2D Fire Particles Simulation Using WebGL and Smoothed Particle Hydrodynamics

Abstract

This paper presents a project that simulates fire particles using WebGL and Smoothed Particle Hydrodynamics (SPH). The method suits fire, flames, and similar gas fuels. This project shows a possible way to simulate fire using a set of controls to modify various parameters of the fire particles, such as the smoothing radius, mass, gas constant, viscosity, gravity, etc. The fire particles mainly rise because of the buoyancy. I used the interactive environment for users to understand the effects of each parameter change on the SPH system. I also added more control parameters to make the fire simulation more vivid. The evaluation approach is based on how realistic the fire is and how each parameter does affect the fire simulation. The final result shows that the implementing method is feasible but needs more improvement.

Keywords

fire, flames, animation, smoothed-particle hydrodynamics, computational fluid dynamics, WebGL, particles

1. Introduction

Modeling natural phenomena such as fire and flames is still a challenging problem in computer animations. Simulations of fluid behavior demand special effects to achieve realistic results. In recent years, the ability to simulate fires realistically has become more important in computer graphics, animation, and gaming industries. Fire simulations are also engaging for virtual reality effects, for example, to help train firefighters or to determine the proper placement of exit signs in smoke-filled rooms. The the interested reader is referred to [Rus94].

Creating a convincing fire simulation is a challenging task due to the various factors that contribute to the appearance and behavior of fire, such as temperature, fuel sources, and surrounding environmental conditions [NFJ02]. Moreover, the chaotic nature of fire and the rapidly changing structure let the computational calculations in animation reproduce accurately.

In this paper, I present a real-time fire particle simulation system that uses Smoothed Particle Hydrodynamics (SPH) to generate realistic fire animations. The SPH system is a mesh-free, particle-based method for simulating fluid dynamics, which has been widely used in various applications, such as astrophysics as the original purpose [Mon92] and computer graphics. The flexibility and robustness of SPH make it an ideal choice for simulating complex nature fluid dynamics. However, based on this property, it would make it more appropriate to simulate fluid things like water and any other liquids. Therefore, unlike normal liquid dynamics, implementing the SPH on fire particles would be a big challenge.

The system uses an HTML and JavaScript-based framework. It enables seamless integration with web applications that can run on various platforms, including computers, and mobile devices, as long as the device has a web browser. The project's primary goal is to create an efficient, basic, and hopefully visually appealing simulation. The other purpose of this project can be extended to education, web page video game design, and some other aspects.

One of the critical challenges in fire simulation is the intricate structure and rapidly changing nature of flames. To solve this issue, the system uses a particle-based approach that models the fire sparks as a collection of individual particles that can react with each other. Using this approach, I can create a high level of flexibility in controlling the appearance and behavior of the fire. Moreover, the SPH method can accurately model the fire with temperature, buoyancy, and turbulence. However, tons of calculations will be involved, and I was trying to provide an efficient and straightforward method to do the math.

I used the general implementation method based on the density, pressure, and force calculation, including gravity, viscosity, and buoyancy forces. And the particles will emit due to the lifetime changes based on the time step. In addition, the simulation optimizes vertex and fragment shaders to fit the system. The detailed implementations of the algorithms and calculations are discussed below in the overview.

2. Related Work

Fire simulation has always been an active research topic in computer graphics for a long time. As a result, many methods exist to model and render the fire phenomena. Since this is a particle-based SPH method, we will mainly focus on these parts.

The particle system has been popular for simulating various nat-

ural phenomena, including fire. Early particle-based fire simulation techniques often relied on simple, ad hoc rules to govern the behavior of fire particles. These rules included mechanisms for particle birth, motion, and death, as well as color and opacity changes [Sim90]. Although the early methods were computationally efficient, they lacked physical accuracy and only showed a visualization of a fire with some particle-based model. However, a significant advance in particle-based fire simulation was made by Foster and Metaxas [FM97], who introduced a physically-based model for simulating the motion of hot, turbulent gases. This approach extends the particle system to model the fluid flow and a separate set of particles to represent the fire's visible structure. Next, Stam [Sta97] extended this work by incorporating a stochastic dynamics model to simulate the effects of turbulence on flexible structures, such as flames. Then the huge development in particle-based fire simulation was the work of Nguyen et al. [NFJ02], who presented a physically-based model for simulating and rendering fire. The method combined a Lagrangian particle system for fluid dynamics with a novel technique for modeling the emissive properties of fire particles.

Regarding particle-based models, smoothed particle hydrodynamics play a significant role in simulating various fluid phenomena in computer graphics, including water, smoke, and fire. One of the critical advantages of SPH is its ability to deal with fluid-fluid and fluid-solid interactions. This makes the SPH suitable for simulating complex fluid dynamics like fire. However, Stam and Fiume [SF95] have been the first to apply SPH to simulate gas and fire phenomena. However, physically-based methods can produce a photo-realistic effect; most are based on solving Navier-Stokes Equation, whose computation cost is relatively high [WYDZ13]. For example, Muller et al. [MCG03] presented an interactive method based on SPH to render fire. Although there's not much directly related to fire simulation systems using SPH, Ihmsen et al. [IOS*14] provided some detailed approaches to implementing an SPH with fluid simulations in computer graphics. They employed SPH to develop a real-time fluid simulation system that focused on the efficiency and scalability of the simulation. The approach uses a predictive-corrective incompressible SPH (PCISPH) method that allows for adaptive time-stepping and efficient boundary handling. That method makes producing high-quality fluid animations in real-time possible. However, I tried to combine the advantages of SPH and the regular particle-based to build an efficient way of fire simulation with less computation cost.

3. Overview

In this project, I have implemented a fire particle simulation system using JavaScript and HTML5 Canvas with the help of WebGL. The simulation aims to create a visually appealing and basic realistic representation of fire particles by considering physical features, including movement, color, and other physical properties. The method is a basic model for implementing fire simulation with smoothed particle hydrodynamics system. The model is divided into two main parts, the base-particle system, and the SPH method.

3.1. Base-Particle System

First, we have developed a particle system consisting of individual particles, each representing small sparks of the fire. Then, the particles will go through creating, updating (in which SPH mainly participates), and rendering on the canvas to simulate the behavior of the fire naturally.

3.1.1. Particle Properties

The fire particle is initiated with several properties, including position, velocity, acceleration, color, and lifetime. These properties are the controlling parameters of the specific status of each particle in the system. These values are updated at each time step.

The position is represented using a 2D coordinate (x,y). The initial position is set to be the base fire source position at the middle bottom of the canvas. The velocity is a 2D vector (vx, vy) representing the particle's speed and direction. The initial velocity is set to be randomly moving within a specific range. The acceleration is another 2D vector (ax, ay) that influences the particles' velocity. In this model implemented, the acceleration is primarily affected by gravity and buoyancy. The color parameter is implemented easily. It is determined by a gradient that transitions from a bright color near the particle's center to a darker color outside the particle. Usually, the fire color would change due to the temperature. The higher the temperature, the lighter to color will be, as shown in figure 1. Each particle has a predetermined lifetime randomly generated within a specified range. The lifetime determines how long the particle will be visible in the simulation before it vanishes.

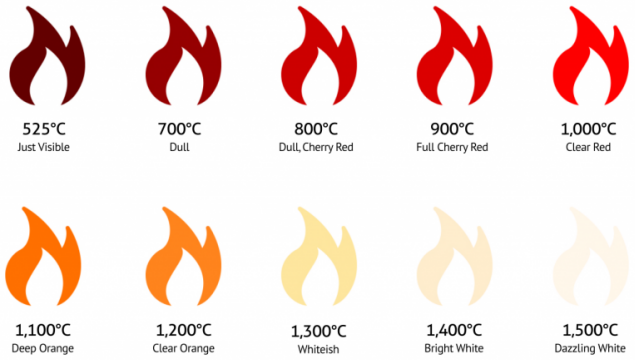

Figure 1: *Relationship between Fire Color and Temperature [Prond]*

3.1.2. Particle Update

The particles will be updated based on the properties and the physics properties on each time step. The update process includes updating the position, velocity, lifetime, and temperature.

To update position, first update the velocity using the acceleration vector:

$$\mathbf{v}t + 1 = \mathbf{v}t + \mathbf{a}_t \Delta t$$

where $\mathbf{v}t$ is the velocity vector at time t, $\mathbf{a}t$ is the acceleration vector at time t, and Δt is the time step. The velocity variations

make the particle go faster or slower due to the effects of gravity, buoyancy, etc. Then the position is updated using the updated velocity vector:

$$\mathbf{p}_{t+1} = \mathbf{p}_t + \mathbf{v}_{t+1}\Delta t$$

where $\mathbf{p}_t$ is the position vector at time t, making the particles move and simulate the realistic vision of fire sparks. Then the lifetime will be updated with the time step. When the lifetime goes below zero, the particle will be removed from the array and thus disappeared.

3.1.3. Render

In the render process, the vertex and fragment shaders were used. The vertex shader calculates the normalized position of the particle dividing the input vertex position by the screen resolution and mapping it to the range of -1 to 1. As for the fragment shader, it is responsible for determining the color of each rendered particle based on the temperature. First, it calculates the distance between the current fragment coordinate and the center of the particle and assigns the intensity of the particles based on the distance. Then the temperature is calculated by multiplying the intensity by the maximum temperature to determine the particle's color eventually.

3.2. Smoothed-Particle Hydrodynamics Implementation

In this fire particle simulation, we use an adapted version of SPH to model the fluid-like behavior of fire. The main significant concept in SPH is using the kernel function to approximate continuous fields, such as density and pressure. Moreover, the governing equation of calculating the forces is also based on this critical part.

The SPH method implies a collection of discrete particles, each with its mass, position, and velocity. Then the density field of the fluid is reconstructed by weighting the mass contribution of neighboring particles within a certain radius using a kernel function. The kernel function, $W(r,h)$, depends on the distance r between two particles and the smoothing length h. A widely used kernel function is the cubic spline kernel:

$$W(r,h) = \frac{1}{\pi h^3}\begin{cases} 1-\frac{3}{2}\left(\frac{r}{h}\right)^2+\frac{3}{4}\left(\frac{r}{h}\right)^3, & 0 \le \frac{r}{h} < 1 \\ \frac{1}{4}\left(2-\frac{r}{h}\right)^3, & 1 \le \frac{r}{h} < 2 \\ 0, & \frac{r}{h} \ge 2 \end{cases} \tag{1}$$

3.2.1. Density and Pressure

With the help of the kernel function, the density at a particle's position can be then computed as [Mon92]:

$$\rho_i = \sum_{j=1}^{N} m_j W\left(\left|r_i - r_j\right|, h\right) \tag{2}$$

where ρ_i is the density at the position of particle i, N is the number of neighboring particles, and m_j is the mass of particle j.

After that, We consider the pressure and viscosity forces to model the system. The pressure force on the particle i can be calculated using the following equation [Mon92]:

$$F_{p,i} = -m_i \sum_{j=1}^{N} m_j \frac{p_i + p_j}{2\rho_j} \nabla W\left(\left|r_i - r_j\right|, h\right) \tag{3}$$

where p_i and p_j are the pressure at the positions of particles i and j, respectively. In addition, ∇W is the gradient of the kernel function.

3.2.2. Viscosity Force

As for the viscosity forces, it can be calculated on particle i with [Mon92]:

$$F_{v,i} = \mu \sum_{j=1}^{N} m_j \frac{v_j - v_i}{\rho_j} \Delta W\left(\left|r_i - r_j\right|, h\right) \tag{4}$$

where μ is the dynamic viscosity, v_i and v_j are the velocities of particles i and j, respectively. ΔW is the Laplacian of the kernel function.

After integrating the pressure and viscosity forces, we can now compute the acceleration of the particles using the following:

$$a_i = \frac{F_{p,i} + F_{v,i}}{m_i} \tag{5}$$

With the accelerations we have calculated, we can then update the velocities and positions using the time integration method. We use the explicit Euler method for simplified implementation [ESHD05]. The updated velocities and positions are calculated using the below formulas where Δt is the time step:

$$\begin{aligned} v_i(t+\Delta t) &= v_i(t) + a_i(t)\Delta t \\ r_i(t+\Delta t) &= r_i(t) + v_i(t)\Delta t \end{aligned} \tag{6}$$

3.2.3. Buoyancy Force

In addition to the pressure and viscosity forces, we must consider the buoyancy force resulting from the density, gravity, and the surrounding environment. The buoyancy force can be calculated with β as the buoyancy coefficient, ρ_{amb} as the ambient density, and g as the acceleration due to gravity:

$$F_{b,i} = -\beta\left(\rho_i - \rho_{amb}\right) g \tag{7}$$

The buoyancy force is then added to the total force acting on each particle and causes the influence on the acceleration, velocity, and position updates.

3.2.4. Algorithm

Figure 2 shows this system's basic SPH simulation algorithm. First, initialize the particle positions, velocities, and other properties like mass, temperature, and color. Then for each loop, do the following process.

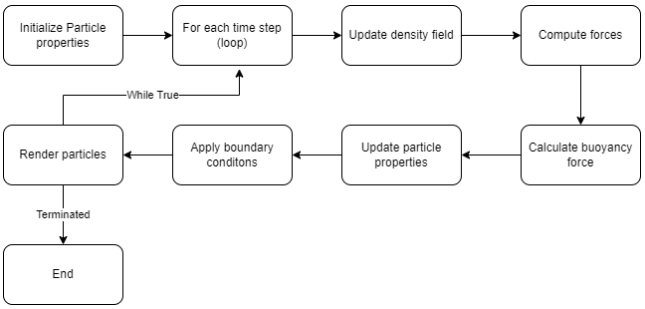

Figure 2: *SPH algorithm Loop*

1. Update the density field using the kernel function and particle masses.
2. Compute the pressure and viscosity forces acting on each particle.
3. Calculate the buoyancy force based on gravity and density.
4. Update the accelerations, velocities, and positions of the particles using the total force and a time integration model(Euler method in this case)
5. Apply the boundary conditions and constraints, such as the constraints on particle lifetimes.
6. Render the updated particle positions and properties.

The loop stops when the system terminates, or keep looping this process to keep the fire particles moving. This algorithm forms the basis of the fire particle system. By implementing the SPH method, after including the buoyancy force, we can ensure the fire particles will arise naturally to meet the natural behavior of fire particles.

4. Evaluation

The project implemented a 2D fire simulation system based on SPH, and this section will present the evaluation of the system both qualitatively and quantitatively. The results provide a thorough understanding of the system's performance, computational limitations, possible improvement areas, and so on.

4.1. Qualitative

To demonstrate the system's capabilities, we created scenarios with different settings, such as varying the number of fire particles, initial conditions, and environmental factors. All these parameters can be controlled from the slider bar beside the canvas. The qualitative evaluation focuses on the visual realism and accuracy of the simulated fire behavior.

4.1.1. Visual Realism and Fire Behavior

The visual quality of the particle system is a crucial aspect to evaluate. By analyzing the generated fire visualizations, the fire has some basic features, for example, the turbulent motion, fire emissions, and color transitions in different temperatures as in figure 3. The fire will perform as desired, and the particle will emit from the bottom of the canvas, where the fire source coordinate locates, and vanish when the lifetime is met.

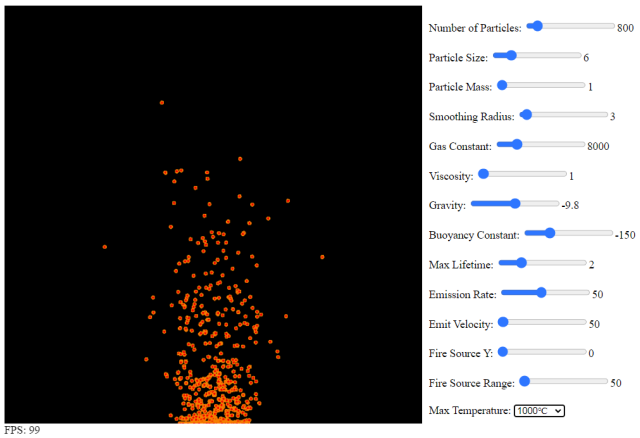

Figure 3: *The visualization of the fire particle system*

In addition to the primary visual quality, we must evaluate the fire behavior under different conditions by changing the SPH variables and buoyancy constants. The system can model the fire particles' buoyancy-driven motion and the fire's expansion and contraction due to the changes in gravity, gas constant, and particle mass. For example, the smoothing radius can control the fire particles to cluster together, as figure 4 shows. Besides, the change in the buoyancy constant can vary the expansion and contraction of the fire. With a higher buoyancy constant, the fire contracts and vice versa.

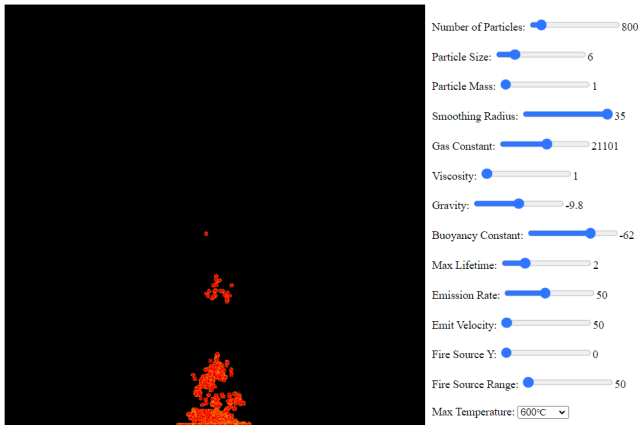

Figure 4: *The visualization of the fire particle system*

4.1.2. Edge Cases and Limitations

While the fire particle simulation system demonstrates some visual quality and fire behavior, it's also essential to identify the faults, edge cases, and issues. For example, for the general faults, the temperature should have a more significant temperature with a lighter color on the outside frame of the flame based on the common knowledge of fire. In addition, the ideal relationship between the pressure and temperature should follow the ideal gas law proposed by Clapeyron [Cla35]:

$$PV = nRT \tag{8}$$

where P is the pressure of the gas, V is the volume of the gas, n is the number of gas particles, R is the universal gas constant, and T is the temperature of the gas in Kelvin. This equation states that the product of the pressure and volume of a gas is proportional to the number of gas particles, the universal gas constant, and the temperature of the gas. Using these two relations will make each particle's physical movement more realistic. In this model implemented, I ignore these factors for simplicity. As for the edge cases, for example, the figure 5 shows when there's a tremendous gas constant value, the fire will expand extremely, which may not be an ideal behavior of a fire in real life. Similarly, in figure 6, the fire particle would expand more to the two sides when the viscosity is substantial. Furthermore, in cases with a vast number of particles, the computational demand increases significantly, which causes the performance issue, and the system will be stuck. It will be similar to other parameters in the control slider bar, so limitations are set to ensure the users will not meet those situations.

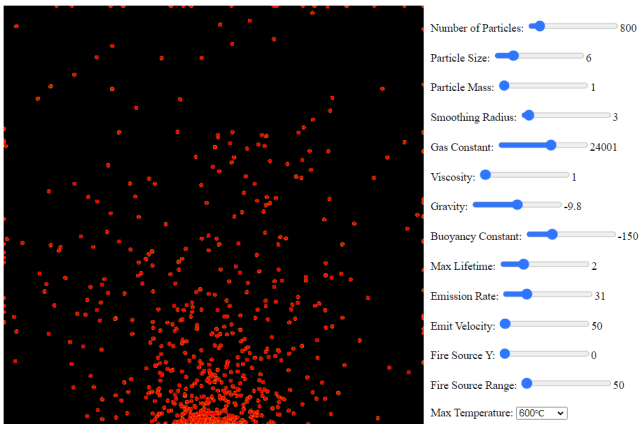

Figure 5: *Edge Case for Gas Constant*

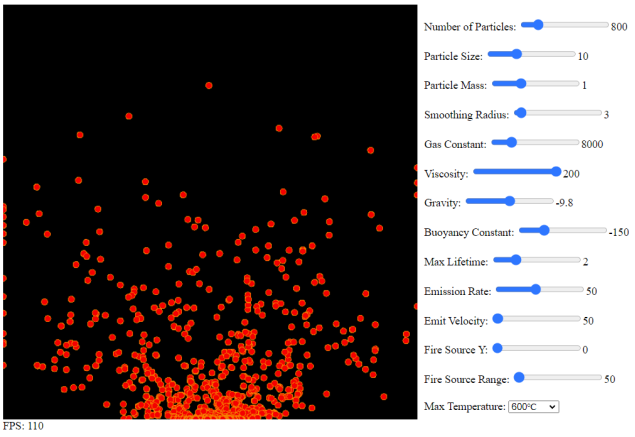

Figure 6: *Edge Case for Viscosity*

4.2. Quantitative

To comprehensively evaluate the system, we also need to perform a quantitative analysis using various metrics and scenarios. These quantitative results can help us understand and improve the performance and computational limitations of the system. In addition, since the system is written in WebGL with HTML and JavaScript, it can run on a web browser on computers or phones. The testing environment I used is my desktop computer with Windows 11 Pro system, AMD Ryzen 7 5800H with Radeon Graphics, 3201 Mhz, 8 Cores, 16 Logical Processors Processor, 32.0 GB DDR4 Memory, RTX 3060 Ti Graphics Card, and Chrome Web Browser with Version 111.0.5563.148 (Official Build) (64-bit).

To assess the quantitative performance of the system, we can parameterize the number of fire particles and evaluate the system's performance under different levels, ranging from a few hundred particles to thousands. The metrics used for evaluation include the Frames per second (FPS), which indicates the smoothness and responsive time. Another two primary metrics are memory usage related to the efficiency of resource utilization and simulation time (loading time), represented by the total time taken to complete a simulation. The primary testing tool will be the FPS display function implemented in the code. The FPS calculation used an exponential moving average (EMA) with a smoothing factor of 0.1 to help in smoothing out short-term fluctuations and provide a more stable estimate of the frame time. The memory usage and loading time parameters will be evaluated using the Chrome build Performance analysis tool in the browser development tool as illustrated in figure 7.

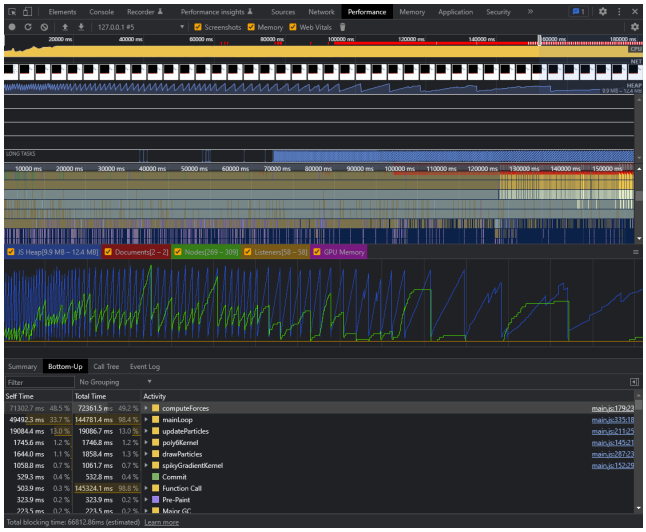

Figure 7: *Performance Analyze Tool*

When there are changes in the number of particles, the FPS will vary correspondingly. When the number of particles is too big that exceeds the desired amount, the FPS will drop dependently. For example, in figure 8, with 5000 particles generated simultaneously, the system will become very slow and not smooth. The FPS value, as seen in the figure 8, has dropped to 5, making the screen move like PowerPoint slides. The

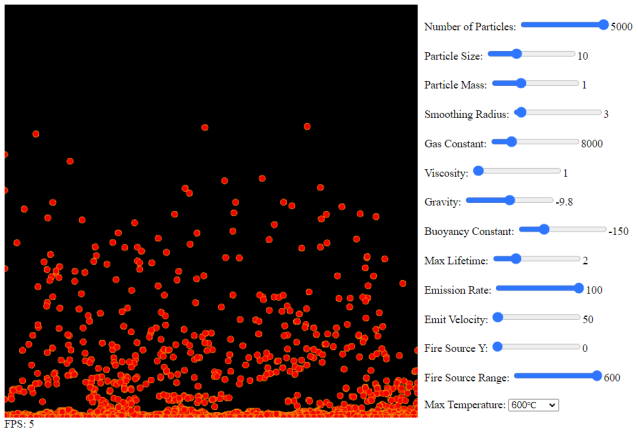

Figure 8: *The FPS with 5000 particles*

Table 1: *FPS, Memory, Loading Time changes on Number of Particles*

Number of Particles	FPS	Memory	Loading Time
300	~110	9.8 MB - 12.3 MB	5.46 ms
500	~105	10.0 MB - 12.3 MB	6.73 ms
800	~90	10.1 MB - 12.4 MB	15.97 ms
900	~75	10.1 MB- 12.4 MB	18.74 ms
1000	~60	10.1 MB - 12.3 MB	21.89 ms
1300	~30	10.1 MB - 12.4 MB	42.67 ms
1500	~25	10.0 MB - 12.3 MB	86.09 ms
2000	~15	10.1 MB - 12.3 MB	110.67 ms
3000	~7	10.0 MB - 11.9 MB	347.42 ms
5000	~5	10.0 MB - 11.7 MB	615.52 ms

After analyzing the FPS, Memory, and Loading Time value changes on the number of particles as shown in the table 1, the stats clearly state that the FPS will highly decrease when the number of particles increases. When the number of particles exceeds 1000, the FPS will drop below 60, which reaches the least frame rate human eyes can expect, and people would feel the screen get stuck. In addition, the loading time changes significantly with the number of particles. Three hundred particles will only use about 5.46 ms to run an entire main loop of generating, updating, and rendering the particles. However, 5000 particles would use 615.52 ms to maintain a whole loop, making the screen stuck and causing the FPS drops hugely.

As for the memory usage, it didn't varies with the number of particles. It stays in the same range as usual. However, after performing some similar processes as the research on the number of particles, I discovered that the memory would change with the particle size as shown in figure **??**, maximum lifetime, emission rate, and fire source range. When the particle size increases, the memory will increase and vice versa for all the factors, as shown in the figure 9.

Figure 9: *Memory Usage Profiles in Performance Analyze Tool*

Table 2: *Memory Usage on Particle Size*

Particle Size	Memory
1	6.8 MB - 9.2 MB
3	6.8 MB - 9.3 MB
5	7 MB - 10 MB
8	7.3 MB - 9.9 MB
10	8.9 MB - 11.2 MB
12	9.5 MB - 11.5 MB
15	10.0 MB - 12.3 MB
20	10.0 MB - 13.3 MB
25	11 MB - 14.5 MB
30	11.2 MB -15.7 MB

4.3. Comparison with Existing Works

To further strengthen the evaluation process, we must compare the fire particle simulation system with other fir/fluid simulation works regarding visual quality, realism, and performance. The fire particle simulation system we implemented is a well-structured animation based on the SPH-related formula and previous works. For example, our fire particle simulation system compares favorably to the creation of Müller et al. [MCG03], which employs a particle-based approach to model fire and smoke. The particle-based method shows a higher flexible and adaptable system, allowing for more changes in the fire behavior in complex geometries and under varying conditions. Furthermore, the system demonstrates efficient performance in terms of memory usage.

However, the system may not perform as well as the work of Ihmsen et al. [IOS*14], which computes complex scenes with millions of sampling points, one- and two-way coupled rigid and elastic solids, multiple phases, and additional features such as foam or

air bubbles at a reasonable expense. The method can achieve more detailed fluid simulation and much more particles in a simulation, especially in scenarios with highly complex geometry and extreme environmental influences. Nevertheless, the increased detail comes at the cost of higher computational demand, which may not be suitable for all system applications.

In summary, the evaluation demonstrates the fire simulation system is essential and visually realistic and does have some changes under various conditions. However, it is still necessary to acknowledge that some computational limitations and edge cases may affect the simulation's performance and suitability. The quantitative evaluation, combined with the qualitative analysis, provides some valuable insights into the strengths and weaknesses of the system. This process dramatically helps us identify potential areas for future improvements and optimizations.

5. Conclusion

This paper presents the development and implementation of a fire particle simulation system that effectively creates visually appealing and realistic fire animations. The project was developed using HTML and JavaScript. The reason for using WebGL is because of the capabilities of web browsers and the ease of deployment in web-based applications. After implementing the project, I explored different concepts and techniques on particle-based fire simulation systems and created a medium-quality fire particle simulation based on them. The primary approach uses Smoothed Particle Hydrodynamics (SPH) method, a grid-based spatial data structure, and particle-based modeling techniques to create a visually convincing fire particle movement simulation.

The SPH method provided a foundation for fluid-like behavior in the fire particle simulation. I used an efficient kernel function to be the base of the SPH system. In addition, calculate the pressure and viscosity so that the particles can interact smoothly and realistically, resulting in a realistic fire particle animation. To further optimize the performance of the simulation, I ignored some parts of the non-important physical features of the fire particles in the simulation to improve the computational efficiency of the approach. In addition, I implement an easy particle blending technique to leverage WebGL to create fire color and transparency effects simply. This technique allows the users to interact with fire effects with some degree of customization. The fire can then reasonably presents the color. Moreover, some of the basic SPH system parameters are used and easy to change with the slide bar to make the users more easily adjust the appearance of the fire to suit the needs and different kinds of fire simulation environments.

5.1. Challenges

Throughout the development of this project, I encountered many challenges. One of the primary challenges was optimizing the simulation. Since the SPH is a giant complex method, I need to take what I need and filter the necessary core features of simulating a fire to ensure smooth, real-time performance number of particles is balanced. I addressed this issue by keeping only the pressure, gravity, temperature, viscosity, and buoyancy parameters to reduce the computational complexity of the particle interactions. Additionally, I used an exponential moving average to calculate the frame rate of the simulation, ensuring a stable and consistent FPS display so that I could analyze and evaluate the system more efficiently and scientifically.

The second challenge is the implementation process of the SPH. Unfortunately, there is little knowledge online about implementing a fire simulation using SPH. Therefore, I need to update the general SPH method of the fluid system and fuse them with the fire properties, including temperature and buoyancy force. That means I must also vary the viscosity, pressure forces, and overall force calculations. Fortunately, the force calculations were implemented successfully.

Another challenge is implementing my own vertex and fragment shader. Since the system will differ from the regular particle-based system, I need to change those two shaders' codes to make them fit my implemented system. Then I can render the particle effects.

5.2. Limitations

While our fire particle simulation system successfully achieved the intended goals, some limitations should be acknowledged, which may affect the final visualization effects. One limitation is that the simulation is not physically accurate. The project primarily aims to combine the SPH method with particle-based fire simulation. It is more focused on creating visually appealing fire effects rather than simulating an extremely accurate physical process of combustion and heat transfer. As a result of this limitation, the simulation may not be suitable for applications that require exceptionally physically proper fire behavior.

Another limitation is the scalability of the simulation. The performance may not accept many simulation particles based on the method implemented. As the number of particles increases, the computational complexity of the system increase, which would cause the performance issues like a frozen screen. Further optimizations may be necessary to maintain real-time performance in more complex scenarios with more particles and more complicated parameter features.

Another minor limitation is based on the edge cases for the parameters. If the parameters use extreme edge scenario values, the fire simulation system may encounter fire behavior issues. For example, the animation may not be like a regular fire in normal life. However, this can also be an advantage of the system that supports a variety of complex environments.

5.3. Future Works

Based on these limitations, several ways for future work can be identified.

Firstly, incorporating physically accurate fire behavior into the simulation would be valuable. This can be achieved by integrating more sophisticated fluid dynamics models, such as the Navier-Stokes equations and the Perlin-Noise function on generating random particles. Additionally, incorporating combustion, heat transfer, and Gas Law models can improve the simulation's physical features to make it visually and physically more realistic.

Another area of future work is optimizing the system, particularly regarding scalability. This may be solved by exploring alternative spatial data structures, such as octrees or k-d trees, offering better performance for many-particle simulations. Additionally, further research into parallelization techniques, such as GPU-based computing, would significantly improve the performance since GPU is specifically designed for such graphic rendering processes.

Another possible future work is related to the 2D to 3D translation. Since 2D is less convincing and realistic than 3D modeling, I can consider involving the Three.js document that helps design 3D simulation in WebGL. Three.js is a popular and widely-used JavaScript library that simplifies creating 3D graphics for the web by providing a high-level API for interacting with WebGL [Cab10]. With Three.js, the fire particle system can be enhanced in several ways. The first and most significant way is improving the rendering. Three.js provides extensive rendering features and optimizations, such as support for physically based rendering materials, shadows, and post-processing effects. Combining the previous future work of optimization, the visual quality and realism of the fire simulation could be significantly improved. The Three.js file also offers a robust scene graph system, simplifying the managing process of complex 3D scenes and objects. This would make it easier to create more stereoscopic fire particle simulations.

In addition, the fire particles are now only particles based on the WebGL build particles. Therefore, it could be more visually realistic with high-quality textures rendered to each particle by updating the fragment shader.

There are also several other potential directions for further exploration. For example, the fire particle simulation system could include additional environmental factors, such as wind and colliding with other objects, whether in liquid, Gas, or solid form. Additionally, the system can adapt to simulate different types of phenomena, such as smoke and a combination of fire and smoke, by adjusting the parameters of the SPH method and the particle blending techniques.

In conclusion, I presented a web-based fire particle simulation system that effectively combines the SPH method and a grid-based spatial data structure to create visually appealing fire animations. Although some challenges were encountered during the implementation, the method has successfully demonstrated the effectiveness of fire particles while maintaining the real-time simulation performance in some way. However, there are still some limitations to the current implementation. Future work and research will improve those features and make the fire particle simulations more vivid.

This project illustrates the possibility of creating complex, partly physical-based simulations within a web browser. Furthermore, with the improvement in techniques of web-based simulations, the future work of animation on fluid dynamics simulation would become much easier and more visually appealing.

Ultimately, the fire particle simulation system offers the feasibility of creating reasonable fire particle emission effects using the SPH method and WebGL as supports on a web page. I hope this work can be developed with a more enhanced method, as mentioned in the future work part. As we develop the techniques for simulating the fire effects in animation, we could eventually create some creative animations and apply them to many aspects of our life.

References

[Cab10] CABELLO R.: Three.js. https://threejs.org/, 2010. Accessed April 9, 2023.

[Cla35] CLAPEYRON E.: Mémoire sur la puissance motrice de la chaleur. *Journal de l'École Polytechnique 14* (1835), 153–190.

[ESHD05] ERLEBEN K., SPORRING J., HENRIKSEN K., DOHLMANN H.: *Physics-Based Animation*. Charles River Media, 2005.

[FM97] FOSTER N., METAXAS D.: Modeling the motion of a hot, turbulent gas. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (1997), pp. 181–188.

[IOS*14] IHMSEN M., ORTHMANN J., SOLENTHALER B., KOLB A., TESCHNER M.: SPH fluids in computer graphics. *Computer Graphics Forum 33*, 1 (2014), 66–76.

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), ACM, pp. 154–159.

[Mon92] MONAGHAN J. J.: Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics 30*, 1 (1992), 543–574.

[NFJ02] NGUYEN D. Q., FEDKIW R., JENSEN H. W.: Physically based modeling and animation of fire. *ACM Transactions on Graphics (TOG) 21*, 3 (2002), 721–728.

[Prond] PROTECTION C. F.: The temperature of fire. https://www.cityfire.co.uk/news/the-temperature-of-fire/, n.d. Accessed April 1, 2023.

[Rus94] RUSHMEIER H.: Rendering participating media: Problems and solutions from application areas. In *Proceedings of the 5th Eurographics Workshop on Rendering* (1994), pp. 35–56.

[SF95] STAM J., FIUME E.: Depicting fire and other gaseous phenomena using diffusion processes. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), ACM Press, pp. 129–136.

[Sim90] SIMS K.: Particle animation and rendering using data parallel computation. In *Computer Graphics (Proceedings of SIGGRAPH 90)* (1990), vol. 24, pp. 405–413.

[Sta97] STAM J.: Stochastic dynamics: Simulating the effects of turbulence on flexible structures. *Computer Graphics Forum 16*, 3 (1997), 159–164.

[WYDZ13] WANG L., YE W., DUAN M., ZHANG Y.: Real-time rendering of flames on arbitrary deformable objects. *Science China Information Sciences 56* (2013), 1–9.