Project 2: Fluid System

The Animation of Natural Phenomena

Adrien Treuille - Carnegie Mellon University

Due 11/04 23:59:59

In this project you will implement a fluid system with the Stable Fluids algorithm plus vorticity confinement. You must also record a video artifact of your system in action. The class will vote on the best artifacts, and the top three winners will receive extra credit. Additionally, you may implement some of the listed extensions (or invent your own!) for extra credit.

Note: Your project must build on the instructional linux machines.

Reading:

- STAM, J . 1999. Stable Fluids. In *Computer Graphics* (SIGGRAPH 1999), ACM, 121–128.
- FEDKIW, R., STAM, J., AND JENSEN, H. 2001. Visual Simulation of Smoke. In *Computer Graphics* (SIGGRAPH 2001), ACM, 15–22.

Skeleton Code:

You have been provided with some skeleton code which you may use to jump-start your project. The code has density fields, velocity fields, and implements basic mouse interaction.

Required Features:

- **Implement Density Advection.** If you're using the skeleton code, then uncomment `Advection` from within `ScalarField::TimeStep`, and implement this function. Use the Semi-Lagrangian advection algorithm described in the Stable Fluids paper. **You should use a MAC-style grid.**
- **Implement Velocity Advection.** In the skeleton code, the function is `Advection` in `VectorField::TimeStep`, as described in Stable Fluids.
- **Implement Pressure.** The function is `Projection` in `VectorField::TimeStep`. The implementation is described in Stable Fluids.
- **Implement Vorticity Confinement.** The function is `VorticityConfinement` in `VectorField::TimeStep`. The implementation is described in Visual Simulation of Smoke.
- **Boundaries.** The edge of your grid should form a rigid boundary. This implies path clipping at the boundary during the advection, and correctly defining the pressure solver at the boundary. Also, be sure not to perform linear interpolation across the boundary!

Optional Features:

You must explicitly indicated which of these you have implemented. Where applicable, your demo must be able to turn each of these features on and off individually so they can be verified.

★	**Multiple densities.** Two different kinds of “smoke” (density) with two different colors.
★	**Temperature.** Causes hot air to rise. As described in this paper.
★	**Higher-order Integrators.** Perform semi-Lagrangian advection with 2nd and 4th order Runge-Kutta.
★ ★	**Monotonic Cubic Interpolation.** Allows for sub voxel accuracy, as described in this paper.
★ ★	**Fixed Objects.** Any edges between voxels could be marked as boundary edges,allowing for arbitrary boundaries.
★ ★ ★	**3D.** Implement the algorithm in 3D. Density rendering should work properly from any angle.
★ ★ ★	**A high resolution hyperbolic solver.** Higher order accuracy. Described in the appendix to this paper.
★ ★ ★	**Particles in Fluids.** Put your particle system in your fluid system, all the fluid field to exert forces on the particles.
★ ★ ★ ★	**Moving Objects.** Allows the user to drag solid objects within the flow. The objects come to rest at exact grid offsets. The movement of the object must exert a force on the fluid.
★ ★ ★ ★	**Water.** Implement a water solver with zero air pressure. Must use a signed distance function for level set, but no need to use the full particle level set method.
★ ★ ★ ★ ★	**Two-way Coupling.** An extension to moving objects. Not only does the object affect the flow, but the flow exerts forces on the object.
★ ★ ★ ★ ★	**Triangle mesh.** Instead of regular grid, simulate on a triangle mesh. (This allows cool boundary surfaces.)
★ ★ ★ ★ ★	**Vortex Particles.** Increase turbulence with vortex particles.
★ ★ ★ ★ ★ ★	**Tetrahedral mesh.** The 3D version of a triangle mesh for simulation. Described here.
★ ★ ★ ★ ★ ★	**Particle Level Set Method.** For higher accuracy, simulate water with the full particle level set method.

More information on Advection:

For the semi-Lagrangian backtrace step, the first thing you need to do is backtrace a set of particles by Δt. You can view this as a differential equation where the state of each particle is **x** its 2D or 3D location. The location **x** starts at the current location where you want to update the value (such as the middle of each grid cell). The motion of **x** is given by the differential equation:

$$\dot{\mathbf{x}} = -\mathbf{u}\left(\mathbf{x}\right)$$

i.e. you interpolate the velocity grid at the position x, negate it to go "backwards" in time, and that forms x's "velocity." You can then use any of the standard integrators we learned in the beginning of class to update x's position. At the very least, you can use an Euler step:

$$\mathbf{x}(\Delta t) = \mathbf{x}(0) - \Delta t\ \mathbf{u}(\mathbf{x}(0))$$

Extra credit (at the 1 star level) would be to use higher order integrators such as 2nd or 4th order Runge-Kutta. This would be a great time to reuse some of your code from project 1. Once you have x's position delta t in the past, you interpolate the grid there and put it in the original grid cell to complete the semi-Lagrangian advection step.

More information on Projection:

To solve the projection step, **you should use a MAC-style grid** and set up a linear system and solve for the pressure. Then subtract the gradient of the pressure step. You may use any code you like to solve the linear system, but I would suggest the linear system solving code from project 1 (included). Essentially the steps are as follows:

Find the pressure by solving the linear system

$$\nabla^2 p = \nabla \cdot \mathbf{v}_{\text{old}}$$

This is similar to how you solved the linear system in project 1, except now the matrix is a discrete finite difference approximation to the laplacian operator. Once you have the pressure p, you can simply take is gradient and subtract that from the velocity field to get the divergence free velocity field:

$$\mathbf{v}_{\text{new}} = \mathbf{v}_{\text{old}} - \nabla p$$

You can check that your algorithm is working correctly by making sure that divergence ($\mathbf{v}_{\text{new}}$) = **0** at all grid points.