Real-time Grid Based Fluid Simulation

CS 284A: Computer Graphics and Imaging, Spring 2021

LLOYD BROWN, YINAN CHEN, SOPHIE WU, WEIYAN ZHU, UC Berkeley

ACM Reference Format:
Lloyd Brown, Yinan Chen, Sophie Wu, Weiyan Zhu. 2021. Real-time Grid Based Fluid Simulation: CS 284A: Computer Graphics and Imaging, Spring 2021. 1, 1 (April 2021), 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Our goal is to implement a grid-based fluid simulation in which users can interact with the grids using their mouse to trigger and interfere the fluid movement. To achieve this goal, we based our project on the Real-Time Fluid Dynamics for Games paper written by Jos Stam [Stam 2003], which suggests using a modified version of the Navier-Stokes Equations which is able to simulate the fluid in real-time while making the fluid look realistic. Eventually, we would like to realize a 3D fluid simulation where the fluid flow could respond to both user interactions and objects within the scene.

2 TECHNICAL APPROACHES

We have planned our progress to first realize the 2D fluid simulation. In this phase, we divided the work into four parts so everyone could work on the project simultaneously. Yinan was responsible for handling the user interactions with the grid and storing fetched user inputs as velocity and density arrays. Sophie and Weiyan worked together to build the framework in three.js and implemented algorithms to realize the fluid simulation. Lloyd was in charge of constructing the grid system, writing the shader, and rendering the scene. All members have contributed to the integration and debugging of the project.

2.1 User Input

User interactions with the grid would interfere with the fluid flow by applying directional velocities to particles and alter the density of particles within each grid. To fetch the input data, we listen to users' mouse events within the canvas. Velocity is determined by the mouse movements over the Δt of time. For every Δt, we accumulate each mouse movement's distance and use the $\frac{sum}{\Delta t}$ to get the velocity change over Δt for x and y directions. We then add this velocity to every grid that is underlying this mouse movement path. As for density, its increment is triggered by users' mouse click-down event and terminated when the mouse is released. The longer the user holds the mouse, the higher the density underlying the mouse area. This is realized by having the density accumulated by every frame to the grid that is nearest to the mouse's pixel position. User inputs are updated in real-time and organized as x and y velocity arrays and a density array for the use in the fluid solver.

Author's address: Lloyd Brown, Yinan Chen, Sophie Wu, Weiyan Zhu, UC Berkeley.

XXXX-XXXX/2021/4-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2.2 Fluid Simulation

Our fluid simulation algorithms are mainly based on the physical equations of fluid flow, the Navier-Stokes equations. These equations are notoriously hard to solve when strict physical accuracy is of prime importance. Our solvers on the other hand are geared towards visual quality. Our emphasis is on stability and speed, which means that our simulations can be advanced with arbitrary time steps.

Our fluids are modeled on a square grid, which contains the information of velocity and density as constants in each grid cell. The simulation is therefore a set of snapshots of the velocity and density grids.

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

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

Fig. 1. Navier-Stoke Equations for the velocity (top) and density (bottom)

2.2.1 Moving Densities. In this section, we will mainly talk about our simulation for the density field according to the equation above, which consists of three terms. The first term represents that the density should follow the change of velocity, the second implies the density will diffuse, and the third states the density increases due to sources.

During implementation, the first step is to make the density follow the velocity field. We use a technique that models the density as a set of particles. In this case, we can simply trace the particles through the velocity field.

The second term takes into account the possible diffusion at a certain rate. Considering a single cell, the cell exchanges densities only with its four neighboring cells. We consider a stable method for the diffusion step. The basic idea is to find the densities which yield the densities we start with when diffused backwards. We can obtain the amount of density these particles carry by linearly interpolating the density at their starting location from the four neighbors.

The third step would be relatively easy because we just filled in the density array based on the user's mouse movement.

2.2.2 Changing Velocities. In our velocity solver, again we want to consider the equations above. Our velocity solver resembles the density solver in many ways, but it requires a new routine called project, which forces the velocity to be mass conserving. Based on Hodge decomposition, we can get the mass conserving velocity field by removing the gradient field from our velocity field. To get this gradient field, we followed the paper to solve the Poisson equation. In the end, we have a mass conserving velocity field which looks more realistic.

We also have another routine to set the boundary. We assume the fluid is contained in a box with solid walls, which means the horizontal velocity should be zero on the vertical walls, and the vertical velocity should be zero on the horizontal walls.

2.3 Rendering

The last portion of our project was to perform the rendering of our simulation based on the density field outputs satisfying the Navier-Stokes equation. Our rendering was performed in Three.Js. Our first step is to instantiate the scene using a perspective camera looking in the -z direction. We then turn to rendering the geometry on the x,y plane at z = 0.

Our approach is to render a grid of colors based upon the densities given by Navier-Stokes. We instantiate this grid such that each cell is a single vertex with a given color at a unique position. We do this by creating a Buffer Geometry to which we add arrays containing the cells positions, size, and colors. We choose to render all of our cells in a normalized space of [-0.5, 0.5] in both x and y (this ensures the camera looks at the middle of the grid. To get the positions we add 0.5 to our x and y values, divide by the number of rows or columns (depending on the axis) and subtract 0.5.

Our rendering simulates fluid motion by changing the color in each cell based on the density. In each timestep we update the densities as described earlier and store them in a 1d array. We then loop through each of the cell's positions and convert them to the corresponding 1d index by undoing the earlier computation. That number is used to index into the density map returning a number of particles.

The last step is to use the number of particles to determine the current color of each cell. We assumed that the color of the water was additive such that with few particles the water would appear white and with many the water would appear a deeper shade of blue. To achieve this we start off the color as white and use the number of particles to interpolate the color between white and blue with more particles tending toward a blue color. Once we have this color we update the color array in the geometry and request an update from the geometry.

3 RESULTS

Our current product simulates the Navier-Stokes equations for the incompressible fluids. When the user clicks and drags their mouse inside the grids, density will be added in the nearest few grids and more smoke will show up. When the user moves their mouse cursor without the mouse down, the fluid will be moved like it is "stirred". The fluid is contained in a box with solid walls, so no fluid should exit the walls. Figure 2 shows our current progress.

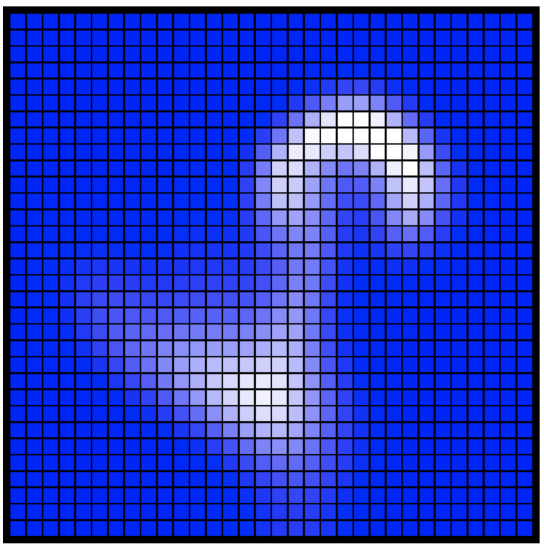

Fig. 2. Screenshots for our current product

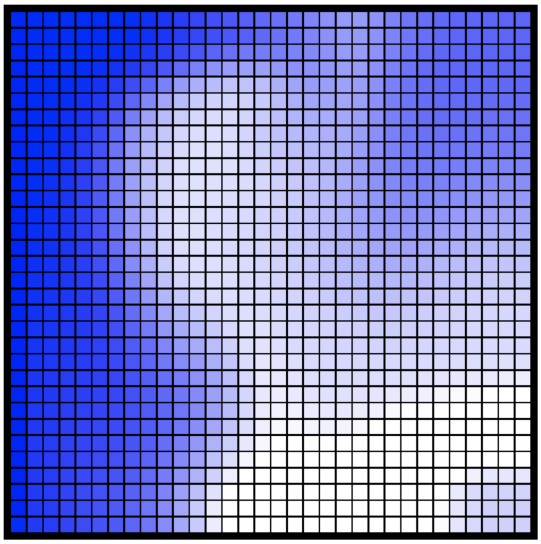

Fig. 3. Screenshots for our current product

4 NEXT STEPS

In our current simulation, there is some unexpected behavior happening in our leftmost column when applying density continuously, and fluid would not completely disperse as time goes. These are our first priorities to fix. Later, as Stam's paper suggests, there is an extension for us to realize the grid-based fluid simulation in a 3D environment. Thus, we would work to add the z-component to our existing 2D simulation system. We would then update the user interactions to accommodate the 3D scene. Instead of working with water, we feel it is more reasonable to simulate smoke flowing inside the 3D cube. Hence, we would integrate smoke-related parameters like buoyancy and temperature into our system and write a shader for the smoke. Eventually, we would have a GUI control

panel available on the web page for users to change all parameters. If time allows, we would try having the whole program running in GPU and consider adding more objects in the scene to interact with the smoke.

REFERENCES

Jos Stam. 2003. Real-time fluid dynamics for games. In *Proceedings of the game developer conference*, Vol. 18. 25.