Visual Simulation of Smoke

Zach Corse[1]

August 5, 2018

[1]University of Pennsylvania, CGGT

Abstract: In this project I endeavor to recreate Fedkiw, Stam, and Jensen's paper *Visual Simulation of Smoke*. This includes handling the traditional algorithms requisite in computer graphics fluid simulations, namely, advection, force updates, and projection in an attempt to model the inviscid, incompressible Euler equations. My particular implementation includes an incomplete Cholesky-preconditioned conjugate gradient solver, vorticity confinement, and a tri-linear interpolation scheme for sampling quantities on a MAC grid. I chose not to use the base code provided for this assignment, and so implemented everything described in this summary from scratch.

1. Theoretical Background

Modeling smoke requires an algorithmic approach to solving the incompressible (divergence-free) and inviscid (zero diffusion) Euler equations given by

$$\nabla \cdot \boldsymbol{u} = 0 \tag{1}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} = -(\boldsymbol{u} \cdot \nabla)\,\boldsymbol{u} - \nabla p + \boldsymbol{f} \tag{2}$$

where $\boldsymbol{u} = (u, v, w)$ describes the three Cartesian velocities stored in the centers of the faces of each MAC grid cell, p is the scalar pressure stored in the center of a MAC grid cell, and $\boldsymbol{f}$ is the sum of all body forces to be applied to velocities stored on the grid.

This PDE is solved using a split integration scheme. We begin with the advection term,

$$\frac{\boldsymbol{u}^* - \boldsymbol{u}}{\Delta t} = -(\boldsymbol{u} \cdot \nabla)\,\boldsymbol{u} \tag{3}$$

and assume that temperature (T) and smoke density (ρ) are advected with the velocity field, such that they update in time according to the same governing equation:

$$\frac{\partial T}{\partial t} = -\left(\boldsymbol{u} \cdot \nabla\right) T \tag{4}$$

$$\frac{\partial \rho}{\partial t} = -\left(\boldsymbol{u} \cdot \nabla\right) \rho \tag{5}$$

Velocities defined on the MAC grid are then updated according to two forces: buoyancy, which we naturally associate with smoke, and vorticity confinement, a technique introduced by Fedkiw et al to account for energy losses that are a consequence of repeated interpolations in Stam's semi-Lagrangian scheme. The buoyancy force for each cell is calculated according to

$$-\alpha\rho\boldsymbol{z} + \beta\left(T - T_{amb}\right)\boldsymbol{z} \tag{6}$$

where $\boldsymbol{z} = (0, 0, 1)$ denotes world up. The first term accounts for the sinking of a dense gas in a gravitational field and the second term describes how hot gases (relative to ambient temperature) rise.

Vorticity confinement is also added to each cell, and is calculated according to

$$\boldsymbol{f}_{conf} = \epsilon h\left(\boldsymbol{N} \times \boldsymbol{\omega}\right) \tag{7}$$

where ϵ is a tuning constant, h is the length of a grid cell, and vorticity, $\boldsymbol{\omega}$ is calculated as the curl of the local velocity field

$$\boldsymbol{\omega} = \nabla \times \boldsymbol{u} \tag{8}$$

and $\boldsymbol{N}$ is calculated as the normalized gradient of the vorticity strength ($\boldsymbol{\eta} = \nabla\left|\boldsymbol{\omega}\right|$):

$$\boldsymbol{N} = \frac{\boldsymbol{\eta}}{\left|\boldsymbol{\eta}\right|} \tag{9}$$

Velocity fields are updated using the sum of these forces according to

$$\frac{\boldsymbol{u}^{**} - \boldsymbol{u}^{*}}{\Delta t} = \boldsymbol{f} \tag{10}$$

To maintain the incompressibility condition, we must ensure that the velocity field remains divergence-free after each time step Δt executed in the simulation. By combining the following

$$\frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^{**}}{\Delta t} = -\nabla p \tag{11}$$

$$\nabla \cdot \boldsymbol{u}^{n+1} = 0 \tag{12}$$

we derive the Poisson equation

$$\nabla^2 p = \frac{1}{\Delta t} \nabla \cdot \boldsymbol{u}^{**} \tag{13}$$

and by solving Poisson's equation for pressure, we can update the velocity field to ensure that it remains divergence-free:

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^{*} - \Delta t \nabla p \tag{14}$$

2. Semi-Lagrangian Advection

I follow Stam's semi-Lagrangian advection scheme to ensure the stability of my simulation. The quantities to be advected are updated by transforming from an Eulerian perspective to a Lagrangian, particle-based view. Velocities stored on the MAC grid are transferred to imaginary particles that arrived there from the previous simulation time step carrying with them velocities, temperature, and density values that are to be assigned to these grid points.

I handle particle back-tracing using RK2 integration:

$$\boldsymbol{x}_{mid} = \boldsymbol{x}_{grid} - \frac{\Delta t}{2}\boldsymbol{u}_{grid} \tag{15}$$

$$\boldsymbol{x}^{n-1} = \boldsymbol{x}_{grid} - \Delta t \boldsymbol{u}_{avg} \tag{16}$$

where $\boldsymbol{u}_{avg}$ is the average of the grid velocity and the velocity sampled at x_{mid}. Sampling is handled via tri-linear interpolation across the voxel whose vertices are defined by the particle's eight nearest neighbor grid coordinates corresponding to the advected quantity in question. The value sampled at x^{n-1} is then assigned to the grid point that we back-traced from.

It should also be noted here that particles advected to a SOLID grid cell are clipped, i.e., the advected quantity they carry is set to zero to avoid advecting through walls. The time step used in the simulation (10^{-3}) was sufficiently small to prevent this from happening in general, and also prevented particles from sticking to walls (which have zero-valued normal velocities).

3. Projection

Poisson's equation is solved on the MAC grid using conjugate gradient descent preconditioned with a modified incomplete Cholesky preconditioner. This was implemented as prescribed by Robert Bridson in *Fluid Simulation for Computer Graphics*. The Laplacian (pressure coefficient) matrix was prepared using the seven-point stencil method, which accounts for SOLID cell types (hard boundaries). In its current configuration, my simulation allows the user to specify the location of SOLID cells, but does not account for moving solid objects.

I include several tests in my code that check to ensure various conditions hold true in my simulation. For example, tests confirm that: the Laplacian is symmetric and its rows sum to zero, the sum of the divergence values calculated on the right-hand side of the Poisson equation sum to zero, and that after velocity values are updated by subtracting pressure gradients, the velocity field on the grid is divergence-free.

4. MAC grid Implementation

I used the standard Eulerian MAC grid for smoke simulations, but would like to note a particular implementation detail that eased calculations on the grid. I stored u, v, w, T, and ρ values in arrays using a double buffer system in which values are read from one buffer and written to the next. Pointers to

these buffers are then swapped after each calculation, such that the next step in the split integration scheme reads from the buffer the previous step wrote to. This obviated the need to allocate a large amount of memory on the stack for each calculation executed over a simulation time step.

5. Rendering and Results

I chose to render particles whose positions update according to the velocity values sampled at their positions in the Eulerian grid. As with advection, velocity values are integrated using a second-order Runge-Kutta scheme. Particle position and velocity data are written to Partio .BGEO files, which are then read and visualized in Houdini. My renders are all colored according to particle velocity. Primary colors represent velocity minima, and these linearly interpolate to white, which represents particle velocity maxima.

I generated five separate renders to demonstrate the versatility of my simulation code:

1. *Single Source at the Center of the Bottom Face of a Box*: This render demonstrates what the canonical, centrally placed, single smoke source in a box looks like. It serves as the experimental "control" relative to the other renders.

2. *Directly Colliding Sources*: This render demonstrates what happens when two sources are aimed at one another. In this case, one is oriented upwards and the other is oriented downwards. Because buoyancy tends to lift smoke in the upward direction, we see the bottom source overtaking the top and folding into it from below.

3. *Twin Indirectly Interacting Sources*: In contrast to the previous render, these sources are aimed along tangents to an imaginary spiral wound inside the simulation cube. We see that these sources interact via a pressure fault line that develops between them that encourages a mutually-induced wrapping, spiral-like behavior.

4. *Three Sources Occupying the Same Sim Cube*: In this case, three sources inject into the same simulation cube, but start injecting at different times. This means that smoke A should fill more space than B, which should fill more space than C. Additionally, A should prevent B from rising as high and as fast as A, while B should prevent C from rising as high and as fast as B.

5. *A Simulation Cube in Which Some Cells are Labeled as SOLID*: This render demonstrates that my core solver handles solid boundary conditions (using the methods prescribed in Bridson's text).

6. Conclusion and Future Work

My code accurately simulates the behavior of smoke in a confined volume using the methods described in Fedkiw et al's paper and Bridson's canonical text. I'd like to add to what I've written by upgrading from a tri-linear interpolation scheme to a tri-cubic one, and I'd also like to add a density field visualization feature to compare against the efficacy of the particle source method I've outlined above.

References

1. Bridson, R. Fluid Simulation for Computer Graphics. CRC Press, 2015

2. Fedkiw, R., Stam, J. & Jensen, H. Visual Simulation of Smoke. ACM. 2001, 101, 15-22. DOI: 10.1145/383259.383260