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 (May 2021), 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Our goal is to implement a fluid-based music analyzer that allows user interaction. 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.

Previous work with fluid-based music renderings have performed real-time simulation [Bodonyi [n.d.]], but have lacked clarity into the current state of given frequencies. Without an understanding of where to look and what to look for, users will be unable to make proper use of a music analyzer. Our approach is to use a strict mapping between both the frequency and the location as well as the frequency and the color. This will enable users to know where to look to understand what's happening with a given frequency as well as what colors to look for to understand the amplitude of a given frequency.

2 TECHNICAL APPROACHES

2.1 Dynamic Inputs

Our fluid simulation uses velocity and density maps to store dynamic inputs that are updated per frame based on either user interactions with canvas or audio input. These dynamic inputs will apply directional velocities to grids and alter their densities, thus to simulate the fluid flow and color change.

2.1.1 User Interactions. For user interactions, we listen to users' mouse events within the canvas. Velocity is associated with the mouse-hover events and will be calculated per frame. In each frame, we accumulate the distance of mouse movement 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 following the mouse movement path. As for density, its increment is triggered by users' mouse click-down event and will be terminated when the mouse is released. Per frame user holding the mouse, we add the density to a circular region of grids underneath the clicked pixel. Therefore, the longer the user holds the mouse, the higher the density underlying the mouse area.

2.1.2 Audio Input. Our system is able to get audio input from MP3 files and the user's microphone. For both inputs, we rely on the Web Audio API to process the audio data. Once the user uploads a music file or the system receives an input from the microphone, an audio analyzer will be created. Per each frame, we use the analyzer's getByteFrequencyData() function to fetch the real-time amplitude value for every range of sound frequency. Each fluid represents a specific sound frequency range and is aligned in-order with lower frequency on the left and higher frequency on the right. We assign each fluid a fixed positive velocity to allow it to flow upward. The density change for each fluid is based on the real-time change of amplitude.

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) [Stam 2003]

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

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

XXXX-XXXX/2021/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 amount of density at these particles can be calculated by linearly interpolating the density from the four closest neighbors of the particles. We use the following procedure to update the density at each step. We focus on one grid that contains the density from the previous time step and one that will contain the new density. We trace the latter cell's center position backwards through its velocity, and then linearly interpolate the density values from the previous grid and assign the value to the current grid.

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. It is intuitive to simply add a portion p of density from each of the surrounding grinds to the current grid, and then remove a corresponding $4p$ amount of density from the current grid. However, this solution can result in unstable simulation for certain p values. We consider a stable method for the diffusion step instead. 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. As Jos Stam suggests, we utilized Gauss-Seidel relaxation, which is a simple linear solver that works well enough in our case. [Stam 2003]

The third step would be relatively easy because we just filled in the density array based on the user's mouse input, audio files input and microphone input. For the mouse input, we mapped the pixels from screen space to the grid space in our simulator. Based on the user's mouse position on the screen, we will add densities to grids at the corresponding locations in our simulator. For the audio files or microphone input, we always add densities at the bottom of the grids, and move them up by adding an upward velocity. We can get the amplitude at different frequency channels from the audio input. The frequency decides what horizontal positions, or columns in our grad we will add densities to. The lower frequencies reside at the left part of the grid, and the higher ones are mapped to the right part. The amplitude determines how many densities are we going to add to the system. The higher the amplitude a frequency channel has, the more density we will add to its corresponding column. Finally, we record the audio data from the past frame as well, and only add densities when the amplitude at one frequency channel increases for a certain amount. This way, we will only visualize the frequency channels that have an obvious change.

We are consistently adding densities to our system, however, we are assuming that the fluid in our simulation is contained in a solid box. This means the fluid cannot escape the box, resulting in a consistently increasing total density amount. In the end, each grid will have an infinite number of density. To address this problem, we removed a portion of the density from the grid at each frame to make the simulation more stable.

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[Stam 2003] 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 music analyzer based on the density field outputs satisfying the Navier-Stokes equation as well as a scheme to allow users to differentiate between frequencies. Our rendering was performed in Three.Js. We begin with instantiating the scene using a perspective camera looking in the -z direction. Then we turn to rendering the geometry on the x,y plane at z = 0.

2.3.1 Core Loop. Our approach is to render based upon the densities given by Navier-Stokes. We instantiate this grid such that each cell is a single vertex with a starting color of black 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.

In each timestep we update the frequencies from the music/microphone and use those frequencies to 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 number of particles is used to determine the final color of each cell.

The choice of timestep size is a tradeoff between responsiveness and computation. We chose a timestep size that offered reasonable fluidity while not being too taxing on the computers used for testing. Though we did not exploit this we believe there is room to optimize the timestep size based on the hardware used for viewing.

2.3.2 Choosing Colors. To provide users with a strong understanding of the state of each frequency we have a strict mapping between color and frequency in our

rendering. To provide this we decided to linearly interpolate the spectrum of color across the x axis of our rendering. Each column in the cell is assigned a base color based on its position. This ultimately requires assigning RGB values to each column such that there are no clear discontinuities in color to an end user.

We start by assigning each column a wavelength as in the visual spectrum between 460nm and 650nm. The first column is assigned to 460nm and the last is assigned to 650nm. All other columns are linearly interpolated to some value. The next step is to generate a base color for each column based on their assigned wavelength. To perform this translation we adopt an analytic approximation [Wyman et al. 2013]. Finally, we translate these values into sRGB values as outlined by the original proposal [Anderson et al. 1996].

With our base colors set we can determine the final color for each position. Our goal was for each cell to have the potential to reach the base color of its column depending on the number of particles. We chose an additive approach where each cell starts black and each particle contributes a fraction of the RGB values. We ensure the color does not pass the intended color by capping the contribution of particles. Once we get the color we update the color array in the geometry and request an update from the geometry.

2.4 Problems encountered and Lessons Learned

Since we divided up our work in each phase to allow all members contributed at the same time, we experienced lots of challenges when integrating them together. One specific example was after the first stage of our project, creating the basic fluid simulation. In isolation, the color assignment algorithm and the fluid simulation functioned as intended but when combined led to a bug creating discontinuities in the color of the simulation. Going forward, we know it is useful to have sanity checks as we progress within our individual stages and to cleanly define the interfaces between our pieces.

We also spent a lot time trying to figure out how each fluid algorithm worked. We once had an error that we just kept getting unexpected results that we wanted, we thought this error was related to the algorithm. Thus, we tried to modify diffusion and advection functions to find where the error was. However, in the end, the error was in fact due to a JavaScript return type error that has not been reported by Chrome inspection. Following this experience, we learned that having a more thorough understanding on the equation and algorithms would help us narrow down the scope for searching errors, eventually saving us time in debugging.

Another issue was with our project planning. We set out to create a fluid-based simulator with several parameters but failed to understand exactly what our novelty would be. This led to difficulties after submitting the milestone where we struggled to figure out the end goal of our project. Thankfully, we found some work related to music and found a gap where we could contribute. In the future, we will do a better job of understanding the novelty of our ideas before implementation.

3 RESULTS

Our initial version of our fluid-based analyzer 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 progress after implementing these features.

Our final product displays the effects of adding our frequency processing (for both music and microphone) as described in section 2.1 as well as our color assignment method as described in section 2.3.2. Figure 3 displays the final result after translating music into frequencies and shows the difference in color assignment for different columns as well as different cells within a column.

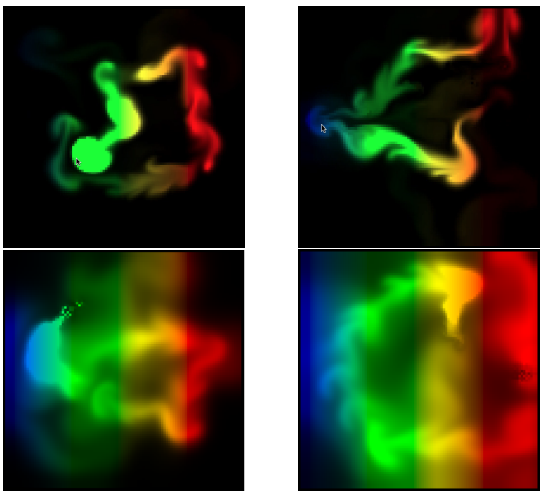

Fig. 2. Fluid simulation with mouse

4 FUTURE WORK

Due to the time limitation of this project, there are several potential improvements we did not get to implement. Currently, our system is running on the CPU. If we can convert our simulation codes to shaders and run the simulation on GPU, our simulation will become more efficient and we would be able to use a larger grid while keeping the simulation in real time. Another extension we can have is making the simulation 3D instead of 2D. It will not be too hard to convert the simulation itself from 2D to 3D, but we would need to come up with a clever way to allow users to interact with the 3D scene. Finally, we can add a GUI control panel on the website to allow users to modify different parameters including diffusion, viscosity and timestamp.

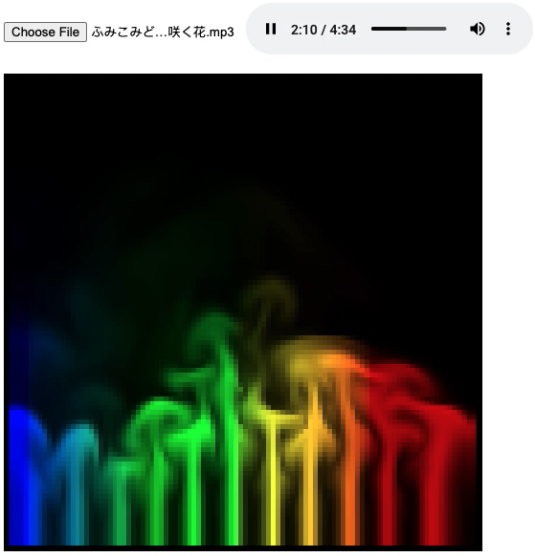

Fig. 3. Music Visualization

5 CONTRIBUTIONS

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.

To use our fluid simulation for a music visualizer, we had to modify the input and rendering part. In this phase, Yinan implemented a new feature to allow users to upload audio files and get the frequency data from it. Weiyan worked on getting input from the microphone instead. Sophie managed to take the frequency data and converted them to density/velocity input to the fluid simulation. Lloyd was responsible for adding different colors to our rendering. Same as last phase, all members have contributed to the integration and debugging of the project. The team has also spent much time tweaking the simulator and parameters to improve the visual quality of the music visualization.

The whole team worked together to deliver the paper and presentation. Lloyd was specifically in charge of the mini-demo.

REFERENCES

Matthew Anderson, Ricardo Motta, Srinivasan Chandrasekar, and Michael Stokes. 1996. Proposal for a standard default color space for the internet—srgb. In *Color and imaging conference*, Vol. 1996. Society for Imaging Science and Technology, 238–245.

Gyula Bodonyi. [n.d.]. Music visualization with fluid simulation. http://gyulabodonyi.com/fluid-based-music-visualisations/

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

Chris Wyman, Peter-Pike Sloan, and Peter Shirley. 2013. Simple Analytic Approximations to the CIE XYZ Color Matching Functions. *Journal of Computer Graphics Techniques (JCGT)* (12 July 2013). http://jcgt.org/published/0002/02/01/