it is the perfect choice for this project.

2.3 Moving Least Squares - Material Point Method (MLS-MPM)

Another method I explored was the MPM which offered blazing fast simulations and had many efficiencies over the Stam approach, due to it being a much newer algorithm, Hu et al. (2018).

After some initial reading into this method I realised it was much more complex to implement, with no existing code examples to work from, and due to my project size being small, I felt this task too great to undertake, so opted to stick with the Stam approach.

3 Implementation

When it came to implementing a solution, I first wanted to understand fluid simulations as best as I could so I could focus on the code rather than the mathematics.

To do this, I watched a video by Inspecto (Inspecto (2020)) that explained clearly how fluid simulations worked, outlined each step of the main algorithm simply, and described why each part was needed and what would happen if they were removed. This was a great video that helped me understand all the other papers I read better.

I also watched a video implementing Ash's (Ash (2005)) adaptation of the Stam algorithm by Shiffman (Shiffman (2019)) in Java which helped show some of the steps required to convert the C code into a more modern implementation.

After getting to the stage of understanding fluid simulations better, I started implementing my own simulation. This was based on Stam's approach, and I referred to Ash's paper and his website "Fluid Simulation for Dummies" (Ash (2006)) as inspiration and found the simple explanations most useful when implementing the code in C++.

I wanted to make sure my approach was as object-oriented as possible and that it used C++ features (unlike Stam's & Ash's C implementations). I also made sure to use the NGL package that I am familiar with, meaning I didn't need to write as much code.

To implement the simulation in this way, a full understanding of the Stam code was needed.

Reusing code from NGL demos and my previous project meant it was quicker to get the program up and running, and there were a few efficiencies I could make on the original code, as well as modifications that made the code easier to understand; reusing code is good programming practice and helps keep the code simple and well laid-out – there is no need to recreate the wheel!

Both the implementations I worked from used dye to show the fluid movement, so to make sure my implementation wasn't a carbon-copy of these, I opted to use methods from NGL's grid-based particle visualisations instead.

In this implementation, each particle would start at a set point in the fluid's vector field. To move, the particle would take an average of the four closest velocities and use this to alter its position. The algorithm went like this:

1. A velocity is added to the grid from user input

2. For each time step

(a) The Stam algorithm is used to calculate the velocity for each point in the vector field

(b) For each particle

   i. Particle gets current position

   ii. Uses position to get closest velocities

   iii. Averages velocities and scales the velocity into new velocity

   iv. Updates position based on the new velocity

(c) The particles are drawn to screen using the geometry shader

Using the NGL particle system meant no additional libraries were required to visualise the fluid. This meant the implementation time could be focussed on understanding the fluid mathematics and code.

4 Critical Analysis

This method of fluid simulation has been widely used across computer graphics. This is mainly due to its simplicity and effectiveness, and the fact that it provides a result that looks - good even if not entirely realistic.

The solver can easily be extended to 3D, and by modifying the "set_bnd" function, internal boundaries could be added and removed whilst the program ran. This means the solver can be applied to video games easily.

Whilst the algorithm runs solely on the CPU, the program could be easily implemented on the GPU using the compute shader which would improved the speed of the simulation drastically, further increasing the usability in real-time computer graphics.

The Gauss-Seidel relaxation provides "good enough" results but could easily be replaced by a better partial differential equation solver.

Overall, the Stam algorithm provides a good basis for fluid simulations, and can be extended to improve its efficiency, effectiveness, and use-cases, which is why it is still widely used in computer graphics 20 years later.

References

Ash, M. (2005), 'Simulation and visualization of a 3d fluid', *Master's thesis, Université d'Orléans France* .

Ash, M. (2006), 'Fluid simulation for dummies'.
URL: *https://mikeash.com/pyblog/fluid-simulation-for-dummies.html*

Galbis, A. & Maestre, M. (2012), *Vector analysis versus vector calculus*, Springer Science & Business Media.

Hu, Y., Fang, Y., Ge, Z., Qu, Z., Zhu, Y., Pradhana, A. & Jiang, C. (2018), 'A moving least squares material point method with displacement discontinuity and two-way rigid body coupling', *ACM Transactions on Graphics (TOG)* **37**(4), 1–14.

Inspecto (2020), 'But how do fluid simulations work?'.
URL: *https://www.youtube.com/watch?v=qsYE1wMEMPA*

Shiffman, D. (2019), 'Coding challenge #132: Fluid simulation'.
URL: *https://www.youtube.com/watch?v=alhpH6ECFvQ*

Stam, J. (2003), Real-time fluid dynamics for games, *in* 'Proceedings of the game developer conference', Vol. 18, p. 25.

Temam, R. (2001), *Navier-Stokes equations: theory and numerical analysis*, Vol. 343, American Mathematical Soc.