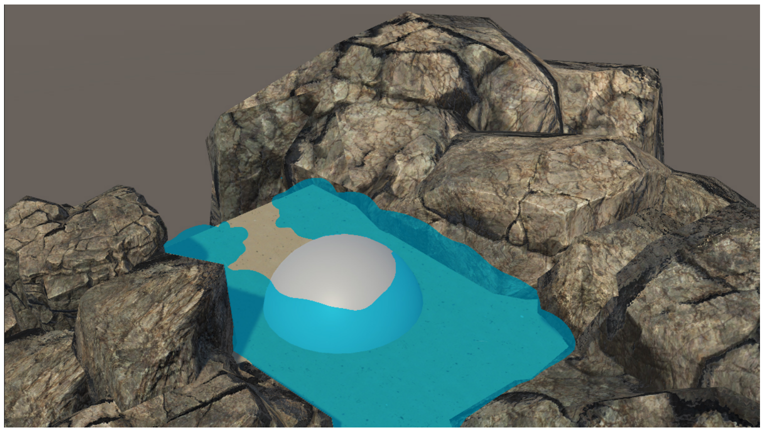
DH2323 River Flow Fluid Simulation



Martin Ryberg Laude
mrl@kth.se
200110256153

Linus Wallin
liwallin@kth.se
200008302739

KTH Royal Institute of Technology

May 17, 2024

Specification

Background

Flowing rivers are often seen in games and 3D animations. How they are implemented and visualized varies, as do their attributes. This makes visualization of flowing liquid an interesting area with a plethora of possible approaches for different use cases. For instance, realistic river interaction with the environment, such as water flowing around rocks.

The most visually realistic visualizations attempt to replicate the dynamic nature of flowing water, and simulate how the river is affected by its environment, such as rocks in a river.

Problem

Investigate how a flowing river scene can be built using simulation that allows for environmental interaction.

Method

Several methods exist for rendering realistically behaving fluids. One of these is called Smoothed Particle Hydrodynamics, which uses small particles and their interaction with each other to simulate how a fluid behaves. Particles can in turn be merged into a continuous surface with a technique called ray-marching, which uses signed distance fields to evaluate distances. We intend to combine these with tools within the game engine Unity to create a scene simulating a river flowing.

Evaluation

The project will focus on evaluating the quality of the river simulation in regards to subjective looks and environmental interaction, and in doing so judge the combination of technologies outlined above.

Contingency

If time becomes an issue, we expect to be able to outsource the raw programmatic implementations of SPH and raymarching. They are well known and not the main focus of this project. Further, the environment itself can be reduced in complexity.

1 Introduction

Fluid simulation is a complex and computationally intensive task that aims to model the dynamic behavior of liquids and gases in virtual environments. Various techniques have been developed to simulate fluids, including the Navier-Stokes equations [7], smoothed particle hydrodynamics (SPH) [5], and the lattice Boltzmann method [1]. These approaches have been widely used in computer graphics, video games, and scientific simulations to create realistic and visually compelling fluid effects.

In this paper, we present a fluid simulation of a river flowing using smoothed particle hydrodynamics (SPH) and ray marching for rendering. SPH is a meshless Lagrangian method that discretizes the fluid into a set of particles, each with its own properties such as position, velocity, and density [5]. The particles interact with each other based on the Navier-Stokes equations, which describe the motion of fluids. SPH has been shown to be effective in simulating a wide range of fluid phenomena, including free-surface flows, viscous fluids, and multiphase flows [6, 3].

To render the fluid surface, we employ ray marching, a volumetric rendering technique that casts rays from the camera into the scene and iteratively marches along each ray until it intersects with the fluid surface [6]. Ray marching allows for efficient and high-quality rendering of complex fluid surfaces, including transparency, refraction, and reflection effects.

Our simulation focuses on the realistic modeling of a river flowing through a landscape. We take into account various physical properties of the river, such as its velocity, density, and viscosity, as well as its interaction with some of the elements in the surrounding terrain. The SPH particles are used to discretize the river water, while the ray marching algorithm renders the fluid surface based on the particle positions and densities.

2 Implementation

We decided to build this simulation in the game engine Unity, as we had prior experience with the software as part of the DH2323 course. This gave us access to a 3D rendering pipeline with scripting capabilities, allowing us to focus on the particle physics and rendering of the liquids.

As a base for the Smoothed Particle Hydrodynamics (SPH) part of the work, we used the paper *Particle-based fluid simulation for interactive applications* [6], which originally introduced SPH as a method of simulating fluids. It presents the equations necessary for analytically solving each particle's parameters, which allows for the programming of such a simulation. Due to time constraints however the exact calculations between particles part of the software are done in compute shaders, which are modified from Lague, S. original SPH shaders. [4]. Without the shaders (and their multithreading) the simulation would not be performant enough to run on any of our available hardware, as a certain amount of particles are necessary for creating a convincing river flow simulation.

To use these shaders we built a C# script that handles the setup and update loop. The simulation pipeline commences with the spawning of particles in a grid pattern, with a small random jitter applied to their positions to prevent perfect alignment. The `Awake()` method initializes the necessary compute buffers and sets up the compute shader kernels with the required parameters and buffers.

The core of the simulation then lies within the `FixedUpdate()` method, which is invoked at a fixed time step. This method dispatches a series of compute shader kernels that perform the essential steps of the SPH algorithm. These steps include computing particle densities and pressures, evaluating forces acting on each particle, collision detection with a specified sphere and the bounding box, and ultimately updating particle positions and velocities.

Additionally, our shaders handle collision to some degree. The shaders themselves have to support this, so the shaders were modified to allow for collision with:

- Box bounds containing the particles (i.e walls, floor and roof).
- A sphere at a certain position and radius

The dimensions of the bounding box and the parameters of the sphere are passed in from the Unity script to the shaders. The reason for including the bounding box is to simulate a river bed, shores and an entry/exit of our river slice in a simple way. The spherical collision was on the other hand included because we wanted a spherical "rock" in the river to highlight the wake it would make in a river, which turned out very well in the end.

Further, we decided to perform more rendering in the `Update()` method. For debugging purposes, and out of interest, we wanted the particles themselves to be optionally visible too. Until we had a raymarching implementation in place, this was entirely necessary to visualize the effects of the code so far. We used Unity's built in `Graphics` library to do so easily.

At this point, after tweaking the SPH parameters, we had visible particles spawn in at once and drop to the floor, in a fluid-like fashion. To get the particles to "flow" like a river, we needed to implement two additional steps:

- A constant directional force acting on each particle
- A way of spawning and despawning particles continuously when they reach certain positional values

The first step was easily tackled by modifying the shader further by applying a constant float value in the x-field in the force vector of each particle. We think this simulates fairly well how a flowing body of water behaves without calculating the whole river from start to end, and visually it looks convincing.

The second step was to implement a way to transfer particles from the end of the river to the beginning to keep it flowing infinitely. This was done through checking if a particle had reached the far side (in the direction the river was flowing) of the bounding box and resetting its location to the near side (at the start of the river flow).

3 Results

The result of our implementation is a simulation of a river where the water is pushed to the side by a spherical object representing a rock that lies in the middle of the river. This is displayed in Figure 1 and in the demo video of the simulation in Appendix A.

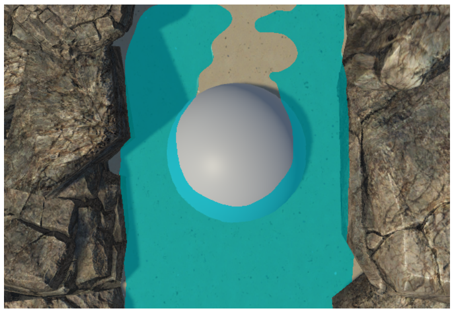


Figure 1: River Simulation From Above

4 Discussion

4.1 Future Work

To develop the river simulation further we could try to optimize the way which the forces on each particle is calculated, since the code currently loops through all particles in the scene when computing the forces that a particle is experiencing. By only looping through the neighbors of the current particle the time complexity of the code could be reduced significantly which would improve the quality of the simulation and also allow for simulation of more particles. This was something that we tried to improve through spatial partitioning. We took inspiration from the presentation *Fast Fixed Nearest-Neighbors* by Hoetzlein [2] which describes a method to partition the space into a grid where each cell is the size of a particle. With this it is possible to only look at particles in neighboring grid cells, which reduces the time complexity of fluid simulations and would allow us to implement even more calculations such as surface tension to create a more realistic river simulation. Unfortunately we were not able to get the partitioning into grid cells to work due to time limitation.

Since the code runs on the GPU it was a lot harder to deal with collisions with other objects in the scene, as we did not have access to the collision meshes in unity. This was the main reason to us only using a sphere as an object which the particles could collide with. We tried to implement collisions with boxes of different sizes, but with the limited time we did not manage to create something that worked. For the future this could be a possible way to improve the project and create a more diverse river simulation.

4.2 Individual Contributions

Most of the work that went in to this problem can be credited both authors equally, as work was conducted in live pair programming sessions. In the rare instances where that was not the case, the divison of labor looked somewhat like the following:

Linus focused more on the shader modifications such as collisions, particle bounds and optimization.

Martin focused more on the C# scripts and getting the pieces to play well together in Unity.

References

[1] Chen, S., and Doolen, G. D. Lattice boltzmann method for fluid flows. *Annual review of fluid mechanics 30*, 1 (1998), 329–364.

[2] Hoetzlein, R. Fast fixed-radius nearest neighbors: Interactive million-particle fluids. Nvidia. Accessed: 2023-05-14.

[3] Ihmsen, M., Orthmann, J., Solenthaler, B., Kolb, A., and Teschner, M. Sph fluids in computer graphics. *Eurographics (State of the Art Reports) 2* (2014), 21–42.

[4] Lague, S. Fluid-sim. GitHub repository, 2024. Accessed: 2023-05-14.

[5] Monaghan, J. J. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics 30*, 1 (1992), 543–574.

[6] Müller, M., Charypar, D., and Gross, M. Particle-based fluid simulation for interactive applications. vol. 2003, pp. 154–159.

[7] Stam, J. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), pp. 121–128.

Appendix A - Video Demo

Video Demo Link

Appendix B - Blog

Blog Link