CS395T Physical Simulation: Eulerian Gas Simulation

James Dong and Aditya Tewari

May 19, 2020

Contents

1 Introduction

Many scientists over the years have been fascinated by the deceptive nature of fluid dynamics. Smoke however is the most fascinating of these fluids, because of its intense heat, and complex self sustained vortices. The dynamic nature of such a fluid makes it invigorating to study. In the course of studying this natural phenomenon, we learned quite a bit. The over 100 man-hours we put in to deliver this final product have been well spent engrossed by the strangeness of smoke.

2 Goals

1. We wanted to create a smoke simulation using Eulerian fluids, where we were able to model the effects of temperature, and the forces related to it.
2. We wanted to view the characteristic behaviors of smoke, including buoyancy and vorticity.
3. We wanted an efficient simulation that took all optimizations available to it.

3 Background

We based our simulation on "Visual Simulation of Smoke" (Fedkiw '01). We assumed that smoke is inviscid and locally incompressible (compressibility effects only matter when velocities are close to speed of sound). As such, we use the standard Euler equations to simulate fluid motion in a regular grid. The paper modeled the external forces of gravity, buoyancy, and vorticity. It introduced the parameters of pressure p, temperature T, smoke density ρ, and velocity $\mathbf{u} = (u, v)$.

4 Technical Description

4.1 Simulation Overview

0. Calculate external forces
1. Update velocity using reverse advection
2. Apply external forces
3. Calculate pressure using conjugate gradient method
4. Update velocity according to pressure gradient
5. Force velocities to zero at boundaries
6. Update temperature and density using reverse advection
7. Compute heat dissipation
8. Spawn new smoke at emitter

4.1.1 Buoyancy and Gravity

We use the simple model proposed in the paper mentioned.

$$F_{\text{buoy}} = (-\alpha\rho + \beta(T - T_{\text{amb}}))\hat{\mathbf{y}} \tag{Buoyancy}$$

This simply balances the gravitational pull and the difference between the temperature of the air around the smoke, which causes a upward buoyant force due to the temperature differential causing a difference in density.

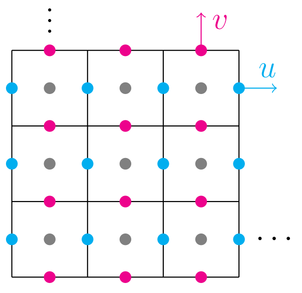
4.1.2 Vorticity Confinement

In coarse discretizations, many small details such as vortices tend to dissipate quickly. We use the method of vorticity confinement to re-inject energy back into the system. The force is designed to increase vorticity of the flow. ω measures the local vorticity at a point (with direction corresponding to the axis of rotation), and $\|\omega\|$ has a local maximum at vortex centers. As such, η points towards vortex centers, so the cross product $\mathbf{N} \times \omega$ points in a direction which increases vorticity. ϵ is a parameter which controls how much vorticity to re-inject.

$$\omega = \nabla \times \mathbf{u} \tag{1}$$
$$\eta = \nabla \|\omega\| \tag{2}$$
$$\mathbf{N} = \frac{\eta}{\|\eta\|} \tag{3}$$
$$F_{\text{conf}} = \epsilon h(\mathbf{N} \times \omega) \tag{Confinement}$$

4.2 Discretization

Figure 1: Diagram of points for discretization.

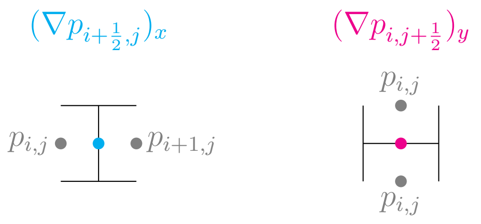


See Figure 1 for a diagram of discretization points. T, ρ, p, F are all computed at the centers, and velocity components are represented at edge centers corresponding to their direction. Below, we use the convention that $u_{i+\frac{1}{2},j}$ lies between cell centers (i, j) and $(i + 1, j)$. In our code, `velocity_x[y][x]` holds $u_{x-\frac{1}{2},y}$, and similarly for `velocity_y`. For simplicity, we set grid size $h = 1$ everywhere.

4.2.1 Gradient

The gradient is used in two places: when computing the gradient of pressure to compute **u** from $\mathbf{u}^*$ (see section on Incompressibility) and when calculating η to compute the direction towards vortex centers for vorticity confinement. The first case is discussed here; the second is done via centered differencing. See Figure 2 for a visual explanation.

Figure 2: Diagram of indices used in gradient calculation.

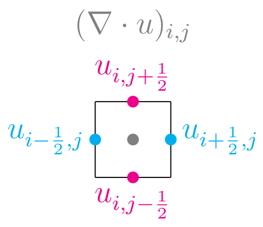

$(\nabla p_{i+\frac{1}{2},j})_x$
$(\nabla p_{i,j+\frac{1}{2}})_y$
$p_{i,j}$
$p_{i,j}$
$p_{i+1,j}$
$p_{i,j}$

$$(\nabla p_{i+\frac{1}{2},j})_x = p_{i+1,j} - p_{i,j} \tag{4}$$
$$(\nabla p_{i,j+\frac{1}{2}})_y = p_{i,j+1} - p_{i,j} \tag{5}$$

4.2.2 Divergence

The divergence is used to generate the right-hand side for the Poisson equation (discussed later), used to solve for pressure. See Figure 3 for a visual explanation of the below formula.

Figure 3: Diagram of indices used in divergence calculation.



$$\nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \tag{6}$$
$$(\nabla \cdot \mathbf{u})_{i,j} = u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j} + v_{i,j+\frac{1}{2}} - v_{i,j-\frac{1}{2}} \tag{7}$$

4.2.3 Curl

The curl is used for calculating vorticity at cell centers. We are only interested in the $\hat{\mathbf{z}}$-component of the curl:

$$(\nabla \times \mathbf{u})_z = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{8}$$

First computing centered estimates of velocity:

$$u_{i,j} \approx \frac{1}{2}(u_{i-\frac{1}{2},j} + u_{i+\frac{1}{2},j}) \tag{9}$$

$$v_{i,j} \approx \frac{1}{2}(v_{i,j-\frac{1}{2}} + u_{i,j+\frac{1}{2}}) \tag{10}$$

Then taking the centered difference to estimate $d\mathbf{u}$:

$$\nabla \times \mathbf{u} \approx \frac{1}{2}\hat{\mathbf{z}}(v_{i+1,j} - v_{i-1,j} - u_{i,j+1} + u_{i,j-1}) \tag{11}$$

4.2.4 Laplacian

The Laplacian at a point is equal to the divergence of the gradient at that point; we can estimate that by measuring the discrete divergence of the discrete gradients, which amounts to taking the sum of the difference between that point and all adjacent points (neighbors).

Each point has its own row and column in the Laplacian matrix. We consider the number of neighbors, n. Each cell gets $-n$ on the diagonal, and each neighbor gets 1 at the corresponding column. Obstacles and boundary cells must be handled slightly differently, but it suffices to not count them as neighbors and for cells contained within obstacles, to set their diagonal entry to 1.

This turns out to be negative definite, so we can use the conjugate gradient method with incomplete Cholesky preconditioner to solve for the pressure, as outlined in the next step.

4.2.5 Incompressibility

We first take an unconstrained step to obtain $\mathbf{u}^*$; using this we compute the necessary pressure to obtain a divergence-free $\mathbf{u}$.

$$\nabla \cdot \mathbf{u} = 0 \tag{12}$$

$$\mathbf{u} = \mathbf{u}^* - \Delta t \nabla p \tag{13}$$

Taking the divergence of equation 13 and substituting equation 12,

$$\nabla^2 p = \frac{1}{\Delta t}\nabla \cdot \mathbf{u}^* \tag{Poisson equation}$$

The discretization of this gives a system of linear equations which can be solved with the conjugate gradient method: specifically, if L is the Laplacian matrix and $\mathbf{p}$ is a vector of pressures, then (negated since L is negative definite)

$$-L\mathbf{p} = -\frac{1}{\Delta t}\nabla \cdot \mathbf{u}^*. \tag{14}$$

4.3 Expansions

4.3.1 Obstacles

Obstacles are modeled using the Neumann boundary condition ($\partial p/\partial \hat{\mathbf{n}} = 0$). We can simply delete the corresponding terms in the Laplacian (diagonals of interior cells are set to 1) and set the velocities to zero each frame. The interior of obstacles is set to the ambient pressure, temperature, and density.

4.3.2 Heat Dissipation

We model heat dissipation using Newton's Law of Cooling, which states that given h as some constant, c as the specific heat capacity, m as the mass, and A as surface area,

$$\frac{dT}{dt} = -\frac{hA}{cm}(T - T_{\text{amb}}) \tag{15}$$

We however in our simulation let $T_{\text{amb}} = 0$.

We notice now if we assume that the smoke is sparse (i.e. $A, m \propto \rho$), we may simply write:

$$\Delta T = -\kappa T \Delta t \tag{16}$$

where $\kappa = hA_p/(cm_p)$ is some constant (A_p and m_p are constants of proportionality such that $A_{\text{total}} = \rho A_p$.)

4.4 Visualization

The output of the simulator is visualized using a simple OpenGL visualizer and GUI for parameters. The grid values are packed into a texture, which is visualized by drawing opaque smoke according to density and adding a fire effect to show temperature. The code for the visualization is located in `main.cpp`, and synchronization between the main thread and simulation thread is done mainly using a triple buffer implemented in `sync.cpp`.

5 Trials and Tribulations

We spent quite a bit of time debugging various parts of our code, agonizing over minute details, in order to get a better simulation. Basically if we had an error it was an indexing error. Surprisingly we didn't have any race conditions and our multithreading and lock-free synchronization code worked correctly on the first try.

6 Limitations

- We currently do not handle the input of more complicated cases of obstacles, such as moving obstacles and those with a different temperature. Our code would however with slight modification be able to handle this case.
- The simulation is only two-dimensional, though it could be made to be three-dimensional without major modifications to the overall structure. The reason this design decision was made is that it provided a more robust, testing environment (much more easily visualizable), and made it easier to spot errors.
- We do not handle conduction between adjacent smoke particles (though this is partially handled by advection), as we thought that it would be mostly negligible (especially compared to dissipation to ambient heat).
- We do not change the background temperature, instead treating it as a cold reservoir.
- As always with any simulation we are unable to model the equations exactly.

7 Bibliography

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. 2001. Visual simulation of smoke. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH '01). Association for Computing Machinery, New York, NY, USA, 15–22. DOI:`https://doi.org/10.1145/383259.383260`