Edge of Equilibrium - Physics and Particle-Based Water Simulation

Fronescu Martin-Cristian
martinfronescu@gmail.com

Găgăuță Andrei-Paul
gagautaandreipaul98@gmail.com

Velișan George-Daniel
georgedaniel230@gmail.com

Abstract

This paper covers the development of Edge of Equilibrium, a puzzle-precision platformer game in which the player needs to guide a ball through complex layouts. We detail the use of techniques such as Smoothed Particle Hydrodynamics to simulate fluids that act as obstacles for the ball.

1 Introduction

Edge of Equilibrium is a puzzle-precision platformer game in which the player controls a ball, trying to guide it from a starting point to a final destination. The main goal of the game is to maintain equilibrium and avoid falling off the course, while also providing the possibility to move fast.

The ball needs to surpass various obstacles, including some water pools (Figure 1) that challenge the player to float on.

Our goal is to simulate ball and water physics as accurately as possible. In the following sections we will discuss how we implemented Smoothed Particle Hydrodynamics (SPH) to achieve the fluid simulation.

2 Related Work

Our game is inspired predominantly by various video game classics, such as:

- **Marble Madness** (Nintendo, 1989) - the game that started it all; Marble Madness is an arcade video game developed by Mark Cerny and published by Atari Games where the player is tasked to move a marble to a finish goal, getting past various terrains and obstacles;
- **Ballance** (Cyparade, 2004) - 2004 puzzle-platformer game, developed by Cyparade and originally published by Atari Europe; the player has the same goal as before, guiding a ball to an end goal, while also having to solve various puzzles by moving certain objects in the level;
- **Hydrophobia** (Dark Energy Digital, 2010) - action-adventure video game released in 2010 that uses realistic fluid physics for its puzzles, provided by the HydroEngine (Dobson, 2007) game engine.

There are various papers covering realistic fluid simulations that were very helpful for the development of our game, with the most important one being:

- **Particle-Based Fluid Simulation for Interactive Applications** (Müller et al., 2003) - a method based on Smoothed Particle Hydrodynamics (SPH) to simulate fluids with free surfaces, utilizing the Navier-Stokes equation.

3 Method

Edge of Equlibrium was made in the Unity engine. It uses partly hand-made materials and assets (such as the ball materials and music), but also some packages with assets for decoration. We implemented the player movement, checkpoint system and water simulation, along with other features.

3.1 Game Theme

The game's theme is an homage to Ballance. It has a dream-like atmosphere, with the accompanying music only reinforcing this. The soundtrack was made to emphasize a sense of awe. That effect is achieved by using lydian chords that are not within one single key.

Edge of Equilibrium also has a secondary theme, that being the theme of nature. The player is surrounded by multiple trees at every point. The game takes place across one single day, which can be observed by the change in lighting.

3.2 Game mechanics

The player is tasked with moving a ball to an end goal. On the way they will encounter various obstacles, such as bridges, gates and even geysers and water pools. To solve these levels the player can activate material switchers that will change the ball's material to one of the following:

- Wood - is the default material, has normal mass and speed, can float on water;
- Rock - is a heavier material, has higher mass and lower speed, sinks in water;
- Paper - is a lighter material, has the lowest mass and the highest speed of all materials, falls slower and is destroyed when entering contact with water (as can be seen in Figure 2);
- Rubber - is way bouncier than the other materials, acts almost the same way as wood otherwise.

Once the ball falls off the level or touches water as paper, the player loses one life. After losing a life, the ball is transported to a respawn location (initial spawn point by default). There are also checkpoints that are activated once the ball touches them. At that point, the game updates the player's respawn location to the checkpoint's. If the player runs out of lives, they are instead sent back to the initial spawn point, no matter what progress they had previously made during their session.

3.3 Implementation

The water is made out of thousands of small blue particles that move somewhat realistically. These particles are placed inside of a bounding box.

The water particles are instanced procedurally on the GPU in a grid, as seen in the ***GridParticle.shader*** graph shader file. They are rendered with the "Transparent" RenderType parameter so that underlying objects can be seen through them. The particles also have their own material with its own properties.

The formulas in the following subsections are based on the original **Smoothed Particle Hydrodynamics** (SPH) proposal (Gingold and Monaghan, 1977).

3.3.1 Data Initialization

Firstly, we created a structure to hold the data for each particle. It has 5 fields: float pressure, float density, Vector3 currentForce, Vector3 currentVelocity and Vector3 position. The structure is defined in both our C# script file and our compute shader file.

In the ***SmoothedParticleHydrodynamics.cs*** file we initialize the list of particles to be created, while also displacing them slightly for a more realistic simulation start.

3.3.2 SPH Forces Computation

The particles cannot go outside of the water "pool"'s bounding box. If they try to go past these boundaries they are pushed back in the opposite direction, with that forced being slightly dampened.

The particles are affected according to this formula (Müller et al., 2003):

$$A_S(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j},$$

where $A_S(r)$ is a scalar quantity interpolated in position r, j is a particle's index, m_j is particle j's mass, ρ_j is the particle's density.

As a particle gets closer to another particle, their densities and pressures will rise linearly. However, we don't want these changes to occur if the particles are far away from each other. Thus, we add a smoothing kernel to the formula:

$$A_S(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(r - r_j, h),$$

where W is the smoothing kernel, h is the core radius of the particles, r_j is the position of particle j. A smoothing kernel should be normalized i.e. have its area above 0 be equal to 1.

We will use the following smoothing kernel:

$$W_{\text{poly6}}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases},$$

in density and pressure calculations.

Because the particles' masses are constant, the density keeps changing constantly. This means the density of a particle in a specific position is:

$$\rho_S(r) = \sum_j m_j W(r - r_j, h).$$

Meanwhile, using a modified version of the ideal gas law we get the value for the pressure:

$$p = k(\rho - \rho_0)T,$$

where k is a gas constant and T is the temperature. Considering that we don't simulate temperature in this project, we would actually obtain the formula:

$$p = R(\rho - \rho_0), R = kT,$$

where ρ_0 represents the resting density of the fluid's particles.

As for the pressure force, it's calculated as follows:

$$f_i^{pressure} = -\sum_j m^2(\frac{p_i}{\rho_i} + \frac{p_j}{\rho_j})\nabla W(r_i - r_j, h),$$

where ∇W is the gradient of the smoothing kernel W. In these calculations, we're using a different kernel, that being:

$$W_{\text{spiky}}(\mathbf{r}, h) = \frac{15}{\pi h^6} \begin{cases} (h-r)^3 & \text{if } 0 \le r \le h \\ 0 & \text{otherwise} \end{cases}.$$

The viscosity force's formula can be written as:

$$f_i^{viscosity} = \sum_j \mu m^2 \frac{v_j - v_i}{\rho_j} \nabla'' W(r_i - r_j, h),$$

with μ as the fluid's viscosity and v as a particle's velocity.

All of the calculations are performed on the GPU, inside the ***SPHCompute.compute*** compute shader file. The results from the shader kernels (***ComputeForces*** and ***ComputeDensityPressure***) are then sent back to the script file on the CPU.

3.3.3 Optimizations

Since each particle affects every other particle, calculating all resulting parameters is a task of complexity $O(n^2)$. In addition, the smoothing kernel function returns negligible values the further the two interacting particles are. As such, we can optimize the forces computation by only calculating the corresponding quantities for particles in close proximity.

To calculate the forces efficiently, we use an approach named Dynamic Hash Grid (Hoetzlein, 2014) to easily find neighboring particles.

Our bounding box is split into 3-dimensional cells the size of the particle radius. We find the corresponding cell for a particle using its position and label it using a custom hash function. All particles in a cell will be labeled the same.

These labels are stored in a buffer that is sorted using Bitonic sort, a parallel algorithm for sorting. We're also using an auxiliary buffer to keep track of where certain labels appear in the sorted buffer. That way, we can linearly go through the label buffer and get all the particles in a specific cell.

In the ***ComputeForces*** function, we find the corresponding cell of a particle, check neighboring cells and get their respective particles from the labeled buffer.

The complexity of Bitonic sort is $O(n \log^2(n))$ and the force computation complexity is $O(nk)$, where n is the total number of particles, and k is the number of neighboring particles, much smaller than n.

3.4 Game immersion

In order to achieve immersion, the water particles are affected by the player's ball and vice-versa. The contribution of the particles to the ball's forces does cause an effect similar to buoyancy. However, in order to make the wood and rubber balls float, we add an artificial upwards force for each particle that enters contact with the sphere. Likewise, we add an artifical downwards force to the rock ball.

If the player's current material is Paper, once they touch a water particle, the ball starts to fade out, after which they're sent back to the respawn location.

4 Future Work

We plan to continue working on this project by adding more levels with increasing difficulty. These levels will include more types of obstacles, such as bridges floating in water and boats.

As for the water simulation itself, we plan to get raymarching working on the liquids' particles. We would also like to optimize the particle computations even further.

5 Conclusion

Implementing Smoothed Particle Hydrodynamics into our game has been an interesting learning experience. Not only did we have to use compute shaders to run our code on the GPU, but we also had to learn how multi-threading works in Unity.

Obviously, we also had to tackle implementing some physics formulas for a more realistic water simulation.

Limitations

Due to how our approach to water simulation works at the moment, we are constrained to only 65535 threads for the particles. That means once we're over this limit, any newly generated particles will not move at all.

Even after optimized the code by comparing each particle only with their cell neighbors, we still experience frame rate drops. On decent hardware, this shouldn't pose a problem in normal cases (as for example level 4), but it's still important to keep in mind.

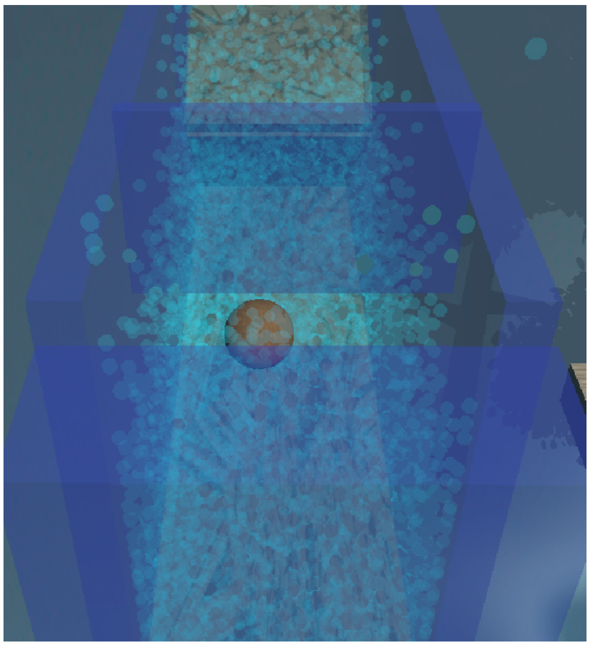
Figure 1: Water "pool" in play mode

References

Atari Europe Cyparade. 2004. Ballance. Accessed: 13.05.2025.

Dark Energy Digital. 2010. Hydrophobia. Accessed: 13.05.2025.

Jason Dobson. 2007. Product: Blade announces next-gen fluid dynamics hydroengine. Accessed: 13.05.2025.

R. A. Gingold and J. J. Monaghan. 1977. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181(Issue 3):375–389.

Rama C. Hoetzlein. 2014. Fast fixed-radius nearest neighbors: Interactive million-particle fluids. In *Fast Fixed-Radius Nearest Neighbors: Interactive Million-Particle Fluids*.

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*, SCA '03, page 154–159, Goslar, DEU. Eurographics Association.

Nintendo. 1989. Marble madness. *Nintendo Power*, (Issue 4). January-February 1989.

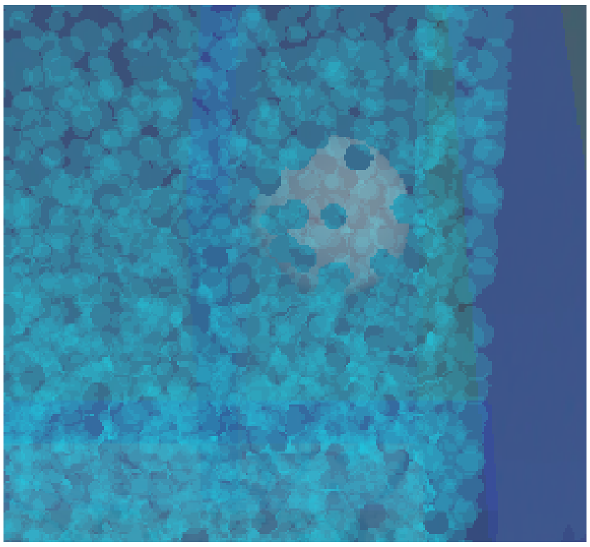
Figure 2: Paper fading out after touching water

A Appendix