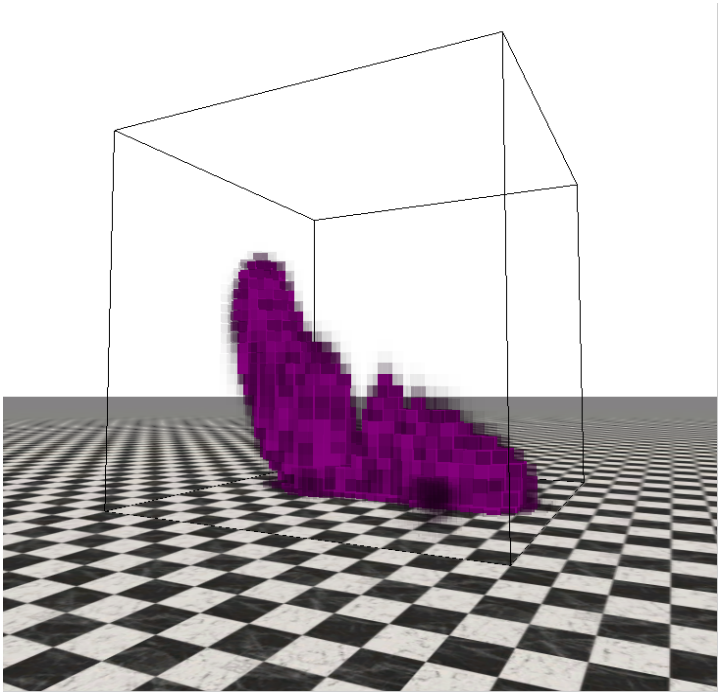
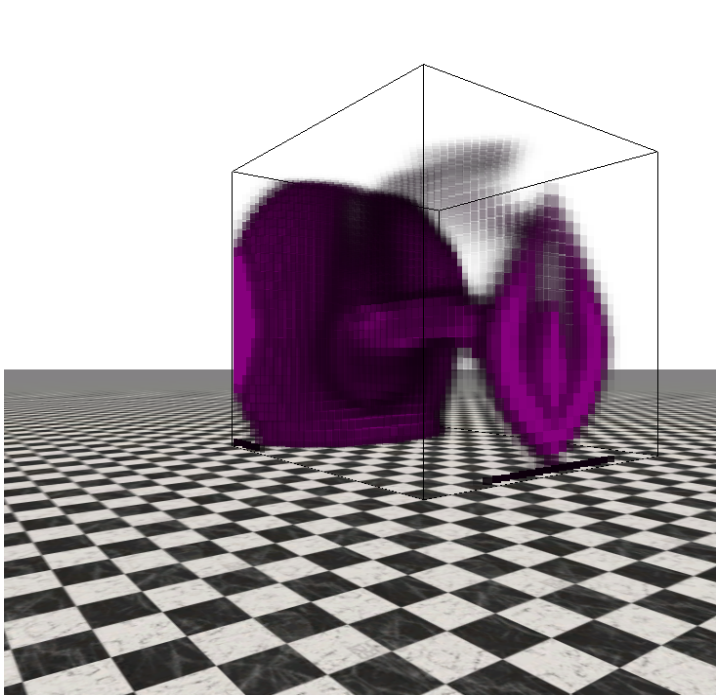
3D Stable Fluids
CS 395T: Physical Simulation and Animation for Computer Graphics
Thursday, December 4, 2014
Kevin Yeh

This program is equipped with basic 3D fluid interactions. The implementation is based off of the Stable Fluids paper and presentation by Jos Stam.

Fluids are rendered using GL_POINTS at a large radius, so it will only be visible at certain angles. No true voxel rendering is provided at this time.

On startup, the user can click and drag on the screen to generate a very dense fluid particle at (x,y,gridSize/2). This will diffuse rapidly when the simulation is running to create a large stream of fluid, so very small user interactions go a long way. When the fluids drag force checkbox is enabled, mouse dragging will instead apply external forces on the system, again at (x,y,gridSize/2). This is much more noticeable if gravity is turned off. The bounds will keep fluid density in, as long as the gravity is very small.

Tunable Parameters: Diffusion Rate, Timestep, Gravity, User Drag Force

I. Mathematical Background

The goal of this project was to implement the stable Navier-Stokes fluid methods described in Jos Stam's Stable Fluids paper and presentation. This project does not implement speed and accuracy optimizations mentioned in the paper, such as FFT and Vorticity Confinement. Instead, Gauss-Seidel iterative relaxation methods are used to avoid expensive linear system solvers.

The paper is based off of the Navier-Stokes equations discussed in class:

$$\nabla \cdot \mathbf{u} = 0 \qquad (1)$$

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f}, \qquad (2)$$

Where *u* is the velocity, *p* is the pressure, *v* is the fluid viscosity, and *f* is the addition of external forces on the system.

The paper introduces a stable method of solution that utilizes the *Helmholtz-Hodge Decomposition* to decompose the velocity field into a mass-conserving field and the associated gradient. Following this decomposition is the reduction of equations (1) and (2) into the mass-conserving u-projection

$$\mathbf{u} = \mathbf{P}\mathbf{w} = \mathbf{w} - \nabla q.$$

and the velocity equation

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{P}\left(-(\mathbf{u} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \mathbf{u} + \mathbf{f}\right),$$

upon which the technical pipeline is defined.

Technical Pipeline

The method of solution for the velocity of each grid cell at the next time step consists of four main components: external force addition, diffusion, advection, and mass-conservation projection. Additionally, bounds are enforced to reduce the amount of the fluid that exits the system through the grid walls.

External forces are very simple to exert on the system. Simply add the force, multiplied by the time step, to the velocity field.

$$\mathbf{w}_1(\mathbf{x}) = \mathbf{w}_0(\mathbf{x}) + \Delta t\, \mathbf{f}(\mathbf{x}, t)$$

Diffusion accounts for the effect of fluid viscosity, and can be discretized as shown in class by adding the velocity gradients to each neighboring grid cell, multiplied by the viscosity constant, time step, and the volume of the grid cell, h^3.

Advection accounts for the propagation of fluid disturbance throughout the system according to the expression

$$-(\mathbf{u} \cdot \nabla)\mathbf{u}.$$

which can be discretized as shown in class, tracing each grid particle back in time and trilinearly interpolating the previous velocities to move the velocity to the current grid particle position. This is unconditionally stable and very simple to implement.

Finally, projection involves the conservation of mass within the system according to the u-projection Poisson equation previously described.

$$\nabla^2 q = \nabla \cdot \mathbf{w}_3 \qquad \mathbf{w}_4 = \mathbf{w}_3 - \nabla q.$$

II. Technical Details

The technical implementation involves a continuous development of the density and velocity fields using the outline provided above. The fields are represented as Mat3D objects, a simple array container, and the velocity field is separated into three axis-aligned components for simplicity. All fields are initialized to zero-matrices.

Each grid is an N+2 x N+2 matrix, where N is the size of the fluid space. This provides a 1-unit buffer space along the borders for enacting boundary conditions.

Update Pipeline
The fluid solver pipeline is outlined in the Simulation::stableFluidSolve() method. Density is first diffused, then advected using the velocity fields. New velocity fields are then calculated through a process of external force addition, diffusion, projection, self-advection, and additional post-projection. Projection is applied twice, as advection results are more accurate. After each modification of the velocity field, bounds are applied to reduce the amount of fluid leaving the system.

Bounds
Bounds are applied using the Simulation::bound_mat() method. The borders are set to the closest inner values, or in the case of velocity matrices, to the negative of those inner values. This intuitively inserts a force that counteracts the force that is applied to the boundaries.

Rendering
The fluid space is rendered as a wire cube. Each grid cell is rendered mapping its density to the transparency of that cell, and using GL_POINTS with a large point size. This can be improved by using a vertex shader to draw actual voxels for each fluid particle. The grid size can be changed by modifying the scale variable. The fluid space resolution can be changed by modifying the param's gridSize variable. These controls are not provided by the GUI.

User Interaction
In addition to gravity, users can exert forces into the system and add fluid density into the space. The program defaults to density additions, allowing users to drag the mouse across the screen and generate dense particles at (x,y,gridSize/2). The concentrated density will diffuse quickly throughout the system. The default diffusion rate is .0001, and the default gravity is -3.4 m/s.

With the fluid drag force checkbox enabled, mouse dragging will instead generate a force in that direction at (x,y,gridSize/2). The force magnitude can be changed using the GUI, but defaults to 10,000.

Issues
Some issues have arised with this implementation, especially regarding fluid activity along the boundaries. In particular, with gravity enabled, fluids appear to leave the system and do not spread as expected naturally. However, applying forces to slam them into the walls does work as expected. This can best be experimented with using a gravity magnitude between -1 and 1. Larger magnitudes can also produce interesting effects by creating a small pool and applying a downward force to spread the pool.