Beach Simulation in Unreal Engine 4

MMA Course Project

Task Specification

The specific task statement is selected to be as follows: “implement a simulation of water interacting with sand particles under the influence of external forces”. Here the external forces are meant to represent waves, whose emergence is not a part of the simulation and has to be added as an external condition.

Selected Approach

There exist a number of approaches to simulate fluid dynamics for computer animations. A large number of techniques rely on the problem discretization: dividing the problem into individual chunks and then calculating the state of each chunk in a given moment in time. For the sake of this discussion the studied approaches can be separated into two categories based on the domain of the basic discretization entities: spatial (the space of the simulation is divided into discrete parts for which each of which the computation is executed), and particular (the matter of the fluid is approximated by a set of points, whose movements under the influence of hydrodynamic forces is the subject of calculations). Either technique has its applications and comes with its own advantages and disadvantages. There exist alternative approaches to fluid simulation that do not fall neatly into one or the other category, but they will not be analyzed here for brevity. The spatial and particular approaches are then compared through the lense of the specified task.

Spatial Approach

The spatial discretization approach to fluid simulation is powerful and versatile. The main idea revolves around the Naiver-Stokes equations that provide a way to keep track of matter movement between discrete regions of space. A detailed description of the equations can be found in [1] (Stam) and [2]. These articles also describe the exact way the equations can be used for fluid simulation. The technique analyzes the state of the simulation domain and applies the aforementioned equations iteratively to assign a number to each point in space based on the beginning state. The number stands for the density of the matter in the given point. For the discrete case the computation is executed for a predefined set of points (e.g. layed out in a uniform grid) and the density data is then used for visualization. The more points are selected in the set, the finer the flow approximation is.
A big downside of the spatial technique is the computational complexity. In order to calculate and visualize the layout of matter in the simulation domain each point of the discretization must be recomputed even for points with no matter in them. Another bottleneck of the computation is the application of the Naiver-Stokes equations. In order for a simulation to fulfill the constraints of the mathematical model a large system of linear equations must be solved using for example Gauss-Seidel iterative method (see [1] for a precise explanation) or other appropriate methods which can provide better computation speed. The spatial technique is not trivial to parallelize. The solution of the system of equations generally requires a lot of communication between threads which can introduce large overhead.

The basic spatial approach is defined for fluid simulations with a single type of liquid and is not trivial to extend it to a more general case of multiple types of liquid interacting with each other. A possible extension for the density based method to include multiple fluid types in a simulation is to solve Naiver-Stokes equations for each type of liquid separately with additional effect of the velocity fields of all the other fluids. This is then applied iteratively, each next iteration using vector fields computed in the previous one. The iterations terminate when a convergence is achieved. This solution introduces a lot of additional computations for every frame and is not usable. A more reasonable solution can be achieved using the other approach.

Particle-Based Approach

Similarly to the previous approach, the particle-based approach to fluid simulation models the fluid flow based on a discretization, however it approximates the matter of the fluid with discrete chunks instead of the entire domain of the simulation. This can be viewed as a representation of the reality that every fluid is made up of small particles - molecules, but in practice the particles usually do not aim to represent individual molecules but rather larger parts of the matter and hence are modeled to have different properties.
The subject of the computation for every frame is the position and velocity of every particle. The computation is based on the same Naiver-Stokes equations, but as opposed to the spatial approach the equations have to be redefined to express pressure and velocity for particles instead of spatial parts of the simulation domain (see [3] for more information). The position of a particle in a moment of time is computed based on its position and velocity in a previous moment as well as the pressure and viscosity forces acting on it. The modified Naiver-Stokes equations serve to describe the effecting forces produced by a particle on another particle. The effect of a particle pressure and viscosity forces acting on a different particle is weighted based on the distance from particles.
The precision of the simulation using particle-based approach relies on the number of particles used for computation and a believable fluid animation requires relatively large numbers of particles.
The particle-based approach has several disadvantages. The simulation relies heavily on the input parameters and is not stable for many configurations. The stability of the particle simulation is not guaranteed in general (there have been proposed algorithms that use additional steps to ensure the properties such as incompressibility are preserved throughout the simulation - see [4]). Since the quality of the simulation's output depends on the particle density (number of particles used to approximate a unit volume of fluid), the computation complexity of the simulation depends on the volume of fluid in the scene, which is not the case for the spatial discretization approach.
A big advantage of this approach over the previous is that it is relatively simple to parallelize. The position and velocity of every particle can be computed independently based on the data from the previous frame. Another way to optimize the calculation is to limit the number of particles that can influence a particle. [3] introduces the notion of a kernel function for weighing the influences of the particles on a given particle. The kernel function is non-zero for a finite radius and zero everywhere else. This fact can be used during the simulation and the number of tested particles may be significantly lowered if an appropriate auxiliary spatial data structure is used (for example a uniform grid (implementation in [4] makes use of a GPU-friendly version of uniform grid).

Another big advantage of the particle-based approach is that it is relatively simple to extend the model to handle more than 1 type of fluid in a simulation. In the equations described in [3] several constants appear that denote specific fluid properties (namely fluid viscosity and mass per particle). By substituting the corresponding constants into the equations during the computation a believable simulation of different types of fluids interacting with each other can be achieved. This is relevant for the specified task of the project, as sand can be viewed as fluid. For this and other reasons the particle-based approach to fluid simulation is used in this project.

Implementation

The implementation uses the particle-based fluid simulation approach as discussed in the previous section. The main class containing the simulation logic and other entities is the Beach Simulation Actor (see BeachSimulationActor.h for class declaration). The class contains the data structure with particles (specified define macros can be used to switch between a 1D array and a uniform grid), as well as the domain of the simulation in form of a static mesh component, an array of fluid type descriptors and a special entity responsible for the particle visualization. The header file also contains multiple macros that can be used to switch between different computation techniques.
The static mesh of the simulation domain is not visible during the game by default and only serves to indicate the boundaries of the simulation in the editor. Although the mesh can be rotated arbitrarily, the domain of the simulation is always an axis-aligned box (calculated from the AABB of the domain mesh). This is the case due to the simplicity of boundary condition calculations.
A particle contains the information used during the simulation (see SimulationParticle.h for definitions). Besides the particle position, pressure, density, and velocity, there is an index of the corresponding fluid type. This index is used to retrieve the corresponding constants from the array of fluid types in the main simulation class. The particle struct design is inspired by [5]. The file SimulationParticle.h also contains the definition of the uniform grid ASDS used to find neighbor particles and the struct for the fluid type descriptors.
The beach simulation actor class also contains arrays of particle spawn objects and external force objects. The spawn objects spawn particles according to the fluid descriptors of the corresponding index. They can spawn particles automatically at the beginning of the simulation or on demand (not used in the implementation). The external force objects act on any particle that is located within their effect regions adding a predefined force to the resulting force of the particle. The direction of the force is selected based on the orientation of the direction mesh component.

Visualization

The base class for particle visualization is ParticleVisualizerComponent (see ParticleVisualizerComponent.h for definitions). It defines an interface for a particle visualizer, mainly the function for updating the visualization based on a pointer to the current array of particles. There are two classes that inherit from the ParticleVisualizerComponent. The first - ParticleSphereComponent (see ParticleSphereComponent.h for definitions) - implements a naive approach where each particle is represented with a static mesh. It is simple and useful for debug purposes but starts introducing noticeable overhead due to a large number of draw calls with larger particle arrays.

The second visualizer option - ParticleInstansedComponent (see ParticleInstansedComponent.h for definitions) - utilizes the instanced mesh component feature of Unreal Engine 4. Thanks to the fact that all particles are visualized with the same static mesh, the data relevant for the visualization can be supplied to the instanced mesh component and the number of separate draw calls can be reduced significantly.
The instanced mesh component also allows to supply additional information about the individual instances to the material which has two applications in the project. The first is to differentiate between the types of fluids. The material selects the diffuse color of every particle based on the supplied type index. The second application helps simulate foaming of the waves at the beach. The material is supplied with an additional number for each particle (namely the ratio of the velocity vector norm to the density at the particle - this aims to simulate the emergence of air bubbles at the points on the surface that have a lot of kinetic energy and are exposed to a lot of air).

Results

The performance of the final implementation depends heavily on the initial configuration of the scene. In a general case (i.e. two fluid types, spawn regions occupying approximately a third of the simulation domain), the simulation runs in 10 fps (~110 ms/frame) with approximately 65000 particles in total. Due to the instability of some configurations and no explicit incompressibility constraints the ASDS may perform differently for different particle layouts. For this reason it is difficult to meaningfully compare the computational efficiency of the naive approach and the final approach. The naive approach (one that does not use the uniform grid ASDS) generally performs much worse, reaching 100 ms/frame with just a couple thousand particles.
Following are the screenshots of the simulations running in the Unreal Engine for different initial configurations of the scene.

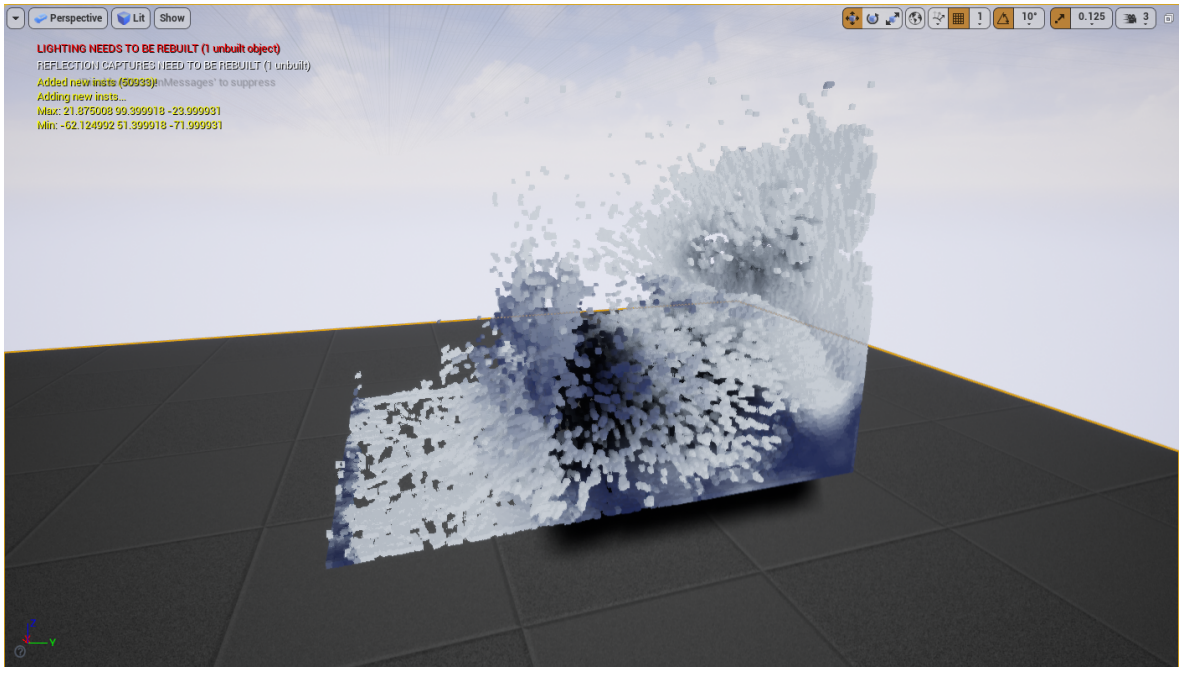

Figure 1: Simulation for 1 type of fluid (water) and an external force pushing the fluid to one side of the domain

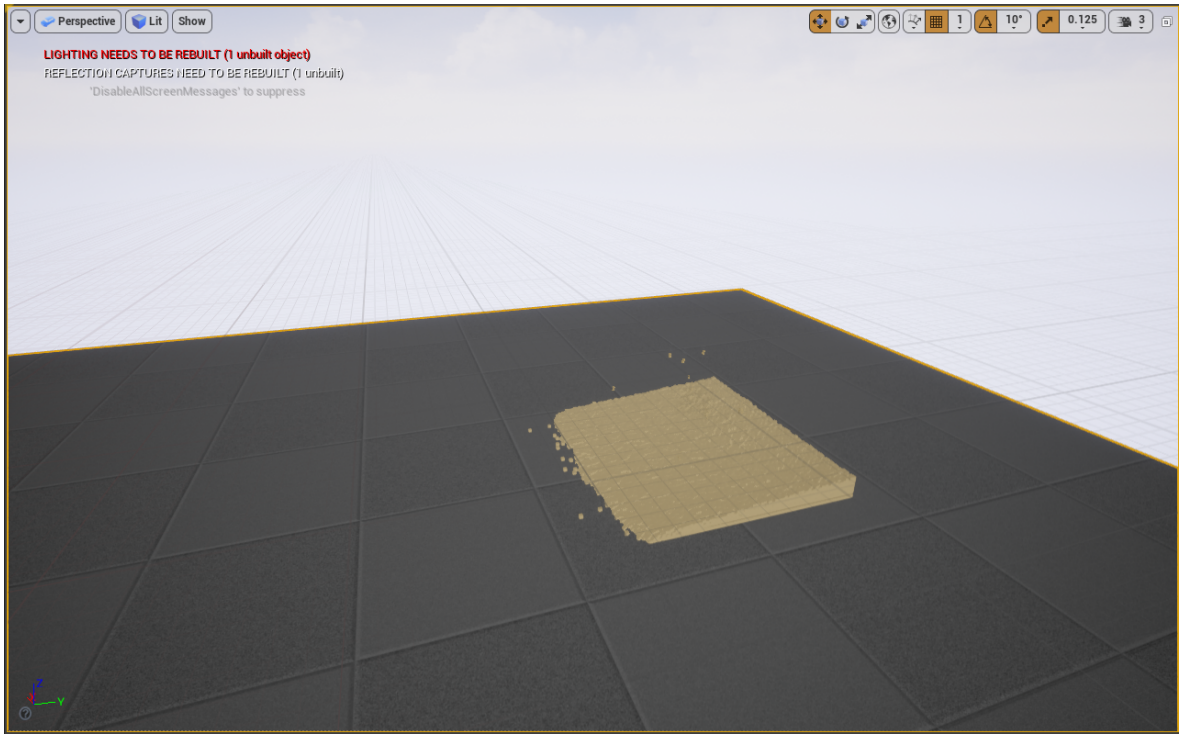

Figure 2: Simulation for 1 type of fluid (sand)

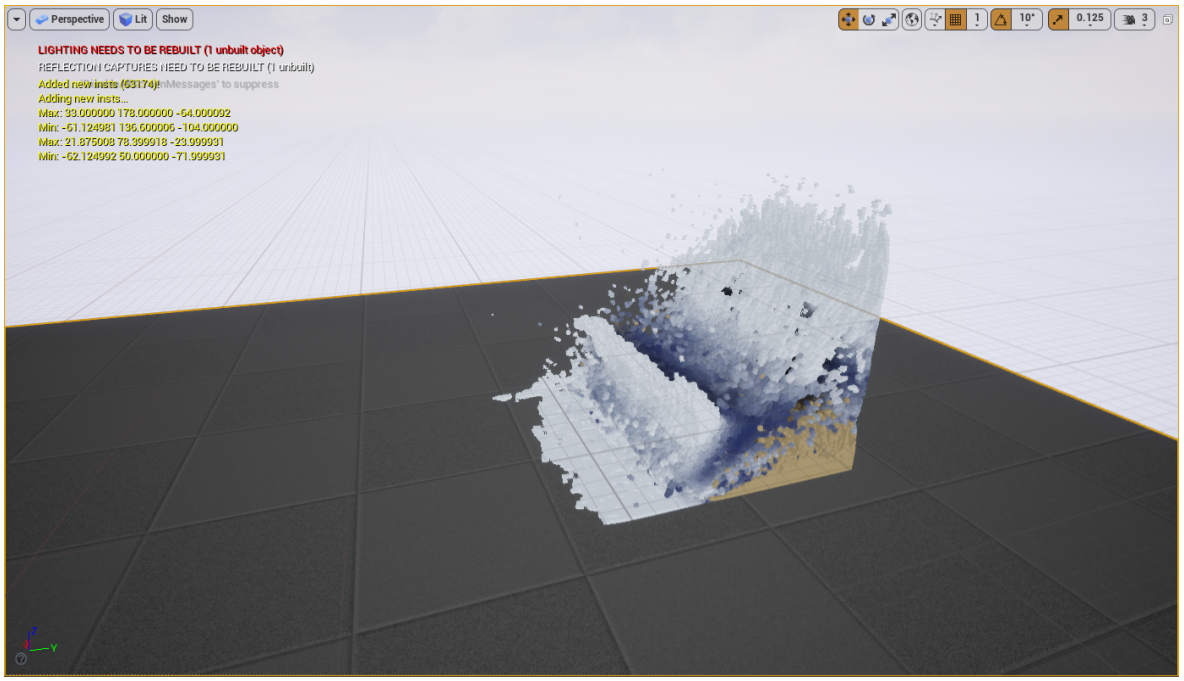

Figure 3: Combination of two fluid types

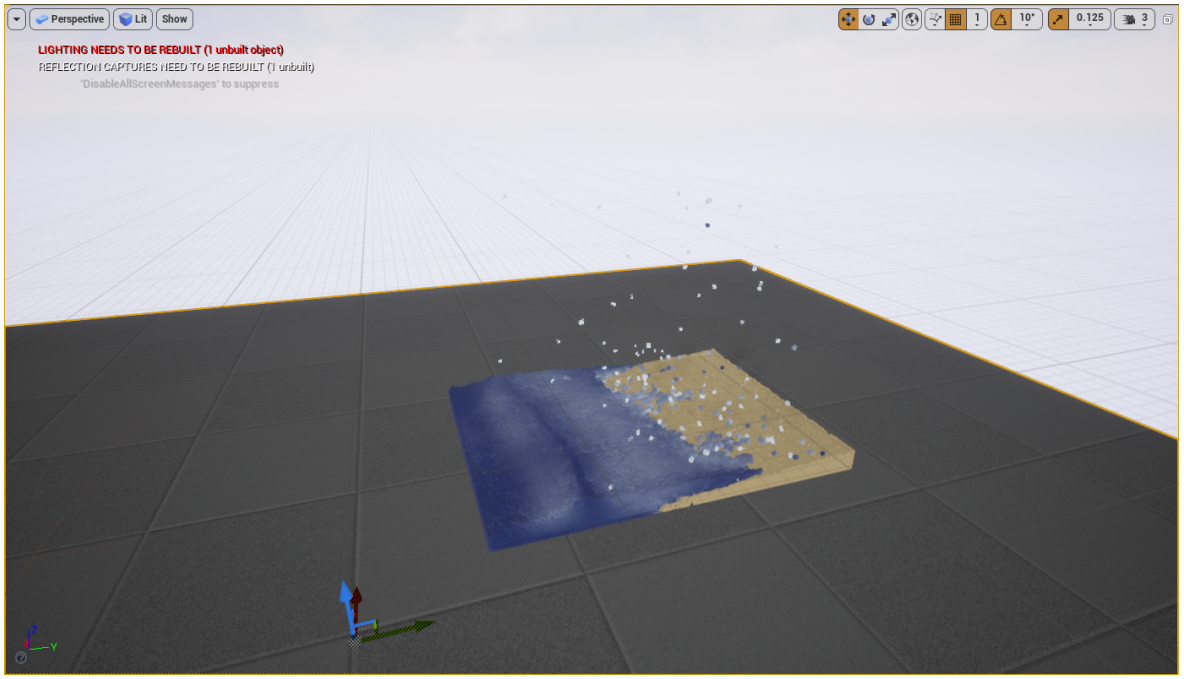

Figure 4: Combination of two fluid types - less effect of the external force

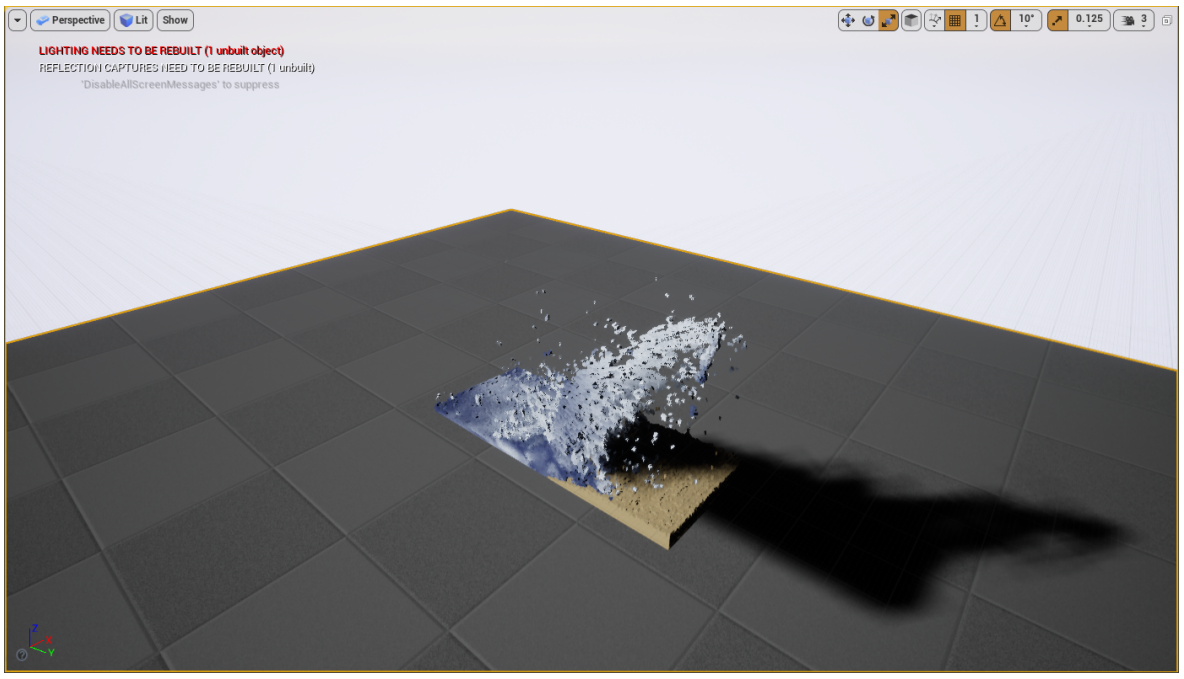

Figure 5: Combination of two fluid types - alternative scenario

Video demonstrations are available here:

- https://drive.google.com/file/d/1VaF9WZ8ykrj39uOcUqxWAFSznuyHJ6aU/view?usp=sharing
- https://drive.google.com/file/d/1wVqXW_ivjddSsEsnPjJqistA4imzRaZ-/view?usp=sharing
- https://drive.google.com/file/d/1wAOo0XCmaQY_cVnVr3LhYNG2YTxNfApe/view?usp=sharing

Issues and Future Work

The current implementation has several issues and can be improved in a number of ways. There is an issue with the naive extension of the mathematical model of the simulation: the viscosity of the sand starts influencing the particles of the water creating more friction between the fluids than is believable. As the simulation approaches a stable state there start to occur fly-away particles from the border of the fluids indicating that the mathematical model of the fluid interactions must be improved.

The simulation is significantly limited by the number of particles feasible for the computation in close to real time. This can be improved by moving the computation on the GPU. A possible approach to constructing a uniform grid data structure suitable for highly parallel computation on GPU is described in [6].

There also is room for improvement in the visualization approaches. A widely used technique for visualizing fluids is marching cubes which allows to generate the border between the mediums and thus can realistically visualize the surface of the water (see [7] for an example of marching cubes algorithm for water visualization). The implementation then has to be extended to allow for different visualizer entities to be responsible for different fluid types.

References

[1] Stam, Jos. *Real-Time Fluid Dynamics for Games*, https://www.dgp.toronto.edu/public_user/stam/reality/Research/pdf/GDC03.pdf. Accessed 6 January 2022.

[2] Stam, Jos. “Stable Fluids.” https://www.dgp.toronto.edu/public_user/stam/reality/Research/pdf/ns.pdf. Accessed 6 January 2022.

[3] Müller, Matthias, et al. “Particle-Based Fluid Simulation for Interactive Applications.” *Eurographics/SIGGRAPH Symposium on Computer Animation (2003)*, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.581.1542&rep=rep1&type=pdf.

[4] Macklin, Miles, and Matthias Muller. “Position Based Fluids.” *ACM TOG 32(4)*, https://mmacklin.com/pbf_sig_preprint.pdf.

[5] “Smoothed Particle Hydrodynamics Fluid Simulation | Ryan Guy's Portfolio.” *RLGuy.com*, 2015, http://rlguy.com/sphfluidsim/. Accessed 6 January 2022.

[6] Green, Simon. “Particle Simulation using CUDA.” *Nvidia*, 1 May 2010, https://developer.download.nvidia.com/assets/cuda/files/particles.pdf. Accessed 6 January 2022.

[7] Du, Shuchen, and Takashi Kanai. “GPU-based Adaptive Surface Reconstruction for Real-time SPH Fluids.” *WSCG 2014 Conference on Computer Graphics, Visualization and Computer Vision*, https://dspace5.zcu.cz/bitstream/11025/11923/1/Du.pdf.