

Visual Simulation of Smoke

Nicolas Morales

November 27, 2012

Goals

- ▶ Smoke simulation for rendering applications
- ▶ The method needs to be physically correct or at least plausible
- ▶ For fast fluid computation we should use a coarser grid
- ▶ We need to have a realistic approximation that avoids numerical dissipation

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.

The Equations of Fluid Flow I

- ▶ Generally, in fluid dynamics, we use the Navier-Stokes system of equations:

$$\nabla \cdot \vec{u} = \vec{0} \quad (1)$$

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

- ▶ \vec{u} is the velocity field
- ▶ p is the pressure
- ▶ ν is the viscosity of the fluid
- ▶ \vec{f} is the sum of all external forces on the fluid
- ▶ The first equation represents the conservation of mass.
- ▶ The second equation represents the conservation of momentum of the fluid.

The Equations of Fluid Flow II

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

$$\nabla \cdot \vec{u} = \vec{0} \quad (3)$$

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

- ▶ This is just the normal Navier-Stokes equations with the viscosity vector removed.

The Equations of Fluid Flow III

- ▶ The term $-(\vec{u} \cdot \nabla)\vec{u}$ is also known as the advection term. It describes 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:

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

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

The Equations of Fluid Flow IV

- ▶ Fedkiw et al. use the calculated temperature and density to compute the buoyancy force:

$$\vec{f}_{buoy} = -\alpha\rho\vec{z} + \beta(T - T_{amb})\vec{z} \quad (7)$$

- ▶ T is the temperature
- ▶ T_{amb} is the ambient temperature of the system
- ▶ \vec{z} points in the upwards direction
- ▶ α and β are magic constants

Numerical Dissipation

- ▶ Solving the advection equations with a coarse grid smooths out fine grain detail called vortices
- ▶ Vortices are curling motions of the fluid velocity field
- ▶ This smoothing out is called numerical dissipation

Vorticity Confinement I

- ▶ Add vortices back in to get a curling smoke effect.
- ▶ The method to do this is called Vorticity Confinement

Vorticity Confinement II

- ▶ We could perturb the field randomly but this causes strange artifacts
- ▶ Instead we'll look for good locations in the fluid to add in vortices

Understanding Vorticity

- ▶ Vorticity is the curl of the velocity of the field:

$$\omega = \nabla \times \vec{u} \quad (8)$$

Vorticity Confinement III

- ▶ In this method, we compute normalized vorticity location vectors:

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

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

- ▶ N is the normalized vorticity location vector.

Vorticity Confinement IV

- ▶ Using our normalized velocity location vectors, find a force that spins around areas of high vorticity:

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

- ▶ N is the normalized velocity location vector computed in the previous slide
- ▶ $\epsilon > 0$ is used to control the amount of detail added back into the flow field
- ▶ h is the distance step between grid cells
- ▶ This technique was introduced by Steinhoff for modeling the flow fields around helicopter rotors.

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.

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 except at the boundary voxels, where they are set to the density of the closest empty voxel

Implementation: Overview

- ▶ Two grids: read and write
- ▶ The user can set an initial grid or it can be empty
- ▶ Velocity update:
 - ▶ Adding in external forces
 - ▶ Solve for the advection term
 - ▶ Force the velocity field to conserve mass
- ▶ Temperature and density update

Implementation: Overview

- ▶ Two grids: read and write
- ▶ The user can set an initial grid or it can be empty
- ▶ Velocity update:
 - ▶ *Adding in external forces*
 - ▶ Solve for the advection term
 - ▶ Force the velocity field to conserve mass
- ▶ Temperature and density update

Implementation: External Forces

- ▶ User forces
- ▶ Buoyancy forces computed using temperature and density
- ▶ Vorticity confinement force
- ▶ Finite difference method to figure out effect on the velocity

Implementation: Overview

- ▶ Two grids: read and write
- ▶ The user can set an initial grid or it can be empty
- ▶ Velocity update:
 - ▶ Adding in external forces
 - ▶ *Solve for the advection term*
 - ▶ Force the velocity field to conserve mass
- ▶ Temperature and density update

Implementation: Advection Term I

- ▶ Semi-Lagrangian method for solving for the advection term.
- ▶ What we're trying to figure out is how the velocity field changes from advection

Semi-Lagrangian Methods I

- ▶ Frames of reference
 - ▶ Eulerian
 - ▶ Lagrangian

Semi-Lagrangian Methods II

- ▶ Semi-Lagrangian method
 - ▶ Combines Eulerian and Lagrangian frames of references
 - ▶ Examine the path of a particle that ends up in a specific grid point

Semi-Lagrangian Methods III

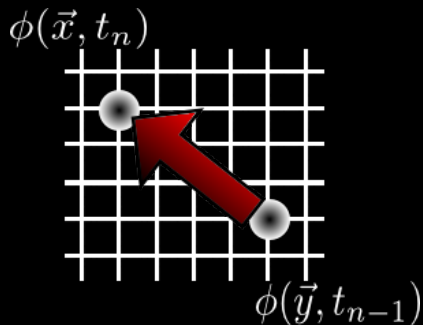


Figure: In the semi-Lagrangian method, a particle is traced back to its old position

Semi-Lagrangian Methods IV

- ▶ Have some function $\phi(\vec{x}, t)$ that takes a point on our grid and a time.
- ▶ We want to know the value of $\phi(\vec{x}, t)$ at a particular node \vec{x} and time t_n .
- ▶ The particle that is at node \vec{x} at time t_n was at some other point \vec{y} at time t_{n-1} .
- ▶ Estimate the old position of the particle:

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

- ▶ $\phi(y, t_{n-1})$ gives us approximately $\phi(x, t_n)$

Semi-Lagrangian Methods V

- ▶ Interpolate if our point \vec{y} does not lie on a grid node
- ▶ Linear interpolation is used for speed and stability
- ▶ Fedkiw et al. introduces a cubic interpolator that's slow but stable
- ▶ Paths are clipped against occupied voxels

Implementation: Advection Term II

- ▶ Use this equation for the advection term:

$$\frac{\vec{u}^* - \vec{u}}{\Delta t} = -(\vec{u} \cdot \nabla)\vec{u} \quad (12)$$

Implementation: Overview

- ▶ Two grids: read and write
- ▶ The user can set an initial grid or it can be empty
- ▶ Velocity update:
 - ▶ Adding in external forces
 - ▶ Solve for the advection term
 - ▶ *Force the velocity field to conserve mass*
- ▶ Temperature and density update

Implementation: Conservation of Mass I

- ▶ Solve for the pressure term ∇p field, \vec{u}^* to be incompressible.
- ▶ For a specific time step we have

$$\frac{\vec{u}^* - \vec{u}}{\Delta t} = -\nabla p \quad (13)$$

- ▶ Multiplying by ∇ we get:

$$\frac{1}{\Delta t} \nabla \vec{u}^* = \nabla^2 p \quad (14)$$

Implementation: Conservation of Mass II



$$\nabla^2 p = \frac{1}{\Delta t} \nabla \cdot \vec{u}^* \quad (15)$$

- ▶ 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:

$$\vec{u} = \vec{u}^* - \Delta t \nabla p \quad (16)$$

Implementation: Conservation of Mass III

- ▶ To solve the Poisson equation use an iterative solver like the conjugate gradient method
- ▶ Choleski preconditioner to accelerate convergence

Implementation: Overview

- ▶ Two grids: read and write
- ▶ The user can set an initial grid or it can be empty
- ▶ Velocity update:
 - ▶ Adding in external forces
 - ▶ Solve for the advection term
 - ▶ Force the velocity field to conserve mass
- ▶ *Temperature and density update*

Implementation: Temperature and Density

- ▶ These are pure advection equations, so 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

Results I

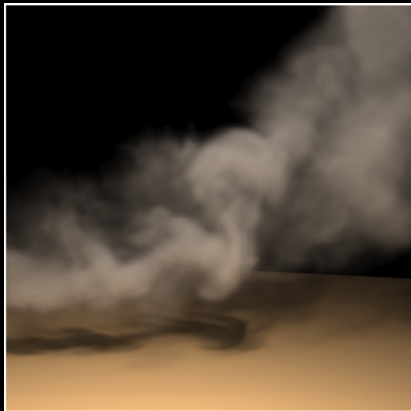


Figure: Rising smoke. There is no external force, only the buoyancy.

Results II

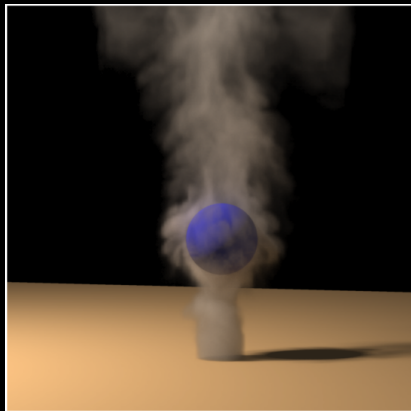


Figure: Smoke correctly interacting with potentially moving objects.

Results III

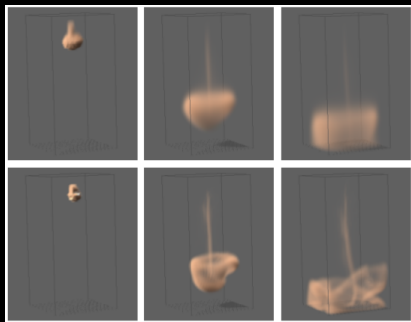


Figure: Comparison with the linear interpolator (top) and cubic (bottom).





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.




Further Work

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

References I

-  T. Brochu.
Fluid simulation for video games.
<http://software.intel.com/en-us/articles/fluid-simulation-for-video-games-part-1>.
-  T. Brochu.
Semi-lagrangian time integration.
<http://www.cs.ubc.ca/~tbrochu/projects/semil.pdf>.
-  A. Chorin.
A numerical method for solving incompressible viscous flow problems.
Computer Graphics, 1967.
-  R. Fedkiw, J. Stam, and H. W. Jensen.
Visual simulation of smoke.
ACM Siggraph 2001, 2001.

References II

-  T. Kim, N. Thürey, D. James, and M. Gross.
Wavelet turbulence for fluid simulation.
ACM Siggraph 2008, 2008.
-  R. Narain, J. Sewall, M. Carlson, and M. Lin.
Fast animation of turbulence using energy transport and procedural synthesis.
ACM Siggraph Asia 2008, 2008.
-  J. Stam.
Stable fluids.
ACM Siggraph 1999, 1999.