# Fluid Simulation and Screen Space Fluid Rendering In VR

## An implementation of a fluid simulation using Smooth Particle Hydrodynamics (SPH) and Screen Space Fluid Rendering (SSFR) for use in VR

JAMES TRY and YAFQA KHAN, University of Washington, USA

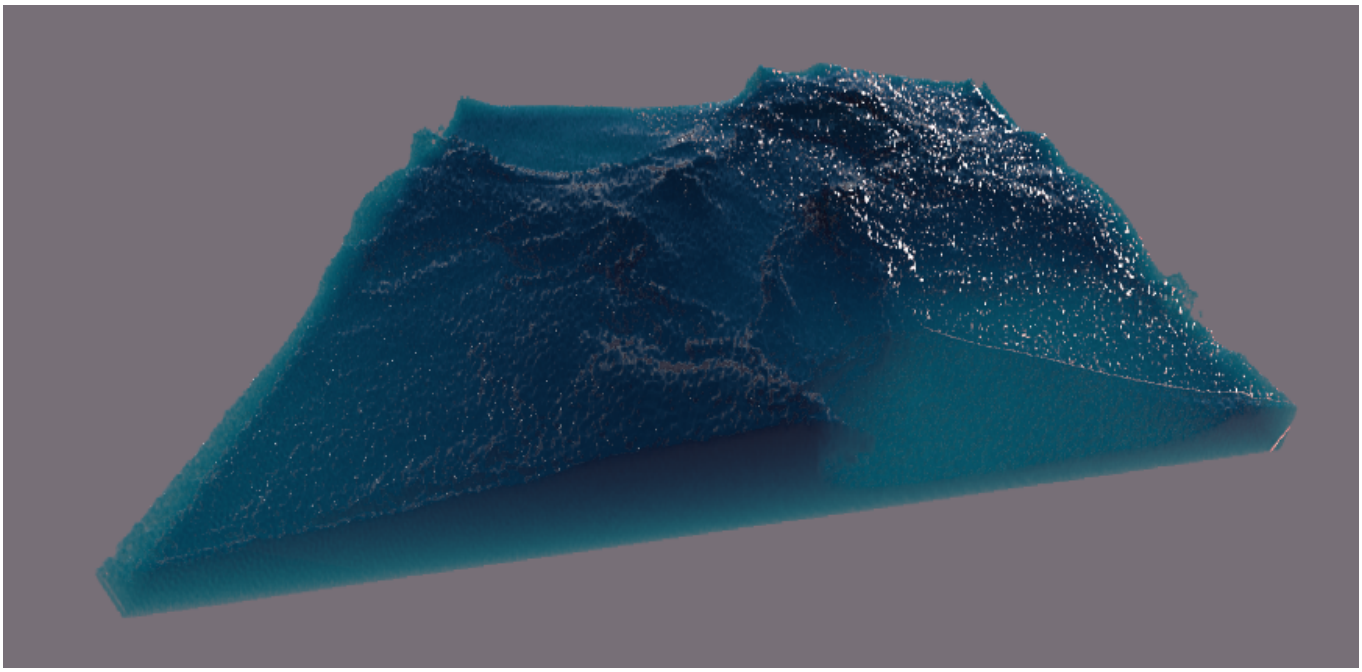

Fig. 1. A picture taken from the Meta XR simulator of a Meta Quest 3 of our simulation with 1.5 million particles. This rendered at around 35 frames per second on the simulator. There are still some weird/incorrect artifacting since it is not completely physically accurate (notice the bottom right quadrant).

We implement fluid simulation and rendering in a VR application. The fluid is composed of many small particles and simulates pressure forces, viscosity, and external forces including gravity and the user's hand motions, as given by the Navier-Stokes equations. We build on the technique "Smooth Particle Hydrodynamics" or SPH to simulate the motion of these fluid particles. To make this work efficiently in a VR application, we had to adapt our algorithms so that they could be parallelized on a GPU. We then render the particles utilizing "Screen Space Fluid Rendering" to give the particles a realistic look while not increasing the performance requirements by a large amount. We find that the use of SPH for the physics and SSFR for rendering is a potent combination that simulates fluids accurately without putting too high of a burden on hardware such as the Meta Quest 3 in certain scenarios. There are still features to be added to this fluid simulation that would make for worthwhile extensions, such as attempting to add foaming for more realism, caustic entities, further optimization utilizing BVHs for collisions.

## 1 INTRODUCTION

The motivation for our project is have an interactable fluid that we could build upon for our VR capstone project next quarter. We are planning to eventually build a VR game in which users can interact with four elements like in the cartoon "Avatar: The Last Airbender": air, water, earth, and fire. A useful stepping stone for this goal is to see how we could viably simulate a fluid in VR. There are other more scientific motivations, such as the usefulness of physically accurate fluid simulations that people are able to interact with your own "physical" hand which might allow for better research in that field.

We found a YouTube video by Sebastian Lague [Lague 2023] which built upon [Müller et al. 2003] to simulate a particle-based fluid in 3D using a technique called "Smooth Particle Hydrodynamics". This technique essentially interpolates the properties of nearby particles to estimate field quantities like density, pressure, and velocity at any point in space. These quantities are then used to the position and velocity of each particle according to the Navier-Stokes equations. We attempted to reproduce the work of [Lague 2023] in Unity and observed the quality of the simulation through Meta Quest 3. As the simulation is computationally intensive, we had to run our simulation on a GPU and adjust our implementation to create a smooth experience for the user.

Overall we found that using SPH and SSFR inside of VR seems like a viable alternative to attempting to do a full 3D flow simulation.

Authors' address: James Try, jamestry@cs.washington.edu; Yafqa Khan, yafqak@cs.washington.edu, University of Washington, USA.

This is mainly useful for things such as real time fluid simulations for games, since we are not looking to have an entirely accurate fluid flow, just one that looks the best. In other environments where accurate fluid flow, our rendering technique will not apply, since we most likely do not want to be rendering any fluids using SSFR as it forgoes accuracy for performance, but the base idea of fluid flow using SPH in VR is possibly a useful stepping stone in other areas.

### 1.1 Contributions

- Utilized Smooth Particle Hydrodynamics (SPH) to simulate a basic fluid simulation.
- Implemented Screen Space Fluid Rendering (SSFR).
- Translated a (mostly) working version of the above two to render in VR.

## 2 RELATED WORK

In 1822 and 1845 Claude Navier and George Stokes formulated the Navier-Stokes equations that describe fluid dynamics. These equations encode conservation of mass, momentum, and energy. These equations can be numerically solved by a computer, opening the field of computational fluid dynamics. We build on the work of [Müller et al. 2003], which is a particle-based approach to fluid dynamics. They use Smooth Particle Hydrodynamics, which is a technique initially proposed for modeling gasses, to estimate the quantities of force fields necessary to simulate the Navier-Stokes equations. Using SPH has the benefit of being relatively simple to implement, while still being able to model the general behavior of fluids. The downside is that a large amount of compute is required to scale up the number of particles, limiting the realism of the fluid. We also found a YouTube video tutorial [Lague 2023] that goes through this exact paper, which we followed along to help build our VR fluid simulator.

## 3 METHOD

We model a fluid as a finite set of particles that are constrained within a rectangular boundary, which automatically guarantees conservation of mass. We apply conservation of momentum to this setting, which is formulated by the Navier-Stokes equation as

$$\mathbf{f} = \rho \frac{D\mathbf{v}}{dt} = -\nabla p + \rho \mathbf{g} + \mu \nabla^2 \mathbf{v}, \tag{1}$$

where $\mathbf{f}$ is the force density field, $\rho$ is the field density, $p$ is the field pressure, $\mathbf{g}$ is the external force density field (including gravity and wall collisions), $\mu$ is the viscosity of the fluid, and $\nabla^2 \mathbf{v}$ is the Laplacian of the velocity field. Thus, we can calculate the acceleration of a particle $i$ as

$$\mathbf{a}_i = \frac{D\mathbf{v_i}}{dt} = \frac{\mathbf{f_i}}{\rho_i}, \tag{2}$$

where $\mathbf{v_i}$ is the velocity of particle $i$ and $\mathbf{f}_i$ and $\rho_i$ are the force density field and density field evaluated at the location of particle $i$, respectively. We estimate the density field and each term of the force desnisty field using a technique called Smooth Particle Hydrodynamics, or SPH, which we describe now.

According to SPH, any scalar quantity $A$ can be estimated at a point $\mathbf{r}$ by interpolating the value of $A$ over every particle within a radius $h$. This is encapsulated in the "SPH equation":

$$A(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h), \tag{3}$$

where $m_j$ and $\rho_j$ is the mass and density of the $j^{th}$ particle, and $A_j$ is the scalar quantity for the $j^{\text{th}}$ particle. $W(\mathbf{r}, h)$ is a smoothing kernel used to interpolate the scalar quantity; the notation $W(\mathbf{r}, h)$ means that the smoothing kernel has support $h$ and is being evaluated at a point $\mathbf{r}$. The kernel must integrate to 1 over its support:

$$\int W(\mathbf{r}, h) dr = 1. \tag{4}$$

There are several options for the smooting kernel. In practice, we use a smoothing kernel of $W(\mathbf{r}, h) = \frac{15}{\pi h^6} \max\{0, (h - |\mathbf{r}|)^3\}$, which has the property that the gradient of $W$ does not "vanish" to zero as $|\mathbf{r}|$ approaches $\mathbf{0}$, which helps prevent particles from clumping together.

Applying the SPH equation to density, we can model the density field at any point $\mathbf{r}$ as

$$\rho(\mathbf{r}) = \sum_j m_j \frac{\rho_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h) = \sum_j m_j W(\mathbf{r} - \mathbf{r}_j, h). \tag{5}$$

Likewise, we can apply SPH evaluate the pressure density field:

$$p(\mathbf{r}) = \sum_j m_j \frac{p_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h). \tag{6}$$

Note that the pressure evaluated at a particle is given by

$$p_j = k\rho_j \tag{7}$$

where $k$ is a gas constant we tune.

Using equations (6) and (7), we can now calculate the $-\nabla p$ term for particle $i$, which we call $\mathbf{f}_\mathbf{i}^{\text{pressure}}$ in (1):

$$\mathbf{f}_\mathbf{i}^{\text{pressure}} = -\nabla p = -\sum_j m_j \frac{p_j}{\rho_j} \nabla W(\mathbf{r} - \mathbf{r}_j, h). \tag{8}$$

An issue with (9) is that $\mathbf{f}_\mathbf{i}^{\text{pressure}}$ is not symmetric: the force particle $i$ experiences due to particle j will not, in general, be equal and opposite to the force particle $j$ experiences due to particle $i$, violating Newton's third law. A simple fix we used is to replace $p_j$ with the average of $p_i$ and $p_j$:

$$\mathbf{f}_\mathbf{i}^{\text{pressure}} = -\nabla p = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h). \tag{9}$$

Finally, we calculate the viscosity term by applying SPH:

$$\mathbf{f}_\mathbf{i}^{\text{viscosity}} = \mu \nabla^2 v(\mathbf{r}_i) = \mu \sum_j m_j \frac{v_j}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h). \tag{10}$$

This force equation also suffers from the problem of asymmetry. Using the fact that viscosity is a function of the relative difference of velocities, we make it symmetric by replacing $v_j$ with $v_j - v_i$:

$$\mathbf{f}_\mathbf{i}^{\text{viscosity}} = \mu \nabla^2 v(\mathbf{r}_i) = \mu \sum_j m_j \frac{v_j - v_i}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h). \tag{11}$$

The remaining term we need to estimate is the external force density field $\mathbf{g}$, which is just the acceleration due to gravity $g$ in general. In the case of wall collisions, our implementation just does naive collision detection and resolution.

Now that we have the equations necessary to estimate the acceleration of each fluid particle, we can now simulate the motion of these particles.

## 4 IMPLEMENTATION DETAILS

### 4.1 Hardware/Software

We used Unity 6 and its Universal Rendering Pipeline (URP) to handle the fluid simulation, rendering, and GPU instancing. To handle the VR portion, the libraries used were OpenXR, MetaXR All-in-One SDK, and the Meta XR Simulator for faster iterating of the project.
The hardware used for this project was a Meta Quest 3 to handle the VR rendering and a RTX 3070 inside a desktop that was used to render the fluid simulation utilizing the Meta Quest 3 link feature.

### 4.2 Fluid Simulation

The fluid simulation was implemented as described in Section 3 of this paper. Some interesting things to note and were hard to get down is that to get this running at an acceptable frame rate we had to utilize Unity's GPU instancing and render pipeline by doing some of the calculations outlined in Section 3 in a compute shader. Specifically, equations (6, 9, 11) are handled this way along with any collisions in between particles here as they are needed to update the positions of our particles and are a bulk of the work that needs to be done for our simulation.
The compute shader output was stored into command buffers to be read by our rendering shader. From that, the entire flow of our simulation was:

(1) Run compute shader.
(2) Store outputs of compute shader into command buffer located inside main driver script.
(3) Read data from command buffers into rendering shader.
(4) Blit rendering shader output to screen.
(5) Repeat 1-4 as long as simulation runs.

### 4.3 Screen Space Fluid Rendering

For this, we utilized these slides from Nvidia [Green 2010].
At a high level, what Screen Space Fluid Rendering entails is rendering only particles that are directly in screen space, allowing for better performance and the possibility for more post-processing now that we have more compute to work with that is not being spent with intensive 3D calculations that attempts to accurately simulate fluid flow. To summarize how this will be implemented from the slides, we will have to generate a depth map of our particles, smooth that out using a form of blurring,
What this means for our fluid simulation, however, is turning our particles into spherical point sprites, which are quads that have pixels outside of a radius trimmed off, and having them always face the camera (seen in Fig 2). This allows us to do our physics calculations in a 2D space, since the point sprites that will be facing us are no longer spherical. From here, we can generate a depth map by finding the distance from any given fragment to the camera position.

Next, we had to calculate the normals and blur the surfaces. Calculating normals can be done at any point once we have the depth map by unprojecting any given pixel using the depth and UV coordinate

Fig. 2. Example of a single particle and a blob of them without any post processing- notice how flat they are. That's cause they are flat!

to get back to eye-space. We can take the partial differences of both the y and x axis to get two tangent vectors local to the surface (our point spherical sprite), which we can take the cross product of to get a perpendicular vector, which is our normal.Then we blurred the surfaces using a bilateral filter and calculated the normals again on this blurred surface, giving us Fig 3.

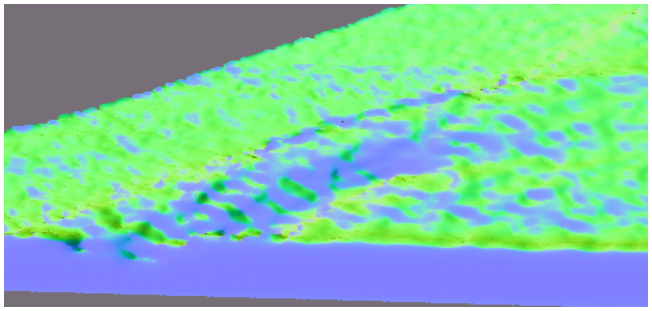

Fig. 3. A closeup of the normal map for our fluid simulation in a nearly fully stable state.

Now we can shade the surface. The slides recommended using thickness shading where we render the particles with additive blending s.t regions with more particles will appear "thicker" or in the shader for thickness, more opaque. Storing this inside a buffer, we will use this information for other shaders to sample from to calculate things such as refractions, reflections, light absorption, transparency, etc.

From here, it was just calculating refractions and reflections to shade the fluid correctly. This was just using Snell's law, and finding any hit position using the depth combined with the camera's view for any given point. We can then attenuate the final color using thickness. We have our final outcome below in Fig 5.

To summarize, our new final process with SSFR looks like this:

(1) Run compute shader for the pixels on the screen.
(2) Store outputs of compute shader particle updates into a command buffer.
(3) Read data from command buffers into both our depth and thickness shader, storing output of both.
(4) Smooth out both of these outputs and store them again together.

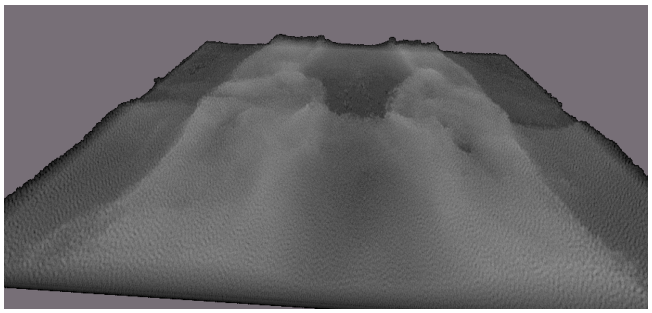

Fig. 4. Our thickness map (white = thicker, more particles are overlapping).

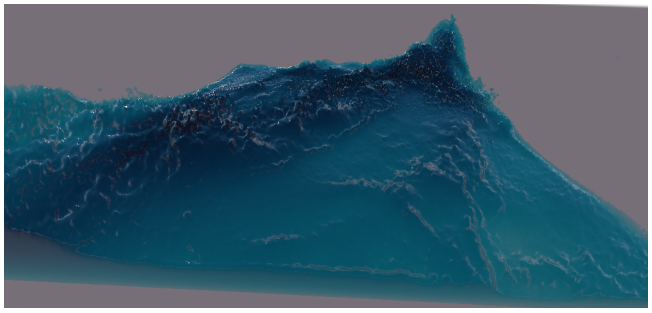

Fig. 5. Our final product with 1m particles, runs fine.

(5) Using only the smoothed depth map, reconstruct the normals from it and store into a render texture.
(6) Render the entire image using all of the above to apply all post processing effects and output to driver script.
(7) Blit rendering shader output to screen.
(8) Repeat 1-7 as long as simulation runs.

### 4.4 VR Integration

To integrate into VR, we faced a similar issue where our shaders did not have any support for stereo rendering. We used the documentation linked from the past quarter to help us get over this issue as hopefully Unity handles most of it. We quickly found out that Unity in fact does not handle most of it, and we had to attempt to rewrite most of our shaders to work with rendering in two different eyes. While we got close, there were still a lot of artifacts and issues with stereoscopic rendering that we were not able to fully finish.

Another one of the bigger issues in the project we faced was that we were unable to implement interaction with our fluids and we ran out of time to truly troubleshoot and implement this. We had a lot of bugs that came with trying to get it work with VR, and we will show and give some hypotheses on why they might be happening.

We believe this issue comes from us incorrectly implementing stereoscopic rendering, since we found that this artifacting mainly happens in areas where we think the right eye is dominant, meaning we believe we have messed up handling boundary cases for our right eye.

The next one was something we could not get down, and it mainly was because of how we implemented SSFR or something that might

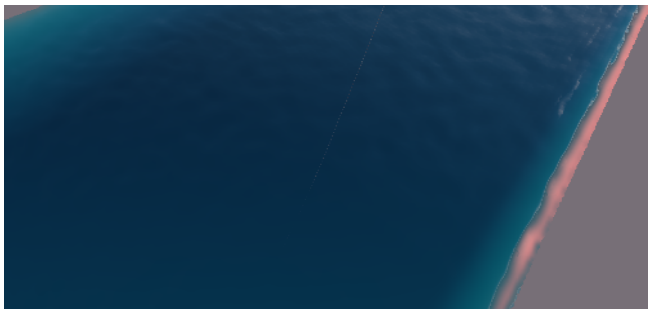

Fig. 6. Artifacting at the boundary of our simulation.

be fundamental to how SSFR works that we could not translate correctly to VR. Since the updates are done in screen space from the camera's view, when turning into VR we found that the liquid would not stay in place. In fact, it would actually follow our gaze.

## 5 EVALUATION OF RESULTS

At this point of the report, we realized a website that would be able to handle gifs would most likely have been a better idea. But we still have some interesting results and pictures to show!

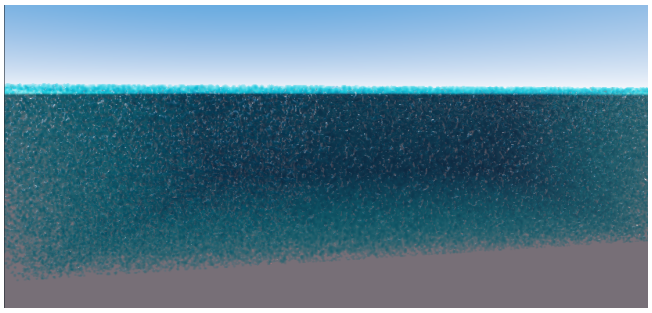

Fig. 7. Our simulation blowing up (very similarly to other SPH simulations) when we do not do enough updates on each frame at 1m particles.

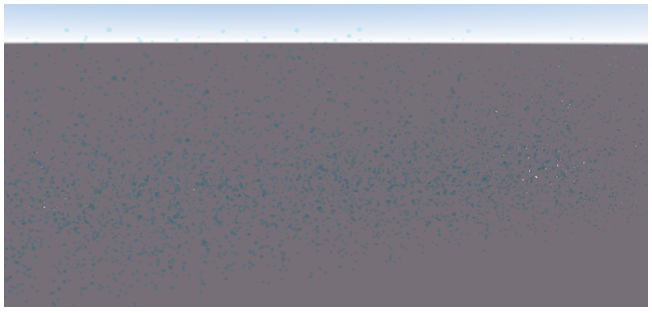

Fig. 8. Our simulation blowing up at 5k particles when we do only one iteration of our solver per timestep (hard to see).

Our fluid simulation blows up in a variety of ways. As seen in Fig 7, what usually happens is that it fills up the box with all of our

particles flying around everywhere. From our troubleshooting, this is caused by a couple of reasons. The first one is when our FPS is too low, which we found when translating to VR. This made sense, as we were now rendering to two different eyes instead of one. Since our simulation worked off a framerate based timestep, this meant that once framerate dipped too low, the updates for each step of the simulation would be too large and explode.
Another way was reducing the number of iterations per frame also caused this explosion, our guess being that if there were not enough iterations, our SPH fluid solver would have way too much numerical instability to be able to correctly calculate all the needed values that keep it at a stable state which SPH requires as outlined in section 3. This is (to our understanding) how SPH works, so our solution was to cap the timestep, and keep the iterations at a reasonable amount (we found this to be 3) that does not cause any explosions in our fluid simulation but keeps performance at a fine state.
Finally, if there were simply too many interactions from outside forces. When trying to model fast-moving water, the amount of interactions goes intensely up, and we found that we had to do more iterations per frame.
For some comparisons, we decided to use Sebastian Lague's final product, as before moving to VR we were recreating his video and his work.

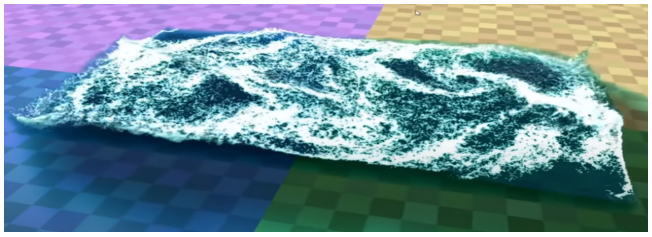

Fig. 9. Sebastian Lague's final fluid simulation from his video [Lague 2024] Notice the foam and shadows on the ground compared to our implementation.

Comparatively, we have our fluid simulation in Fig 10 where we tried to recreate the video but there was no mention of the exact details of particle amount, so unsure of what the exact setup was.

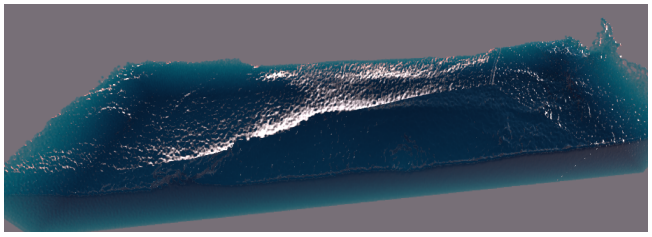

Fig. 10. Our recreation.

## 6 DISCUSSION OF BENEFITS AND LIMITATIONS

The key benefit of using SPH for fluid simulation is that it provides a relatively simple scheme for updating particle positions and velocity by interpolating properties of nearby particles. Updating these variables per iteration is also parallelizable on GPU. The main drawback is that more realistic fluid simulation requires more particles, which in turn requires more compute to simulate. The quality of the simulation is also sensitive to the simulation parameters like the gas constant $k$, the viscosity $\mu$, and the number of simulation steps per frame, and it is not intuitive how to tune these parameters. In terms of implementation, because we implemented a custom shader to speed up each simulation step, the Meta Quest 3 did not automatically render the simulation to both eyes; it only appeared through the left eye and we had to implement stereo rendering at the shader level for the project to render in both eyes. This made the project harder than anticipated, and made it such that we were unable to implement additional features as we only had 3 weeks to work on this project.

## 7 FUTURE WORK

Some additional future work we can implement is adding in the spray, foam, and bubbles that were implemented in the video to make the fluids look more realistic when they are crashing into each other. Getting SSFR fully working inside of VR is also another big thing, so bugfixing that entirely is a future extension of this project. We also still want interactability with these fluids inside since our main motivation was to use this as a baseline for our waterbending, but we found out that is most likely not feasible at a large scale. However, basic interaction, especially with a smaller simulation with SSFR looks to be entirely possible in VR. Finally, future work would involve fully bugfixing our stereoscopic rendering such that it renders in both eyes correctly and forms the image inside of our headset.

## 8 CONCLUSION

We implemented fluid simulation and rendering using Smoothed Particle Hydrodynamics and Screen Space Fluid Rendering for a VR headset. We found that doing so required heavy computational power, especially as the number of particles increased. This required the user to be near a GPU, which is somewhat impractical and diminishes the user experience. For more practical yet accurate fluid simulations in a VR setting, we recommend future work investigate either (1) non-particle-based models for fluid simulation or (2) faster particle-based simulation algorithms that can run efficiently on a GPU server, which the user could interact with through a web app. Overall, we enjoyed making this project and found it a useful step towards our ultimate goal of building a VR game.

## ACKNOWLEDGMENTS

We thank Douglass Lanman for providing the Meta Quest 3 and allowing us to change our project proposal from a Turing machine simulation. We also thank John Akers for helping us troubleshoot some Meta Quest 3 setup problems on Ed. Thank you to Sebastian Lague for his YT video on Rendering Fluids as a lot of implementing SSFR was closely following along with what he was doing and translating it into a VR format. Thank you to Andy Danforth for his work on Interactable Fluids in VR as that helped and inspired us for this project.

## REFERENCES

Simon Green. 2010. Screen Space Fluid Rendering For Games. https://developer.download.nvidia.com/presentations/2010/gdc/Direct3D_Effects.pdf Accessed: 2025-03-17.

Sebastian Lague. 2023. Coding Adventure: Simulating Fluids. https://www.youtube.com/watch?v=rSKMYc1CQHE Accessed: 2025-03-17.

Sebastian Lague. 2024. Coding Adventure: Rendering Fluids. https://www.youtube.com/watch?v=kOkfC5fLfgE Accessed: 2025-03-17.

Matthias Müller, David Charypar, and Markus Gross. 2003. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Diego, California) *(SCA '03)*. Eurographics Association, Goslar, DEU, 154–159.