Fluid Simulation using OpenGL and CUDA

Reem Alghamdi

Solving fluid simulation requires finding a solution to the naiver-stokes equation for incompressible flow:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u}, \quad (1.1)$$

$$\nabla \cdot \vec{u} = 0. \quad (1.2)$$

Where:

- $\boldsymbol{u}$ is the velocity
- ρ is the density
- $\boldsymbol{p}$ is the pressure
- g/F is the force
- **ν** is the viscosity: how resistant the fluid is to flow

In addition, the operator ∇ (nabla) appears 3 times in different contexts:

- $\nabla \boldsymbol{p}$ is the gradient of the pressure.
- $\nabla . \nabla \boldsymbol{u}$ is the laplacian of the velocity.
- $\nabla . \boldsymbol{u}$ is the divergence of the velocity.

All of these quantities are estimated using finite differences in practice.

To make the solution simpler, the viscosity term can be neglected, i.e., viscosity is set to zero. This is referred to as the Euler equation.

In addition, there are two approaches to solve the above equation analytically. In the Eulerian approach, information is traced using a grid. In comparison, the Lagrangian approach traces each particle independently. This project uses the Eulerian approach on a regular 2D grid to solve 2D fluid simulations using OpenGL and CUDA.

Algorithm

The algorithm works in 4 steps:

1. Advection
2. Diffusion
3. Pressure solve
4. Set boundary conditions

Advection

The first step is the advection of the fluid. Advection describes how the fluid move things (itself included). In other words, given some quantity Q on our simulation grid, how will Q change Δt later?

To find the value of the quantity at position x after Δt, the new position of the particle is first calculated by integrating along the velocity field to find the new position after Δt, then, the new quantity is simply the linear interpolation of the quantities near the new position calculated.

In implementation, integration was tested using Euler, Runge-Kutta-2 and Runge-Kutta-4. In the GPU implementation, the timestep was small enough and it was fast, so no noticeable difference was observed. But in the CPU, Euler was good enough albeit not accurate, RK-2 showed the best performance time and quality-wise. RK4 was simply too slow.

For interpolation, bilinear interpolation was used inside the grid, linear interpolation for the boundaries, and nearest neighbor for out of the grid approximation.

Diffusion

This step was the last to be implemented. However, it is correctly done after advection.

This step is only relevant if the viscosity is not zero. It is solved using the same linear solution discussed in the next section.

In addition, after the diffusion step, we can apply forces to the field.

Pressure solve

For pressure solve, the divergence is first calculated. Then, it is used to solve the Poisson equation

$$\nabla^2 p = \nabla \cdot \mathbf{w},$$

In the implementation, it is analytically solved using Jacobi or Gauss-seidel. There is no noticeable difference in their performance.

Once the pressure value is calculated, it is subtracted from the velocity field after advection (and diffusion) to get the divergent-free velocities.

Boundary

Finally, after everything is solved, the boundary is checked. In this project, a no-slip (zero) velocity boundary condition and pure Neumann pressure boundary condition are implemented. To satisfy the conditions, the boundary values should be:

- Pressure: same value as the neighboring cell
- Velocity: the negative of the neighboring cell value

Results

For the results and the code, please check: https://github.com/reem-codes/fluid_simulation

References

- Braley, Colin and Adrian Sandu. Fluid Simulation For Computer Graphics: A Tutorial in Grid Based and Particle Based Methods, 2009.
- Bridson, Robert. Fluid Simulation for Computer Graphics. CRC Press, 2016.
- Harris, Mark. “Chapter 38. Fast Fluid Dynamics Simulation on the GPU.” GPU Gems. Nvidia, https://developer.nvidia.com/gpugems/gpugems/part-vi-beyond-triangles/chapter-38-fast-fluid-dynamics-simulation-gpu
- KAUST’s GPU and Scientific Visualization slides :D