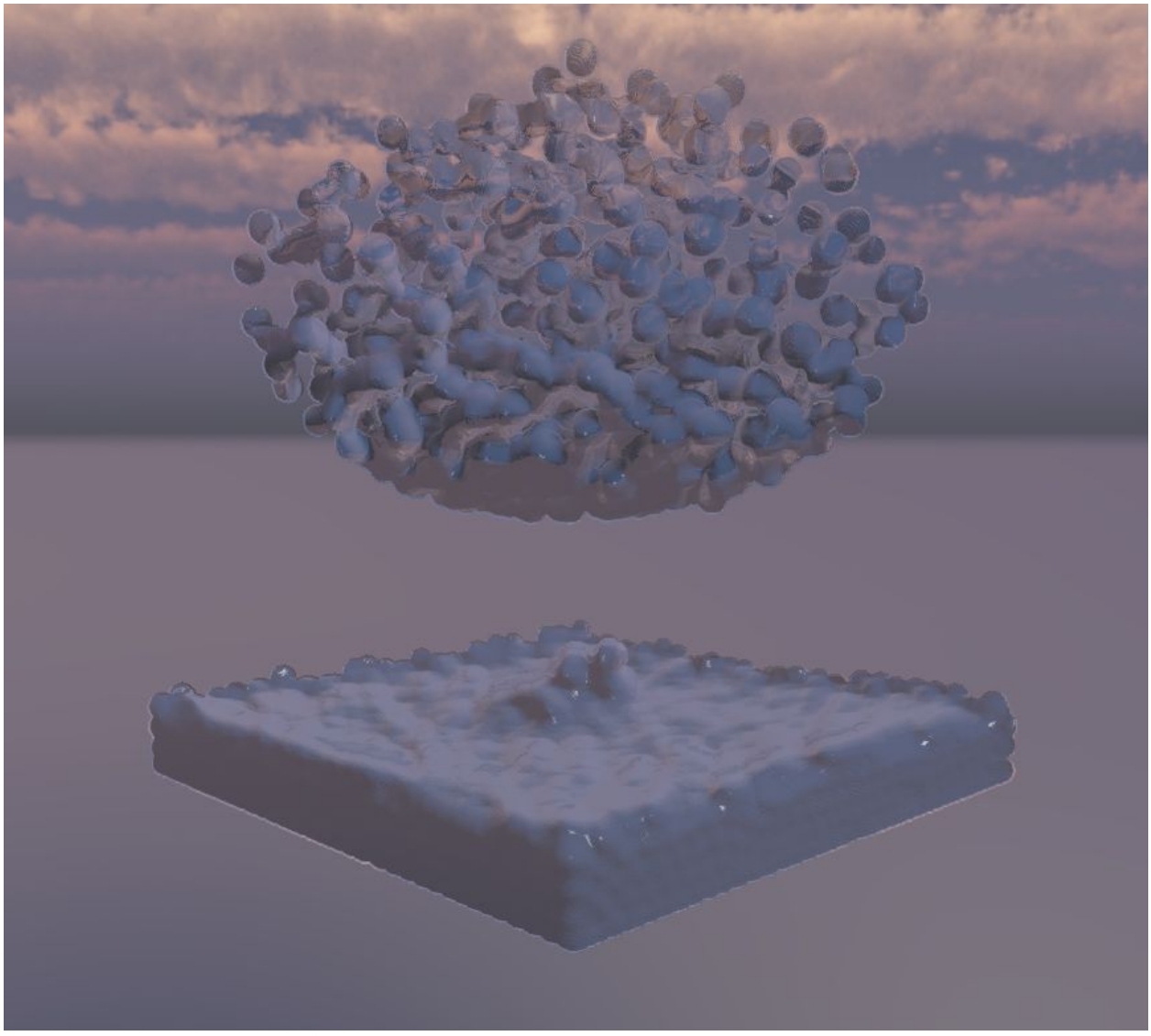
SPH Simulation and Rendering

Q Bayo (bayo0006), Ben Lemma (lemma017), Dan Shervheim (sherv029)



Project Goals

The goal of this project was to explore an SPH fluid simulation. First, we planned on implementing the SPH algorithms by modifying a previous project (assignment 1, the particle system). We also planned to implement a spatial data structure to speed up the nearest neighbor searches. We also wanted to focus on improving the rendering of the

surface over the typical "particles as spheres" approach. Given enough time, we also wanted to explore procedural sound generation from our simulation.

Working Features

We implemented the SPH simulation described by Clavet et al. in *"Particle-based Viscoelastic Fluid Simulation."* We also implemented a screen space method to render the SPH simulation as a convincing fluid surface, the majority of which follows Van der Laan et al. *"Screen-space Fluid Rendering with Curvature Flow."*

Analysis of the Project

The general idea behind SPH is that fluid is modeled as a sufficiently large collection of sufficiently tiny particles. One of the biggest elements in simulating convincing fluid is incompressibility. Essentially, mass must be conserved and so areas of high density fluid will flow into areas of low density fluid, in an attempt to minimize the differences in pressure. We model this by introducing spring forces for each particle, the magnitude of which is based on the density of neighboring particles. We calculate the density of each particle to be the sum of the inverse squared distance from all of the particles surrounding it. We then use our density to calculate a pressure force. After calculating the pressure forces for each particle, we then use Eulerian integration to find the new particle velocities and positions. Finally, we restrict the positions of the particles to be within a specified "bounding volume" by simply clamping the positions component-wise and reflecting the velocity by the bounding volume normal.

We followed the algorithm proposed by Clavet exactly, with one minor deviation in that we compute our particles in parallel on the GPU. In practice, the algorithm remains mostly the same. For further details, please see Clavet et al.

We also wanted our simulation to look nice, while still running in reasonable framerates. To accomplish this, we implemented (most of) the screen-space surface reconstruction

method proposed by Van Der Laan et al. *"Screen Space Fluid Rendering with Curvature."*

In short, we use two cameras. One renders the scene as normal (without the water). A second camera then renders a sphere at each particle position. Rather than shading the sphere with lighting calculations, we simply output the depth of the fragment (distance to camera). This essentially yields a depth texture. We then smooth this depth texture via bilateral filter convolution (implemented on GPU as compute shader) to reduce the appearance of "spheres" and introduce a more "uniform" surface. It was important to use the bilateral filter (over, say, a Gaussian filter) as bilateral filters do not smooth out edges. This allowed us to smooth relatively flat areas, while still maintaining the proper outline of the various parts of the fluid. Finally, we reconstruct the world position of each fragment and take the screen-space derivatives to recalculate the world space normals. Once the normals are reconstructed, we can simply shade the surface as we would any standard mesh, via fragment shader. In our simulation, we used the following formula (roughly):

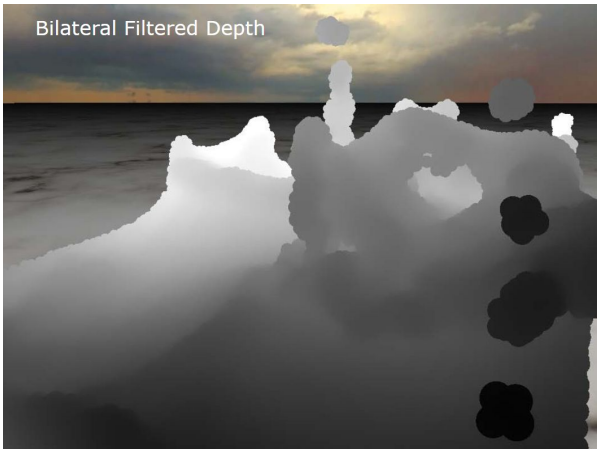
color = diffuseColor*wrappedNDotL + phongSpecular + fresnel * sceneReflection.

We apply one final trick, and that is to offset the scene texture (i.e. what's behind the fluid) by the normal of the fluid, and then lerp 50% between the opaque fluid and the warped scene to produce the appearance of a transparent fluid which is "refracting" the surface behind it. The only part of Van der Laan's method we didn't implement is the density-based transparency. Instead, we utilize a constant 50% transparency throughout the entire fluid volume.

The advantage of doing the surface reconstruction in screen space is that the complexity does not depend on the number of particles in the simulation. The complexity is instead dependent on the percentage of the screen occupied by water.

For example, 1 million particles could be simulated but if the user is facing away from the water, no extra computation time is wasted on rendering them.

The following images are from a GDC talk given by Van der Laan. They demonstrate the generated depth texture, the smoothed depth texture, and the diffusely shaded surface (shaded by reconstructed normals). We follow the same method and achieved almost identical results.



Difficulties Encountered

One of the difficulties we encountered during the SPH fluid simulation was implementing the spatial data structure on the GPU. In the end, we decided to abandon it and run a brute force nearest neighbor search, while simulating fewer particles (~7500).

We also faced difficulties with parameter tuning. We could not find a good default and so, by trial and error, stumbled upon a set of parameters which seemed to look reasonable. We also had a lot of stability issues which we mitigated by clamping the magnitude of the velocity so it did not exceed a specified threshold.

Ideas for future work

SPH is a fully mesh-free particle based method, where the particles carry the material properties of the medium it is simulating (fluid, gases). In this case, for future ideas, we would be interested in the Two-way object coupling. This will allow interaction between particles such as to move by other colliding geometry and push back. Furthermore, we would like to explore more realistic force models such as mesh-based surface tension and buoyant force.

In addition, we would like to work on increasing the efficiency of screen space rendering. Currently, the bilateral filter is applied by convolving a 2D kernel over render texture; we would want to separate it into X and Y passes to reduce computation time at the cost of minor artifacts. We also like to work on rendering particles as camera-facing quads rather than spheres. Our last two future ideas are generating procedural sound and increasing efficiency of SPH simulation. Currently, O(n^2) brute force implemented in compute shader (apparent time in O(n) due to the parallel nature of GPU). if we implement a spatial data structure, we would have a much better time complexity.