CS 284/184 Final Project: Fire Simulation

Mitchell Ding
University of California, Berkeley
Berkeley, CA, USA
riceb@berkeley.edu

Jen Nguyen
University of California, Berkeley
Berkeley, CA, USA
j3nguyen01@berkeley.edu

Hoang To
University of California, Berkeley
Berkeley, CA, USA
hoangto@berkeley.edu

Yihang Zeng
University of California, Berkeley
Berkeley, CA, USA
yhz3ng@berkeley.edu

ABSTRACT

In this project, we implement a 2D candle fire simulation using Three.js and WebGL. We base our implementation on a 2D smoke simulation [4] using Navier-Stokes equation of fluid dynamics. The basis of the simulation is an Eulerian grid which we use to keep track of relevant forces and a series of fragment shaders, namely advect, divergence, jacobi, external density, external velocity, external temperature, buoyance, and vorticity, to manipulate these forces and visualize the resulting candle fire. We also modify a candle object in Blender and brought it into the screen to complete the scene.

CCS CONCEPTS

• **Computing methodologies → Physical simulation**.

KEYWORDS

Fire Simulation, Position Based Dynamics, particle system, Fluid Dynamics, Physics-based Modeling

ACM Reference Format:
Mitchell Ding, Jen Nguyen, Hoang To, and Yihang Zeng. 2023. CS 284/184 Final Project: Fire Simulation. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Fire simulation is vastly different from rendering solid objects due to its irregular behavior and sensitive response to air movement. Thus, a separate physics engine needs to be implemented along with appropriate lighting for a fire to appear realistic. There are a lot of popular and existing means of stimulating fire. In this project, we create a candle fire simulation based on the Navier Stokes equation[7] for fluid dynamics and shaders implemented with webGL through Threejs library. The inspiration comes from the realistic smoke simulation [4] and utilizing its flexible potential with colors and external densities. We then proceed to implement similar physics and tweak the parameters and colors to introduce a fire-like behavior instead of smoke. We find simulating a candle fire particularly interesting due to its special oval shape with a pointy top. The exploration of utilizing the smoke simulation engine to mimic fire is especially interesting because of how different but also similar their behaviors are.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 SETUP THE ENVIRONMENT

We decide to use Three.js since Three.js is an open-source javascript library that offers methods for interfacing WebGL core. A well-explained documentation can be found here[1]. We also want to add a candle to the website. As such, we search for a 3D asset of the candle and modify the object in Blender, which was an important step in debugging an issue we ran into.

3 TECHNICAL APPROACH

3.1 Navier-Stokes

Our fire simulation is largely based on the Navier-Stokes equations of fluid dynamics, assuming an incompressible, homogeneous fluid [2].

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u - \frac{1}{\rho}\nabla p + \nu\nabla^2 u + F \tag{1}$$

In this equation, u is the velocity field, ρ is density, p is pressure, ν is viscosity, and F represents any additional forces. Viscosity is virtually zero for fire, so we left that respective shader out of our implementation. The first term represents the advection, which is the velocity of a fluid that causes the fluid to transport objects, densities, and other quantities along with the flow. The term inside the parenthesis is the divergence, which represents the rate at which density exits the region. The second term simulates how pressure gathers and generates force and provides acceleration to the surrounding molecules.

We extract our shaders from this equation by solving for each individual term. Advection represents the first term. Divergence, the Jacobi iterations, and gradient subtraction represent the second term. Finally, the remaining shaders can be grouped into the F term. (Note that user-input external forces operate slightly differently from other external forces). Each of these terms is treated as a shader in its individual javascript file and is applied one by one when rendering the scene. The following sections will explain each individual term and how they are calculated. Figure 1 shows how everything ties together in our fire simulation. The algorithm is

presented in Jos Stam's paper on Real-Time Fluid Dynamics in Games[6].

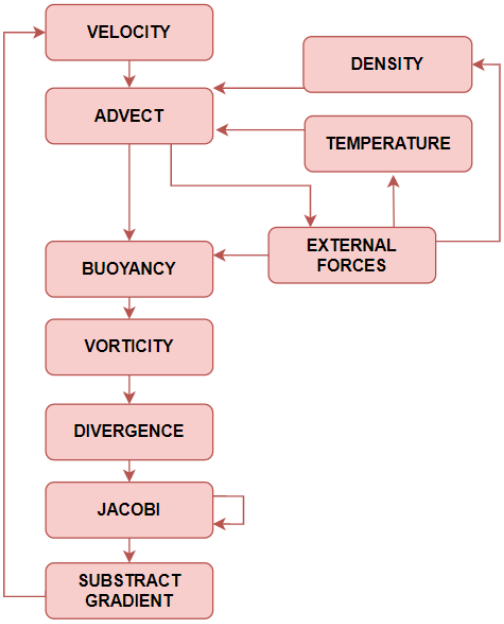

Figure 1: Diagram of the interactions of the Navier–Stokes equation

3.2 Advection

Advection handles the dissipation of the fire. This is modeled by the position of each particle in the advection field and velocity field. The dissipation approximation is computed at each timestep 1.0, using the following update rule.

$$q(x, t + \delta t) = q(x - u(x, t)\delta t, t) \tag{2}$$

It simply uses the velocity of the particle to update the positions and get the advection. The q function entails the advection field and can be set by the parameters to control the dissipation speed of temperature, velocity, and density. For our particular purposes, we set the dissipation to a relatively higher setting compared to regular smoke, because smoke persists in the air while fire needs to be sustained, and does not spread through the air like how smoke behaves.

3.3 External Forces

In this simulation, the user input is considered as the external forces, which are processed by these shaders to modify the three respective slabs. For external density and external velocity, a fixed value is incorporated into the fields from the beginning, whereas for external temperature, we plan to let the user regulate the amount added by clicking. By having distinct shaders for each slab, we can generate more personalized visual representations.

3.4 Buoyance

The buoyancy shader exerts a force in the vertical direction, and this is where the temperature slab becomes relevant. The equation for buoyancy is expressed as:

$$f_{buoyancy} = (-\kappa_d + \sigma(T - T_0))\hat{j} \tag{3}$$

For a realistic smoke simulation, if the temperature of the smoke rises, the buoyant force also increases because heat tends to ascend. Conversely, when the density of the fire rises, it becomes more massive and the buoyant force relatively decreases, making it sink. Note that these are user-adjustable values. For our particular purpose of fire simulation, a fire should not sink at all. Therefore, we tune the κ value to 0.0, so that there is no negative component on the vertical direction. On the other hand, we keep $\sigma = 0.8$, and thus, when the user adjusts the temperature to a higher setting, buoyancy increases, and we observe that the fire rises higher as well.

We demonstrate the differences in Figure 2 and Figure 3. We can see that by lowering the temperature, the flame becomes shorter.

Figure 2: Candle fire, Temperature = 4 (default)

Figure 3: Candle fire, Temperature = 1

3.5 Vorticity

Vorticity is a shader that gives the fire simulation the curling behavior by restoring the dampened external forces. Vorticity is modeled as a field with the following equations:

$$\Psi = \frac{n}{|n|}, n = \nabla|\omega|, \omega = \nabla \times u \tag{4}$$

Then we can calculate the restorative force for each particle:

$$f = \epsilon(\Psi \times \omega)\delta x \tag{5}$$

Vorticity is also a user-adjustable field. Increasing vorticity enhances the curling behavior of the fire. We demonstrate the differences in Figure 4 and Figure 5. It is relatively harder to see from a still image, but the outer layer of the fire curls more when vorticity is increased from 0.2 to 0.5.

Figure 4: Candle fire, vorticity (curl) = 0.2 (default)

Figure 5: Candle fire, vorticity (curl) = 0.5

3.6 Pressure

The calculation of pressure involves three steps, and hence, three shaders: divergence, Jacobi iteration, and gradient subtraction. The divergence shader calculates a temporary surface, while the Jacobi iteration shader calculates the actual pressure values. In gradient subtraction, the pressure gradient computed by Jacobi iteration is subtracted from the velocity field, thereby updating it. To solve the Poisson pressure equation, we employ Jacobi iteration with an initial guess of 0. The equation for Jacobi iteration is expressed as:

$$x_{ij}^{k+1} = \frac{x_{i-1,j}^{(k)} + x_{i+1,j}^{(k)} + x_{i,j-1}^{(k)} + x_{i,j+1}^{(k)} + \alpha b_{i,j}}{\beta} \tag{6}$$

Here, x denotes pressure, β represents $\nabla \cdot \omega$, α is -1.0, and β is 4.0. We determined that performing 20 to 40 iterations provided a balance between realism and performance. Once the pressure field is computed, we complete the projection step by subtracting its gradient from the velocity field:

$$u = \omega - \nabla\rho \tag{7}$$

4 ACCOMPLISHMENTS

4.1 Milestone

Referencing the Smoke Simulator [4], our first milestone was being able to create a single flame that resembles a candle fire. We utilized Three.js and 2DWebGL to build the physics engine and the renderer. We spent time trying to create a foundation for us to work off of in order to tweak the parameters, producing a more realistic fire. The fire was set in position in the middle of the screen, and there was no user input parameters.

4.2 Final Accomplishments

Building on top of our milestone, we improved the looks of the candle fire, making its shape more realistic (more oval shape). We also added user input parameters such as temperature, vorticity, radius, and pressure, using GUI. We also allowed the candle to be moved horizontally and vertically, given a fixed camera point. Lastly, we rendered a candle object using an MTL loader to complete the scene. The end result was a prettier-looking candle being lit on fire, with parameters to control its look and size.

5 RESULTS AND REFLECTION

5.1 Fire Results

In Figure 6 we present again a demonstration of our default candle fire. In our default setting, pressure = 40, temperature = 4, vorticity = 0.2, and radius = 20.

5.2 Problems Encountered

Our main problem came from limited means of debugging the shaders. We have setup the html file that hosts the rendering, and all the math was done through this html file that was linked to the javascript containing the shader logic. Thus, we could only see if the fire was rendered on screen and how it looked. When the code would break due to some unknown reasons, causing the fire to not be rendered, we would have limited knowledge or error messages

Figure 6: Candle with 2D simulated fire

that could help us on what went wrong. There is not a clear means for doing so, and getting them to work properly required a lot of trial and error on our part. This set us back quite a bit, though at the end we worked through it and made the renderer work.

Another problem we had was trying to add shaders that mimic smoke and ember coming out of our fire. However, we realized given how the shaders work, we cannot "layer" the shading and would need to implement an additional algorithm to determine the value of a texel with smoke and/or ember effects on top of fire. This increases our intended scope for the project significantly. Combined with how we have limited familiarity with Threejs and debugging, we resort to making the candle fire more perfect, along with loading a foreign candle object to complete the scene.

5.3 Lessons Learned

Our understanding of rendering fluids, smoke, and any other moving object significantly increased through this project. Working through the mathematics behind fluid dynamics and simulation has made us realize how closely related computer graphics is with physics – any simulation of the real world requires physics knowledge in that particular field, either it being fluid dynamics, thermodynamics, or radiometry. We had a lot of fun dissecting the physics behind our animated fire movement and realized how applicable our implementation is to other animated materials (such as water or smoke). For some of our group members, we were able to become more proficient in JavaScript and became acquainted with WebGL and the Three.js library through various resources[3, 5]. We have also learned how to better collaborate and divide up work efficiently so everyone could utilize their strength. This project has made all of us more confident and more comfortable working in a teamwork environment, and our fascination of computer graphics has grown so much.

5.4 Contributions

Each of our group members played an essential role in creating this fire product.

Mitchell contributed to implementing the smoke simulation pipeline and parameters tuning, and wrote the bulk of this paper, especially the technical approach and results.

Hoang helped implement the smoke simulation pipeline as well. Additionally, he wrote the object-loading code, as well as contributed to the bulk of this paper.

Jen handled some of the coding for this project, namely rendering the object and the fire at the same time. Jen also worked on tweaking the parameters for the fire, and looked into other methods to use for generating the fire initially.

Yihang tackled coding the smoke simulation and was very helpful during discussions and paper finalization.

REFERENCES

[1] [n. d.]. *three.js Documentation.* Retrieved May 02, 2023 from https://threejs.org/docs/.
[2] Mark J Harris and GPU Gems. 2004. *Chapter 38: Fast Fluid Dynamics Simulation on the GPU.* Addison-Wesley Professional. 637–665 pages.
[3] Francesco Pegoraro Luca Angioloni. [n. d.]. *Smoke Flame particles Systems.* Retrieved May 02, 2023 from https://github.com/LucaAngioloni/SmokeGL.
[4] Kenneth Tsai Rachel Bhadra, Jonathan Ngan. 2018. *Smoke Simulator.* Retrieved April 23, 2023 from https://rachelbhadra.github.io/smoke_simulator/index.html.
[5] Omar Shehata. [n. d.]. *How to Write a Smoke Shader.* Retrieved May 02, 2023 from https://gamedevelopment.tutsplus.com/tutorials/how-to-write-a-smoke-shader--cms-25587.
[6] Jos Stam. 2003. Real-Time Fluid Dynamics for Games. (05 2003).
[7] Wikipedia. 2023. *Navier–Stokes equations.* Retrieved from https://en.wikipedia.org/wiki/Navier-Stokes_equations.