Fluid Simulation: Smoke and Collision

James Bao, Xin Wang

December 28, 2020

1 Introduction

In this project, we decided to simulate smoke colliding into an object. We attempt to generate a cloud of smoke that has to move through a bunny or a sphere. Throughout this paper, we will describe the methods we used, the data representation of the methods as well as our algorithm, followed by some of our results.

2 Related Work

For this project, we referenced 2 main sources. Firstly, we referenced videos and lecture slides on fluid simulation by David Levin [3] and a corresponding video found here. In addition, we also heavily referenced fluid notes [1] for various details. Other sources we referenced were the original Siggraph paper[2] introducing the method of fluid simulation we used, as well as other course notes [4].

3 Discretization

The smoke simulation lives within an axis-aligned *smoke box* – particles will only exist within the smoke box. We discretize the smoke box into a grid of square cells. Each cell holds information about the velocity, temperature, smoke density, etc. of the smoke particles at that location. In other words, the values in the cells define the behavior of smoke particles nearby. We can compute the velocity of a smoke particle using values in its surrounding cells.

3.1 Marker-and-Cell (MAC) Grid

Specifically, the grid is structured as a MAC grid, also known as a staggered grid. [3] [1] In each cell, we store the pressure, density, and temperature value in the center of the cell. For example, the temperature value would represent the temperature of a hypothetical smoke particle located in the center of the cell. In addition, we store the velocity in the center of cell faces such that the x velocity is stored in the faces perpendicular to the x-axis, and similarly for y and z velocities. We also store other values useful for vorticity confinement inside the centers and faces of the cell.

Notice that properties stored stored in the MAC grid (e.g. x velocities) each form their own grid. To compute the x velocity of an arbitrary point, we can find the cube in the x velocity grid surrounding that point, and perform *trilinear interpolation* to get an estimation. We can do the same for y and z velocities to compute the velocity of that point.

4 Algorithm

We initialize our MAC grid with random velocity values to emulate the air flow in our smoke box. Velocities at the boundaries normal to the smoke box face are initialized to zero. Temperature and density values in cells are initialized differently between cells that have and do not have smoke particles. This is to differentiate between ambient temperature versus smoke temperature and air density versus smoke density.

At every time-step, we update the values in the MAC grid based on physical properties of smoke, then we use the MAC grid to compute the velocity of each particle and move the particles along their velocity. We outline each step below.

4.1 Updating Velocity

4.1.1 Velocity Advection

We update the values of each velocity grid using the existing "velocity field" defined in the MAC grid. For a given grid-point at position p_t and velocity v_t – we can compute the velocity by averaging velocities at neighboring cells – we can compute its velocity at the next time-step using a semi-Lagrangian method. This grid-point's next velocity is equal

to the velocity of an imaginary particle that will arrive at this grid-point in the next timestamp. In other words, this imaginary particle's current velocity is the grid-point's next velocity. We can estimate this imaginary particle current location as $p_t - \Delta t v_t$ then interpolate the velocity at this position to get the grid-point's velocity at the next timestamp.

4.1.2 Buoyancy force and vorticity confinement

Beyond initial velocities provided by our smoke box, we also have external forces acting on each of our smoke particles. Primarily there are 2 external forces that affect our smoke particles. Firstly we have buoyancy force. Smoke naturally rises, but is also pulled down by gravity, in our code we enhanced the gravity effect and reduced the buoyancy force in order to achieve our desired effect. Since buoyancy only effect the vertical movement, this force only effects the vGrid. The code simply loops through all of our grid points, and applies a simple mathematical equation $f = -\alpha * d + \beta * (T - AmbT)$ [1], where f is the buoyancy force, α is the gravity, d is the average density, β is the buoyancy, T is the current temperature and $AmbT$ is the ambient temperature. We add this force on top of the current velocity to get a floating or sinking effect.

The other external force is vorticity confinement. This force is described as a boost to the smokes natural flow. This confines the dissipation of smoke and encourages the smoke to stay in local vortexes. By applying this force on our velocities, we essentially redirect some vectors to rotate rather than directly move away from the mass of smoke particles. This is achieved by storing the curls and curl gradients of each velocity in a staggered grid much like pressure.

4.1.3 Pressure Projection

In order to calculate the pressure of the particles in each grid of our staggered grid, we used pressure projection. This step helps reproduce the incompressible fluid property by effecting the velocities in the faces surrounding a pressure point. Following the descriptions from the lecture slides [3], we solved for the pressure projections using the surrounding velocities.

4.2 Temperature and Density Advection

Temperature and density advection is also computed using the semi-Lagrangian method, similar to velocity advection. The advection of these fields are influenced by the velocity field in the smoke box.

4.3 Smoke Particle Position Advection

We represented smoke particles in an nx3 matrix, where n is the number of smoke particles, each represented by a 3D position. To update each particle's location, we first compute their velocities – using trilinear interpolation on the velocity grids. If necessary, we shrink the computed velocity values to keep particles within the smoke box. In order to combat certain shortcomings of the Forward Euler time integration, we used second-order Runge-Kutta [1]. This is achieved by taking our "new position" (original position + time-step * velocity), computing the velocity of this new position, and averaging the two velocities obtained. This method is more stable and allows us to use a slightly larger time-step in our simulations.

In order to handle objects, we have to update our velocities and positions accordingly to avoid these objects. As mentioned before, our objects are either a bunny (represented through a matrix of points and faces, or a "mesh") or a sphere (represented through a constant radius and a center position). There are two points where we check for the object. We first check to see if a smoke particle is touching the object. If it is, we apply a normal force from the surface of the object to the smoke particle's original velocity, updating our velocity. The second play we check is if the smoke particle is already inside an object, in which case we move the smoke out to the nearest surface to adjust for this error.

5 Improvements and Future work

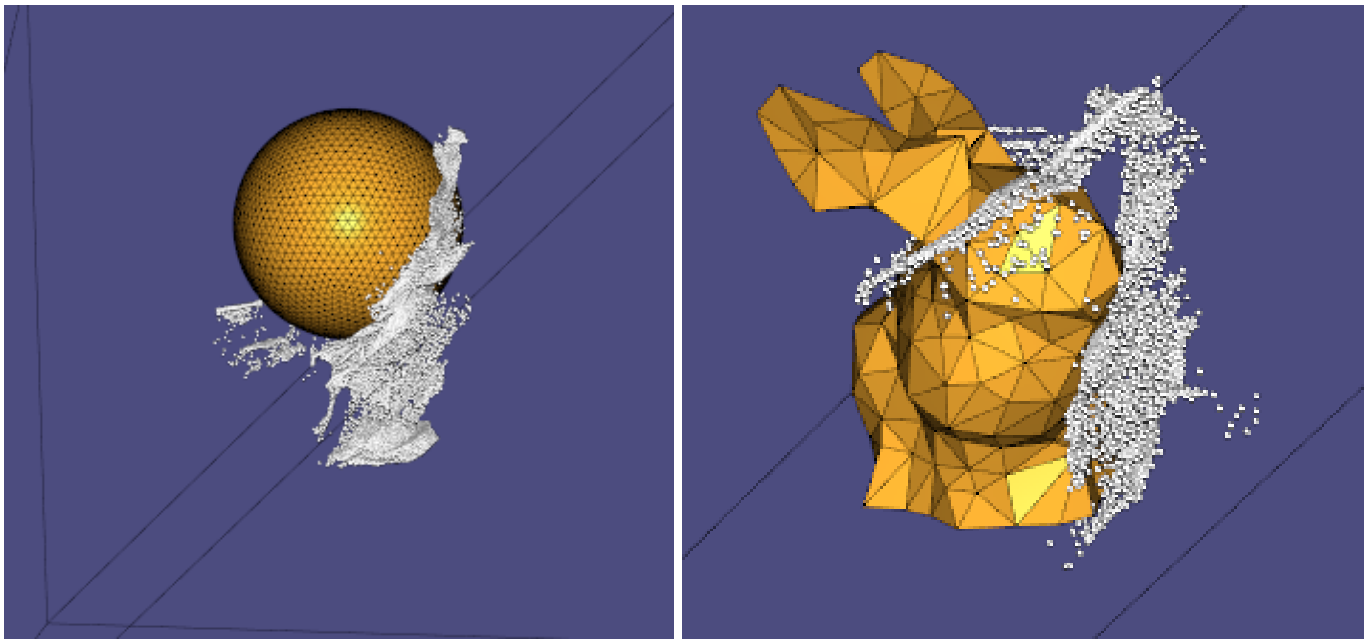
5.0.1 Sharper Interpolation

One feature we attempted to implement but was not ultimately successful in was implementing sharper interpolation as described in [1]. This would use cubic interpolation instead of a cubic interpolation, which counters semi-Lagrangian's property of smoothing out the velocity fields. This was not implemented as it is most noticeable in a large smoke simulation which we did not do.

5.0.2 Moving bunny

Another feature we wished to work on was to be able to drag the bunny and make it bounce around, and see how that would affect smoke. Due to time limitations and computer slowness, we chose not to implement this feature.

6 Results



References

[1] Robert Bridson and Matthias Müller-Fischer. "Fluid Simulation: SIGGRAPH 2007 Course Notes¡i¿Video Files Associated with This Course Are Available from the Citation Page¡/i¿". In: *ACM SIGGRAPH 2007 Courses*. SIGGRAPH '07. San Diego, California: Association for Computing Machinery, 2007, pp. 1–81. ISBN: 9781450318235. DOI: 10.1145/1281500.1281681. URL: https://doi.org/10.1145/1281500.1281681.

[2] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. "Visual Simulation of Smoke". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: Association for Computing Machinery, 2001, pp. 15–22. ISBN: 158113374X. DOI: 10.1145/383259.383260. URL: https://doi.org/10.1145/383259.383260.

[3] David Levin. *Physics-based animation lecture 10: Fluid Simulation*. URL: https://github.com/dilevin/CSC417-physics-based-animation/blob/master/lectures/10-fluid-simulation-final.pdf. (accessed: 12.27.2020).

[4] Aline Normoyle. *Fluid Simulation Tutorial*. URL: https://www.alinenormoyle.com/TutorialFluid.html. (accessed: 12.27.2020).