Goals, Algorithms, and Components:

The goal of this project is to implement a physics-based fluid simulation using the Particle-in-Cell / Fluid-Implicit-Particle (PIC/FLIP) method.

The equation used to represent fluid mechanics is known as the Navier-Stokes momentum equation. This equation accounts for the 3 forces that can act on a fluid particle: gravity, pressure gradient force, and viscous force. However, for the purposes of the simulator, we care about the velocity. The derivative of this equation can be taken as shown in the figure. This new equation features $u \cdot \nabla u$ or the advection term and $\partial u/\partial t$ as the unsteadiness term. Advection refers to change of velocity due to the fluid's motion while unsteadiness describes local changes to the fluid velocity at a point independent of the fluid's motion. Set equal to the other side of the equation we still have gravity and pressure which acts as a gradient from high to low pressure represented by $-1/\rho \nabla p$ where ρ is density. Viscosity can be approximated to zero as seen in the figure. This gives us the Euler Momentum equation to use.

By convention, we view fluids as made up of infinitely small boxes of molecules. Knowing this, it's clear to see how implementing a box-like data structure to determine how forces are applied to particles would be useful. The staggered grid, or marker-and-cell (MAC) grid is utilized by fluid simulations to represent the boxes that contain particles of a simulation and to create an abstraction to calculate velocity acting on particles and between neighboring particles. For the purposes of our simulation, we create one in 3D. Each boundary face of a cell of the grid stores a velocity in its respective direction (u, v, w) for (x, y, z) respectively. The center of each cell stores the pressure of that cell acting as the pressure gradient.

Splat is the term for taking the velocity of a particle in a cell and applying its influence over neighboring cells. This creates a representation of the Eularian field of fluid velocities. We do this each timestep to update the velocities of the grid as particles move within it. To determine the influence of the velocity of a particle in a cell on neighboring cells, barycentric weights are computed. These are computed by "localizing" the coordinates for the particle in its grid by subtracting the lower corner pos from the particle pos to get (x*,y*,z*) and then dividing by delta x to get (i,j,k) the index of the cell in the grid, where delta x is the length of a cell. The weights are equal to (x*,y*,z*) / delta x - (i,j,k). Next, to calculate the weighted velocity contribution stored at a neighboring cell, or to splat, shift the position of the particle in a cell by delta x / 2 in each direction. Add to this velocity in the manner shown in the splat figure. Note that this is done in 3 dimensions so this shift is done 3 times with each dimension applied to the 8 potentially changed boundary velocities. Along with splatting comes the normalization of these velocities stored in the cell boundaries. For each splat's contributed velocity, accumulate the weights in the manner shown in the splat weight figure. Then divide all velocities by these weights once splatting is complete such that vel(i,j,k) = vel(i,j,k)/accum(i,j,k). An exception is made for at the outside boundaries of the staggered grid, where the velocities at the center and opposite face of the cell are equal to zero. The velocities of the faces perpendicular to the outermost face are equal to the velocity of the cell one layer inward.

Advection velocity change is done by averaging neighboring cells with barycentric weights again and applying that to a particle's velocity. To find advection(u,v,w) velocities, trilinear interpolation must be done with the shifted positions and weights. Adv(u), Adv(v), Adv(w)are calculated as shown in the advection figure. Next, the particle's position due to advection is calculated by pos + adv(up, vp, wp)Δt.

Gravity is the body force applied to fluids within the Navier-Stokes momentum equation. Each velocity stored after splatting in the y direction (denoted by v) has $9.80665 * \Delta t$ subtracted to apply this.

Pressure projection is done to apply the force that accounts for the conservation of the mass of the fluid. The term d is the velocity field's divergence equal to $\nabla \cdot u = \partial u/\partial x + \partial u/\partial y + \partial u/\partial z$ shown in the divergence figure. By the incompressibility constraint, the divergence should equal zero meaning these equations can be substituted for the next timestep's velocities as shown in the divergence figure.. The equation can then be rearranged and then divided by $\Delta t/(\rho(\Delta x)2)$ to get what is shown in the figure. This comes together to use the linear equation Ap = d to determine pressure in the staggered grid. p = A^-1*d is not reliable to find p because A is not always invertible. Because of this, the Conjugate Gradient Algorithm is utilized in the way shown in its figure. This allows for this linear equation to be used without A to be created for p to be found.

The final component of this fluid simulation is using PIC and FLIP. PIC interpolates the velocities closest to a particle and assigns the particle's velocity to be that new value. Meanwhile, FLIP updates the particle based on change in interpolated grid velocity. Because of this, FLIP doesn't have the same "filtering" or "smoothing" tendency as PIC so it preserves more of a turbulent behavior in fluids. Blending between these is useful to ensure a more realistic simulation. This method is called PIC / FLIP. It is done by using something called the FLIP ratio f used in the following way: up = f(up - uold) + unew. up is the particle's velocity. uold is the velocity before PIC has the aforementioned forces applied to the velocities in the staggered grid. unew is the velocity after these are applied. This equation in our fluid simulator acts such that f of 0 is purley PIC algorithm and f of 1 is purley FLIP. In the simulation done, the FLIP ratio is .95.

Source:

https://unusualinsights.github.io/fluid_tutorial/

Implementation

The implementation for this fluid simulator is based off of the tutorial below and is made to work for tests provided by it. The basis for the algorithm to complete is as follows, updating the particles in the simulation accordingly:

```
Initialize the time t = 0.
Start with a zero-divergence velocity field u(x, y, z, t).
For the duration of the simulation:
    Set u = Advect(u(x, y, z, t), Δt).
    Set u += Δtg.
    Set u = ProjectPressure(u, Δt) such that u has approximately zero divergence.
    Set t += Δt.
```

To achieve this, several data structures and methods to achieve these functions are created. Array3D is, as the name implies, a 3D array that has functions implemented for algebraic manipulation of its values and storing necessary values of any type. Particle is the file for the particle object that contains a position and velocity. The inputs folder contains parameters for setting up the simulation and an input file containing predefined Particles for it to use. These are read by SimulationParameters. FluidSimulator is the main file for running the simulator. With these defined, the files that contain the methods that do a majority of the work of the simulation as defined above can be described, StaggeredGrid and PressureSolver.

The Staggered Grid is built upon 9 3D arrays of around the size of the dimensions x,y,z of the grid each. These are: sp that stores pressure, su sv sw that stores x y z direction velocities respectively, fu fv fw that stores x y z direction accumulated weights respectively (and later pre PIC velocities), labels that store identifiers for cell type (SOLID,EMPTY,FLUID), and neighbors that stores unsigned shorts with bits mapped to neighbor cell info. su sv sw and fu fv fw do not have identical dimensions however because each has + 1 dimension in the direction of their velocities/weights to account for +/- delta x / 2 when splatting in each cell and velocities stored for 2 boundaries of a cell because of this.

The ParticlesToGrid method handles splatting to the staggered grid for each particle it takes as an input. First cells are reset to EMPTY on the inside and SOLID on the outside and 0 velocity. Next, a shift is applied in each respective dimension direction (u,v,w) down by delta x / 2 in 2 dimension directions and the arrays (su fu,sv fv,sw fw) are updated respectively in accordance with the splatting figure. Normalization is then applied on cell velocities against the weights. Next fu, fv, fw are set equal to su, sv, sw because fu, fv, fw no longer have a use and these velocities can be used for PIC/FLIP blending. Finally, outside boundary velocities are set in the way briefly mentioned in the components section.

The Advect method computes the advection of position in the grid by trilinearly interpolating as shown in its figure with a given particle position and su sv sw times t, time, added to the original position and then clamps the result and returns the new position.

The ApplyGravity method subtracts t*9.80665 from each velocity in sv (the y direction) and then resets boundary velocities for PIC as done before in the splat method (because they are incorrect after the subtraction).

The ProjectPressure method first resets pressures to 0 in sp. Next, each cell i,j,k has its neighbor info bits set in neighbors(i,j,k) to store information used to determine pressure as shown in the figure. PressureSolver now runs the algorithm to compute a pressure as described in the figure across each pressure in the grid. Finally, pressure is applied to each non solid cell of the grid by subtracting the pressure for (i,j,k) and (i-1,j-1,k-1) depending on the direction of the velocity from each of su sv and sw.

The GridToParticle method handles PIC / FLIP blending by taking in a ratio and particle. The method used for trilinear interpolation in Advect is utilized again but with pos with fu fv fw (which contains the velocities stored during splat before other forces are subtracted from the velocities and boundaries are modified) to get uold and pos with su sv sw (the final velocities for the timestep) to get unew. The aforementioned equation up = f(up - uold) + unew is calculated and the modified up is returned as the particle's velocity.

The final runner, FluidSimulator, functions in the following way: The time step is incremented, (particle)p->pos = Advect, ParticlesToGrid (splat) is done, ApplyGravity is done, ProjectPressure is done, and then p->vel is set equal to GridToParticle. This allows the outline for the simulation above. The result is shown in gifs.

Strange behavior occurs in the result of the PIC/FLIP when not pure PIC or a ratio 0. Some particles get stuck on the roof of the grid for a short period unrealistically. A major limitation of this implementation is the time it takes to run the simulation. It takes 1-2 minutes per particle configuration generated (1 frame of the GIF) to be computed on my machine making it take hours for a gif result to be created. While there are many particles interacting with each other at each timestep, optimization algorithms could likely be used to compute these interactions with greater efficiency.

Source:

https://unusualinsights.github.io/fluid_tutorial/