Visual Simulation of Smoke

Nicolas Morales

November 27, 2012

1. Goals
    - We want to simulate smoke for rendering applications
        - This means that we can make certain approximations in our model that speed things up
        - We also want to use a coarser grid than methods specifically for computational fluid dynamics
    - Realistic smoke rendering also has to be physically correct and not just fast.
        - This means our method cannot suffer from things like numerical dissipation because of the approximations it makes
2. Main Reference
    - Most of this talk will come from the paper *Visual Simulation of Smoke* by Fedkiw, Stam, and Jensen.
    - This paper is both a numerical simulation and rendering paper. We will only be covering the simulation aspect, but if you want to know more about the rendering aspect the paper covers both a hardware based renderer and a photon mapping renderer.
3. The Equations of Fluid Flow I
    - Generally, in fluid dynamics, we use the Navier-Stokes system of equations:

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

$$\frac{\delta \vec{u}}{\delta t} = -(\vec{u} \cdot \nabla)\vec{u} - \frac{1}{p}\nabla p + v\nabla^2 \vec{u} + \vec{f} \tag{2}$$

    - $\vec{u}$ is the velocity field
    - p is the pressure
    - v is the viscocity of the fluid
    - $\vec{f}$ is the sum of all external forces on the fluid

- The first equation represents the conservation of mass. Mass is neither created nor destroyed in the system.
- The second equation represents the conservation of momentum of the fluid.

4. The Equations of Fluid Flow II

   - In their paper, Fedkiw et al. use the following versions of the Navier-Stokes equations:

$$\nabla \cdot \vec{u} = \vec{0} \tag{3}$$

$$\frac{\delta \vec{u}}{\delta t} = -(\vec{u} \cdot \nabla)\vec{u} - \nabla p + \vec{f} \tag{4}$$

   - This is just the normal Navier-Stokes equations with the viscosity vector removed. The $\frac{1}{p}$ factor is just ignored.

5. The Equations of Fluid Flow III

   - The term $-(\vec{u} \cdot \nabla)\vec{u}$ is also known as the advection term. Think of it as describing how the velocity of the fluid is affected by currents in the fluid.
   - We can also use this term to describe how temperature T and density ρ are moved along the system:

$$\frac{\delta T}{\delta t} = -(\vec{u} \cdot \nabla)T \tag{5}$$

$$\frac{\delta \rho}{\delta t} = -(\vec{u} \cdot \nabla)\rho \tag{6}$$

6. The Equations of Fluid Flow IV

   - Density and temperature affect the velocity of the fluid. Heated gasses tend to rise, while very dense gases fall downwards. Fedkiw et al. use the calculated temperature and density to compute the buoyancy force:

$$\vec{f_{buoy}} = -\alpha\rho\vec{z} + \beta(T - T_{amb})\vec{z} \tag{7}$$

   - In this equation, T is the temperature, T_{amb} is the ambient temperature of the system, and $\vec{z}$ points in the upwards direction. α and β are magic constants that should be appropriate for the simulation.

7. Numerical Dissipation

   - We now have three equations with the advection term. However, solving for these terms numerically can cause dissipation. This smooths out certain features of fluids, such as vorticity, that we want.

8. Vorticity Confinement I

- Because the numerical dissipation smooths out vortices in our system, we can add it back in to get a curling smoke effect.
- The method to do this is called Vorticity Confinement
- One way could be to perturb the field to add in some randomization. This can be a physically based perturbation. However, because small scale features won't necessarily be added in to the correct place, this can make the simulation curl too much, as if it were alive.

9. Vorticity Confinement II

   - Fedkiw et al.'s method looks for locations within the flow field where these features should be generated and adds them in to generate realistic turbulence.

10. Understanding Vorticity

    - Vorticity is the curl of the velocity of the field. Imagine having a small sphere suspended in a fixed point in the fluid. As the fluid moves, the sphere can start spinning.

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

11. Vorticity Confinement III

    - In this method, we compute normalized vorticity location vectors. These vectors point from lower vorticity concentrations to higher concentrations.

$$\eta = \nabla|\omega|$$

$$N = \frac{\eta}{|\eta|} \tag{9}$$

12. Vorticity Confinement IV

    - Using our normalized velocity location vectors, we can then determine a force that spins around areas of high vorticity.

$$\vec{f}_{conf} = \epsilon h(N \times \omega) \tag{10}$$

    - In this equation, N is the normalized velocity location vector computed in the previous slide and $\epsilon > 0$ is used to control the amount of detail added back into the flow field. h is the distance step between grid cells and makes sure the equation stays accurate if the mesh is refined.
    - This technique was introduced by Steinhoff for modeling the flow fields around helicopter rotors.

13. Implementation: Discretization

- The system is discretized into a grid of voxels for solving our equations numerically.
- Each voxel center has an associated temperature, density, and external forces defined for it.
- Velocity is defined for each face of the voxel.

14. Implementation: Boundaries

    - Boundaries are handled by setting a flag for occupied voxels.
    - All faces of occupied voxels have their velocities set to the velocity of the occupying object.
    - The temperature of the occupied voxels are also set to the object's temperature
    - Occupied cells also have a density of zero, since there is no smoke in that region. However, at the boundary of the object, the density is set to the density of the the closest unoccupied voxel so as to avoid sharp transitions.

15. Implementation: Overview

    - The simulation uses two grids: one to read from and one to write to each simulation step. Each simulation step, the grids are swapped.
    - The user can set an initial grid or it can be empty.
    - At each time step, the velocity is first updated. This is three steps:
      - Adding in external forces
      - Solve for the advection term
      - Force the velocity field to conserve mass
    - Then, the temperature and density are updated

16. Implementation: External Forces

    - External forces that influence velocity include ones supplied by the user, buoyancy forces computed using temperature and density, and the vorticity confinement force.
    - The influence on velocity is calculated using the Euler method of just adding in the total force multiplied by the time step.

17. Implementation: Advection Term

    - The method that this paper proposes uses a semi-Lagrangian method for solving for the advection term.
    - What we're trying to figure out is how the velocity field changes from advection

18. Semi-Lagrangian Methods I

- By now you have likely heard of the Eulerian and Lagrangian frames of reference
- The Eulerian frame of reference describes how a fluid behaves by sampling the velocity, density, etc at specific points, usually on a grid. These points are fixed and do not move.
- The Lagrangian frame of reference is from an individual particle and moves with it as it moves through the vector field

19. Semi-Lagrangian Methods II

    - The semi-Lagrangian method is commonly used in atmospheric simulation.
    - It involves combining the Eulerian and Lagrangian frames of references.
    - A grid is maintained, just like in the Eulerian frame of reference. However, when we advance a time step, we examine the path of a particle that ends up in a specific grid point.

20. Semi-Lagrangian Methods III

Figure 1: In the semi-Lagrangian method, a particle is traced back to its old position

21. Semi-Lagrangian Methods IV

    - To put that into a mathematical form, we can image a function $\phi(\vec{x}, t)$ that takes a point on our grid and a time.
    - Let's say we want to know the value of $\phi(\vec{x}, t)$ at a particular node $\vec{x}$ and time t_n. We can then see where the particle that is at node $\vec{x}$ at time t_n was at some other point $\vec{y}$ at time t_{n-1}.
    - We can estimate the old position of the particle by subtracting the velocity times the time step:

$$\vec{y} = \vec{x} - \Delta t F(x, t_{n-1}) \tag{11}$$

- Then simply evaluating our function $\phi(y, t_{n-1})$ will give us the approximate value of $\phi(x, t_n)$, since that's where the particle is now.

22. Semi-Lagrangian Methods V

- If our point $\vec{y}$ does not lie on a grid node, we can just interpolate values.
- While linear interpolation is only first order in its accuracy, it's numerically stable.
- Fedkiw et al. use a specific cubic interpolation scheme for higher orders of accuracy that still maintains stability.
- Note that if a path runs through an occupied voxel, it can be clipped and then interpolated if needed.

23. Implementation: Conservation of Mass I

- We still need to take into account the pressure term ∇p of the Navier-Stokes equations. We can do this by forcing our modified velocity field, $\vec{u}^*$ to be incompressible.
- For a specific time step we have

$$\frac{\vec{u}^* - \vec{u}}{\Delta t} = -\nabla p \tag{12}$$

- Multiplying by ∇ we get:

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

24. Implementation: Conservation of Mass II

-

$$\nabla^2 p = \frac{1}{\Delta t}\nabla \cdot \vec{u}^* \tag{14}$$

- This is a Poisson equation with a Neumann boundary condition, where the pressure is constant at the boundary
- We can then get our final velocity field $\vec{u}$ by subtracting out the pressure gradient:

$$\vec{u} = \vec{u}^* - \Delta t \nabla p \tag{15}$$

25. Implementation: Conservation of Mass III

- The Poisson equation is then solved using an iterative method. The result is a sparse system of linear equations. The system is symmetric, so the paper uses a conjugate gradient method to solve the system.
- The paper also uses methods to accelerate convergence. Their implementation runs for 20 iterations before yielding visually acceptable results

26. Implementation: Temperature and Density

    - Finally, we can update our temperature and density with the equations from earlier.
    - Since these are pure advection equations, we can solve using the semi-Lagrangian method.
    - The only difference is that we need to trace back to voxel centers rather than faces like we had to do with the velocity

27. Results I

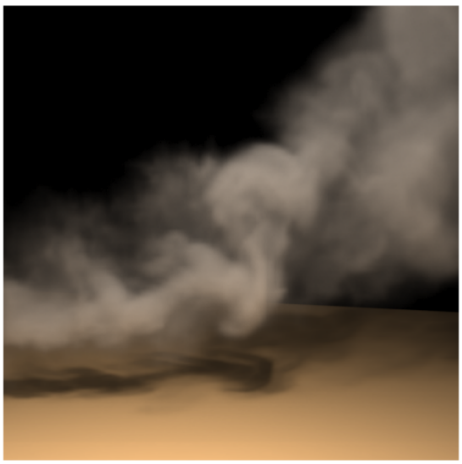

Figure 2: Rising smoke. There is no external force, only the buoyancy.

28. Results II

29. Results III

30. Results IV

    - While not real time, running the simulation and the hardware renderer was about 1 second per frame on a 40x40x40 grid.
    - On a 20x20x40 grid, the simulation time for the linear interpolator was 0.1 seconds per frame. For the cubic interpolator, it was 1.8 seconds per frame.
    - More complex simulations ran from around 30 to 75 seconds per frame.

31. Further Work

    - The method introduced by Fedkiw et al. for reintroducing vorticity is not physically correct; it injects extra energy

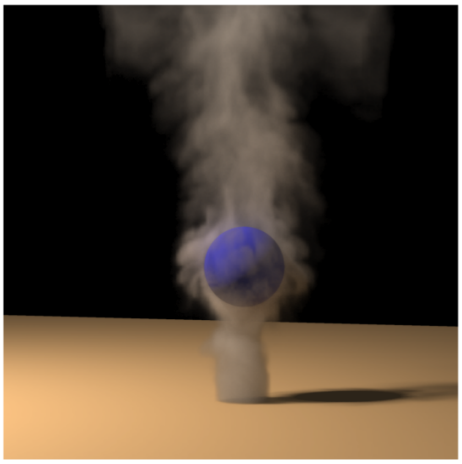

Figure 3: Smoke correctly interacting with potentially moving objects.

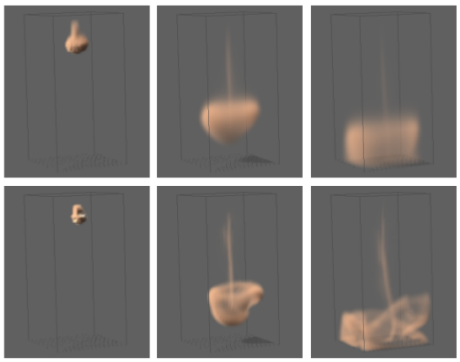

Figure 4: Comparison with the linear interpolator (top) and cubic (bottom).

- More modern methods of computing these kinds of small-scale detail are a Lagrangian approach introduced by Narain, Sewall, Carlson, and Lin and a wavelet approach by Kim, Thürey, James, and Gross.

32. References