Interactive, real-time fluid simulator for the visualization of smoke

André C.M.J. Gillan, University of Montana, 2020

Introduction

Fluid motion is responsible for many complex real-world effects that are beautiful and visually appealing: smoke, flickering flames, crashing waves, cloud formation and motion, and explosions are all phenomena that are described, at least partially, by fluid motion. Animating fluid phenomena is important because it provides a sense of realism to an animation that a subject can compare with their real-life experience of fluid phenomena. The vast majority of fluid behaviors of interest in animation can be described by the incompressible Navier-Stokes equations, which are a set of partial differential equations describing the motion of a fluid in response to pressure and other forces (Bridson, 2016, p. 3). These equations can be written as:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u} \qquad (\mathbf{1.1})$$

$$\nabla \cdot \vec{u} = 0 \qquad (\mathbf{1.2})$$

In these equations, $\vec{u}$ is a vector of fluid velocity in two- or three- dimensions, ρ is the density of the fluid, p is pressure, ν is the kinematic viscosity of the fluid, and $\vec{g}$ is the sum of external body forces on the fluid (in this case $\vec{g}$ is used because frequently, only gravity is of interest).

The first equation is called the momentum equation and is essentially a restatement of Newton's Second Law $\vec{F} = m\vec{a}$ for a fluid modeled as a continuum. The second equation is the incompressibility condition, and specifies that the volume doesn't change with time. Even though real fluids such as air and water do change volume in response to pressure, under normal circumstances the effect is minute, complex to simulate and is unnecessary for visual simulations (Bridson, 2016, p. 11).

Equation **1.1** above is restated in (Stam, 2003) as:

$$\frac{\partial \vec{u}}{\partial t} = -(\vec{u} \cdot \nabla)\vec{u} + \nu \nabla^2 \vec{u} + \vec{f} \qquad (\mathbf{2.1})$$

where the the body force term $\vec{g}$ is instead notated as $\vec{f}$. Additionally, the notation ∇^2 is used instead of $\nabla \cdot \nabla$ for the Laplacian operator, and the terms are rearranged. The rearrangement of terms in equation **2.1** is given to show the similarity with another differential equation for the movement of density (such as smoke particle density) in the velocity field given in equation **2.2** below:

$$\frac{\partial \rho}{\partial t} = -(\vec{u} \cdot \nabla)\rho + \kappa \nabla^2 \rho + S \qquad (\mathbf{2.2})$$

The similarity between these two equations permits the same functions to compute density evolution and velocity evolution according to the method in (Stam, 2003). The term $-(\vec{u} \cdot \nabla)\rho$ describes the advection of density along the velocity field (which is analogous to $-(\vec{u} \cdot \nabla)\vec{u}$ in equation **2.1**, whereby velocity is self-advected in the velocity field). The term $\kappa \nabla^2 \rho$ describes simple diffusion of density according to a diffusion coefficient κ (which is analogous to $\nu \nabla^2 \vec{u}$ in equation **2.1**, the term for viscous diffusion). Often, the calculation of the viscosity term is ignored when simulating fluids for animation, in order to simplify the calculations and because numerical dissipation overwhelms the actual viscosity of the fluid which is being simulated (Bridson, 2016, p. 39). Lastly, S identifies a source of density such as the end of a cigarette in the animation.

Finding approximate solutions for the incompressible Navier-Stokes equations involves discretizing the fluid properties in both time and

space. The original method in (Stam, 2003) is unconditionally stable, because numerical dissipation and linear interpolation in self-advection ensures that new velocities calculated never grow larger than the velocities used from the previous time step for calculation without added force. Though this restricts the physical accuracy of the simulation by introducing an artificial dampening, the stable behavior allows for the use of an arbitrary time-step or a changing time step that is especially valuable when conducting a real-time simulation (Stam, 1999). The fluid solver in (Stam, 2003) uses a body-centered grid in which velocity values and additional attributes (such as density) are stored in the center of a 2- or 3- dimensional cell. This structure has benefits in simplifying the code for the fluid solver, especially because it allows for the same functions used in solving parts of equation **2.1** for velocity to be reused in solving parts of equation **2.2** for density.

Another fluid solver, the marker-and-cell method (Harlow et al., 1965) uses a a staggered grid system for storing velocity components in the grid. The benefit of using a staggered grid is that more accurate central differences can be used for computing gradients of smoke density, temperature and for computing the divergence of the velocity field, as opposed to using forward or backward differences (Fedkiw et al., 2001). Using a staggered grid should not increase the computational cost compared to storing velocity vectors in a body-centered cubic grid as proposed in (Stam, 2003), though the implementation is more complicated.

Method

We have created a fluid solver based on (Stam, 2003), but with some modifications for improved accuracy and for specialization in visual simulation of smoke. This includes adaptation of the method in (Stam, 2003) to use a staggered grid for velocity values as demonstrated in (Harlow et al., 1965; Fedkiw et al., 2001); a requirement for this is that functions are required to have somewhat different behavior depending on whether velocity or other material properties are being calculated.

The basic procedure for simulating fluids in this solver is roughly the same as that given in (Stam, 2003). Firstly, a user can interact with a window containing the fluid visualization by adding either forces or smoke density with the mouse; these are added to (initially) empty array(s) for the relevant property (the force values are stored in arrays which are the same size as those of the velocity grids, but note that at this point the arrays store force values, which have not yet been converted to velocities).

After smoke density or force values are obtained from the user interface using a `get_from_UI()` function, the velocity grid is updated sequentially through several steps in a `vel_step()` function. Firstly, the source force(s) from the `get_from_UI()` are converted to velocities by multiplying by the time step for the simulation, and added to the velocity arrays using the `add_source()` function. Next, the `diffuse()` function is used to introduce the effects of viscosity on the velocity field (if viscosity is included at all).

Next, the `project()` function is used to enforce the incompressibility condition. This function numerically solves the Helmholtz-Hodge decomposition of the velocity field to obtain the divergence-free component and the irrotational component (Stam, 2003). The divergence-free component of a velocity field satisfies the incompressibility condition, and effectively results in the nice swirly-like vortices that are desirable in a realistic fluid simulation. Several steps in the `vel_step()` require that the velocity field is incompressible, therefore the `project()` function is run several times in the course of one velocity step. Calculation of the projection step requires matrix inversion of a sparse matrix and so the iterative Gauss-Seidel relaxation method is used as in (Stam, 2003).

The next operation in the `vel_step()` is the self-advection of velocities using the linear backtrace method described in (Stam, 2003) and implemented as the `advect()` function. As the self-advection calculation modifies the velocity field, `project()` is again called after this step.

After this, the velocity field is modified with functions additional to those originally used in (Stam, 2003), in order to better simulate the properties of smoke. Modeling smoke requires including additional terms for the fluid's temperature and the density of smoke, both of which affect the fluid's velocity (Bridson, 2016, p. 99; Harlow et al., 1965). A simple model in which the temperature and smoke density are advected along with the fluid's velocity, and external forces are added to the fluid according to these properties, is used. In order to create a more realistic simulation of rising smoke, an additional (non-staggered) array holds a temperature value, and additional temperature is added to this array in the same locations that smoke density is added by the user during the `get_from_UI()` function. This temperature is then used during the `apply_buoyant_force()` function in which an upward force proportional to the location-specific temperature value and a constant buoyancy coefficient is applied to the simulation.

This simulation uses the same boundary conditions as given in (Stam, 2003), which models a closed 2-dimensional box in which no fluid can enter or exit. This condition is applied by setting the horizontal velocity outside a vertical wall to be the negative of the horizontal velocity inside the wall, and by setting the vertical velocity outside a horizontal wall to be the negative of the vertical velocity inside the wall. This is accomplished by adding an extra row of cells on every side of the box in which these boundary conditions are applied. The values of other properties (such as smoke density) are set to be the same in the boundary cells as inside the simulation area. This boundary condition is enforced by the `set_bnd()` method which must be run after any step that modifies the velocity or a material property such as smoke density. Other boundary conditions are possible, such as an inflow of material from across a boundary (which has been examined by the author), but in this simulation only the closed-box boundary condition is used. An combined effect of the closed-box boundary conditions and the temperature-dependent buoyant force is that rotating convection cells naturally form in the simulation, which results in attractive swirly motion of material throughout the simulation.

The last step in the `vel_step()` routine is to apply a vorticity confinement force to the velocity field. A vorticity confinement term allows for small-scale, turbulent features that are characteristic of smoke and other gaseous fluids. Adding a vorticity confinement term to the momentum conservation equations for fluid dynamics was first proposed in (Steinhoff et al., 1994) for modeling very complex flow fields in, for example, engineering applications, but it has also proven to be useful in overcoming the nonphysical numerical dissipation resulting from a coarse grid used for graphics simulations (Fedkiw et al., 2001). Vorticity (ω) is calculated in both 2- and 3- dimensions as the curl of the velocity field:

$$\omega = \nabla \times \vec{u} \qquad (\mathbf{5.1})$$

Note that vorticity is a vector field in the 3-dimensional case and a signed scalar field in the 2-dimensional case. We will use 2-dimensional notation here but the following equations would otherwise apply in the 3-dimensional case. Normalized vorticity location vectors that point from low vorticity to high vorticity are calculated according to:

$$\vec{N} = \frac{\nabla |\omega|}{|\nabla |\omega||} \qquad (\mathbf{5.2})$$

In practice, a small quantity is added to the numerator in the calculation of equation 5.2 to prevent divide-by-zero errors. Lastly, a vorticity confinement force (f_{conf}) for modifying the velocity field is calculated according to:

$$f_{conf} = \epsilon h(\vec{N} \times \vec{\omega}) \qquad (\mathbf{5.3})$$

In the 2-dimensional case, the vector $\vec{\omega}$ is calculated by setting a 3-d vector with the scalar value of ω constant in the z-dimension and no value in the x- and y- dimensions. This results in a cross product in the same plane as $\vec{N}$ (and in the same plane as the 2-d velocity field of the simulation). In equation **5.3**, h is the inter-grid spacing and ϵ is a vorticity confinement factor chosen according to visual effect. A value much less than 1 seems to work well. As in all cases in the simulation where a force is applied, f_{conf} is converted to an addition into the velocity field by multiplying the force by the time-step dt. All of this is performed in a function named `apply_vorticity_confinement()`, after which the `project()` routine is called once more at the conclusion of the `vel_step()` routine. It is important to note, however, that the addition of the vorticity confinement term means that the fluid simulation is no longer strictly stable; however in practice the simulation is unlikely to "blow-up" because of the small value of the vorticity confinement term ϵ chosen and the small number of discrete vorticity centers in a simulated fluid where this term is applied in full force.

After the `vel_step()` updates the simulation velocities, a density step routine updates the smoke density. Three separate arrays store densities for the three primary colors (red, green and blue); each function of the density step routine is applied separately for each of the colors to allow for movement and mixing of separate colors.

Most routines in the `den_step()` procedure are directly analogous to routines in the `vel_step()` procedure. Firstly, the `add_source()` function is used to add smoke density that was stored in the density arrays during the `get_from_UI()` phase. Next, the `diffuse()` routine is used to model diffusion of smoke density from high concentration to low concentration; diffusion in 2-dimensions only occurs between a grid cell and its four direct neighbors as in (Stam, 2003). Finally, smoke density is moved along the fluid velocity field using the `advect()` routine. In addition to these routines used in (Stam, 2003), the realism of the simulation is enhanced with a `diffuse_away()` function which reduces the smoke density in each cell by an amount proportional to the density in the cell, in order to simulate a dissipation of visible smoke over time. These routines applied for smoke density are also applied to the temperature array during the `dens_step()` procedure, though the coefficients used can differ from those used for smoke density.

The overall procedure for the simulation is summarized below. Functions that occur in the original algorithm in (Stam, 2003) are bolded, though the implementation details will differ from the code given in (Stam, 2003):

```
get_from_UI()
vel_step():
      add_source() #forces from UI
      diffuse()  #viscous diffusion
      project()
      advect()  #self-advection
      project()
      apply_buoyant_force()
      apply_vorticity_confinement()
      project()
dens_step(): #density and temperature
      add_source()
      diffuse()
      advect()
      diffuse_away()
```

Implementation

This interactive fluid simulator was implemented in Python 3 and tested in Ubuntu using Python 3.6.3. The simulator has several dependencies that require installation. NumPy is used for storing multidimensional numeric data and efficient array operations. PyOpenGL is a Python binding to OpenGL, GLU and GLUT used for displaying the simulation graphics. Pygame and ThorPy are used for the GUI toolbox and widgets. NumPy, PyOpenGL, Pygame and ThorPy can all be installed using pip.

Starting the application opens a square, black window with the title "Fluid Solver with

Staggered Grid” and a tall, narrow window titled “Toolbox” containing various buttons and sliders. The application as initially loaded is shown below in figure 1.

The toolbox to the right shows the settings upon initialization. The top of the toolbox shows four squares of different colors; clicking on one these squares allows the user to select a color for smoke that will be subsequently added when interacting with the simulation window. Initially, the color of smoke added will be white.

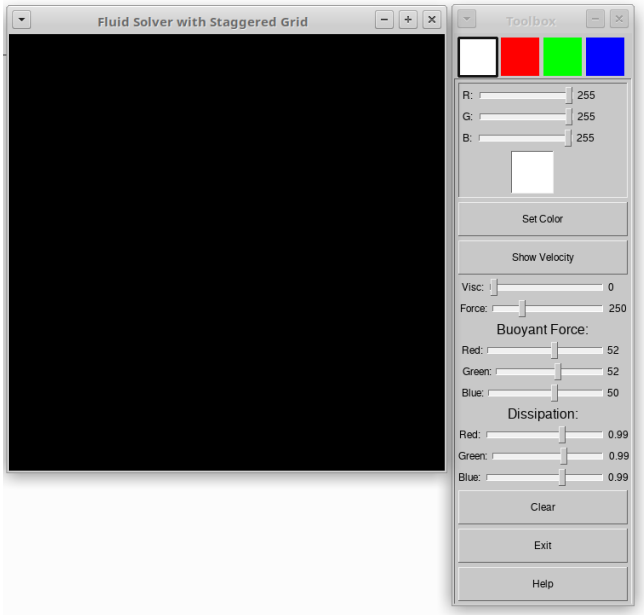

Figure 1: The application upon initialization, with the simulation window to the left and the toolbox window to the right.

Below the selectable color squares is a window with sliders for changing the the color settings to a user-defined color. Double clicking a selectable color box allows the user to select a color for modifying; this color box will then be surrounded with a thick black border and will also show up in the window below with the sliders set to the appropriate RGB values. The user can click and drag these sliders to define a new color, and the color box below will show the color represented by these new RGB values. When the user clicks the “Set Color” button, this new color will fill in the color box at the top selected with the black border, and the user can subsequently use this color in the simulation (by single-clicking its box at the top).

Below the “Set Color” button is a button that reads, initially, “Show Velocity”. Upon starting the simulation and whenever this button reads “Show Velocity”, the user is in density mode and can view an animation of the smoke density in the simulation. An example of density mode in the simulation window is show in figure 2 below.

Figure 2: The simulation window viewed in density mode.

Upon clicking the “Show Velocity” button, the simulation window enters velocity mode, in which the velocity at each grid point in the simulation is shown with a line rendered in the appropriate direction and with a length that scales with the velocity at that point. The lines are rendered in the color chosen by the user for rendering smoke. This button on the toolbox now reads “Show Smoke Density” instead. An image of the simulation window in velocity mode is shown in figure 3 on the next page.

The simulation mode can also be toggled between density mode and velocity mode by pressing the “v” key on the keyboard.

Below the “Show Velocity” button is a viscosity slider that allows the user to change the simulation viscosity. Increasing the viscosity makes fluid movement slow down more quickly and a very high viscosity can cause the simulated smoke density to stop moving entirely.

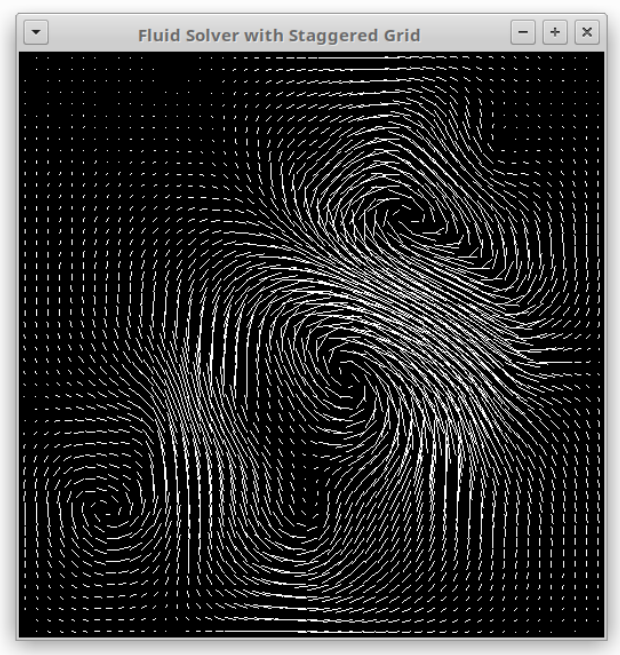

Figure 3: The simulation window viewed in velocity mode.

Below the viscosity slider is a force slider that allows the user to select the amount of force introduced by left-clicking and dragging in the simulation window. By increasing the force value, the magnitude of the velocities introduced when left-clicking is also increased.

Below the viscosity slider is a trio of sliders that allows the user to select the magnitude of the buoyant force that is calculated for each of the three primary colors. Because the density step is calculated independently for each of the three primary colors, a different buoyant force can also be calculated for each color. The result of having different buoyant forces can create interesting visual effects, such as having one color rise while another falls (due to a negative buoyant force).

Below the buoyant force sliders, there are also sliders allowing the dissipation coefficients to be modified for each of the primary colors. A dissipation coefficient less than 1.0 results in smoke density for that color disappointing over time if no additional density is added by the user. A dissipation coefficient of exactly 1.0 means that the amount of smoke density does not change (unless added by the user), but can be moved around the simulation by adding forces. A dissipation coefficient of greater than 1.0 is also possible, this results in smoke density growing and expanding outward as soon as it is added to the simulation. The possibility for setting different dissipation coefficients for different colors can result in some interesting visual effects, such as white smoke separating into different colors as one color grows with a dissipation coefficient greater than 1.0 while another slowly disappears.

Below the dissipation sliders is one more set of buttons. The “Clear” button sets all density, velocity and temperature in the simulation to 0, creating a empty screen as if the simulation was restarted. The simulation can also be cleared by pressing “c” on the keyboard. The “Help” button opens an html document in the web browser that gives some basic instructions for using the simulation. The “Exit” button closes all the windows in the simulation. This can also be accomplished by clicking the “x” button at the top-right of either window or by pressing the “Esc” key on the keyboard.

Bibliography

Bridson, R. (2016). *Fluid simulation for computer graphics* (2nd ed.). Boca Raton, FL: CRC Press.

Fedkiw, R., Stam, J., & Jensen, H. W. (2001). Visual simulation of smoke. *Proceedings of the 28th Annual Conference on Computer Graphics*

and Interactive Techniques - SIGGRAPH 01. doi: 10.1145/383259.383260
Harlow, F. H., & Welch, J. E. (1965). Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. *Physics of Fluids*, *8*(12), 2182. doi: 10.1063/1.1761178

Stam, J. (1999). Stable fluids. *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH 99*. doi: 10.1145/311535.311548

Stam, J. (2003). Real-time fluid dynamics for games.

Steinhoff, J., & Underhill, D. (1994). Modification of the Euler equations for ‘‘vorticity confinement’’: Application to the computation of interacting vortex rings. *Physics of Fluids*, *6*(8), 2738–2744. doi: 10.1063/1.868164