Fluid Simulation

Rudolf Ortner
Johannes Kepler University
Linz, Austria

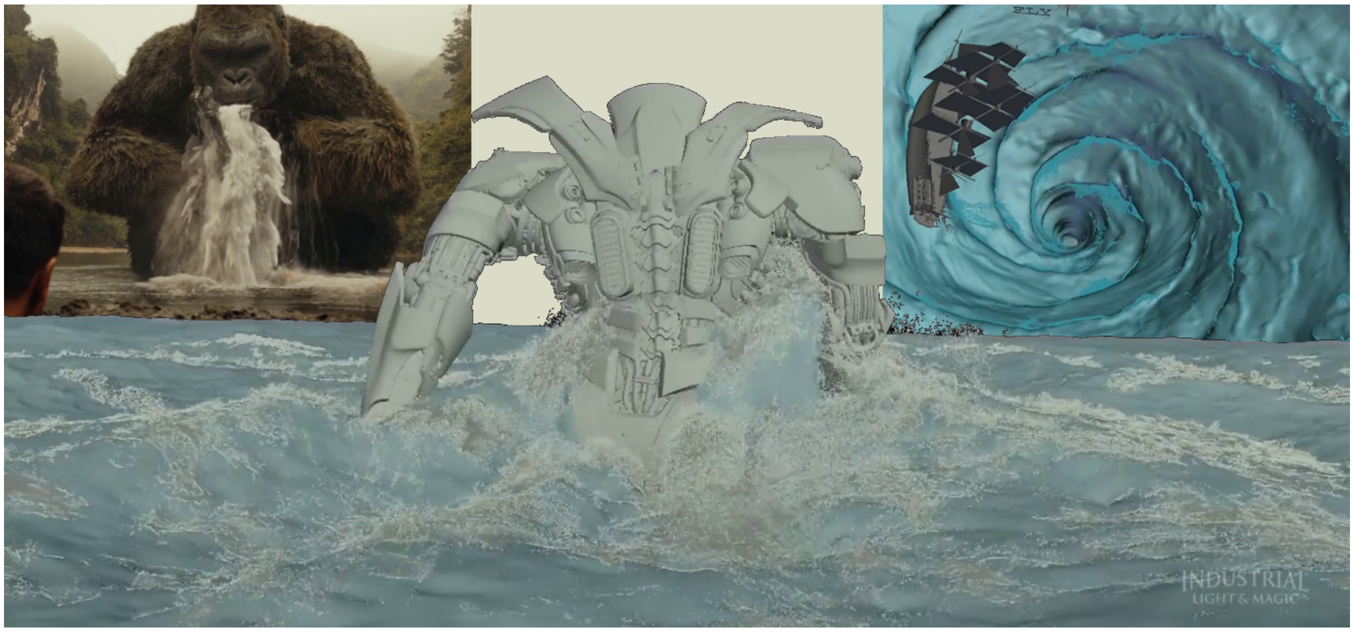

Figure 1: Great fluid simulation examples by ILM [8]

ABSTRACT

Fluid simulation becomes more and more important especially in visual effects. Since the field is very broad we decided to write this paper as a small introduction to computational fluid dynamics and its methods.

CCS CONCEPTS

• **Computing methodologies → Physical simulation**.

KEYWORDS

fluid simulation, overview

1 INTRODUCTION

The area of computational fluid dynamics is huge. Many different approaches have been developed and a lot of them can be combined to form new methods. We would like to show the underlying physics of every fluid simulation and the basic methodologies that all other inventions build upon.

2 WHAT IS A FLUID?

In physics when we talk about fluids we can either mean gases or liquids. This is because many physical laws apply to both of them and only differ in there influence and small nuances. This is an important fact because of their similarities many algorithmic approaches and data-structures for simulating them are shared beyond those. Nevertheless this presentation focuses primarily on liquid simulations.

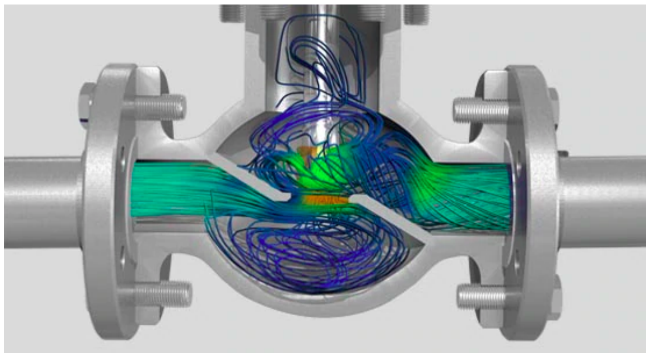

Figure 2: Simulation of the flow of water in a pipe [1].

3 WHY FLUID SIMULATION?

Now we want to point out some areas in which fluid simulations in general are used a lot. Only one aspect of it is its usage in the field of engineering. In the development phase of expensive and complicated products, often simulations are used to check if everything performs as it should. As we can see in figure 2 their CFD simulator visualizes the flow of liquid with simple stream lines. In this use

case that is totally fine because full liquid surface reconstructions are mostly not required.
And of course the biggest area of them all, at least in our humble opinion, is the one of visual effects (VFX). Today, the demand on realistic looking fluid simulations is rising and also the requirements. Simulations have to be more realistic, last longer and contain much bigger scenes than 10 years ago. As the examples in figure 1 from Industrial Light & Magic (ILM) show, huge regions of water have to be simulated. In this image from Pacific Rim for example, you can see highly detailed waves with splashes and foam on them. All in all, their visual effects look great!

4 THE PHYSICS

The underlying formulas for all compressible and viscous fluids are the so called Navier-Stokes equations. As one can see, these are quite complicated and difficult to solve for an exact solution. These formulas contains various kinds of values.

$$\rho \cdot \left(\frac{\partial u}{\partial t} + u \cdot \nabla u\right) = -\nabla p + \nabla \cdot \left\{\mu \left(\nabla u + (\nabla u)^T - \frac{2}{3}(\nabla \cdot u)I\right) + \zeta\,(\nabla \cdot u)\,I\right\} \rho \cdot g \quad (1)$$

To make the simulation process easier, often the so called Euler formula is used. These formula does not have the viscosity term in it and has a second condition which makes the formula hold for incompressible fluids. As we are going to examine liquid simulations in this talk, this is no problem because liquids are almost incompressible. We want to pronounce here the word “almost“. If liquids were totally incompressible, then we wouldn’t be able to hear anything below water, because sound moves as small changes in pressure.

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = -\frac{1}{\rho}\nabla p + g \quad (2)$$

$$\nabla \cdot u = 0 \quad (3)$$

5 METHOD OVERVIEW

Now lets have a look at some existing simulation methods. At first let’s discuss the two viewpoints. The Eulerian viewpoint and the Lagrangian viewpoint. Image you are a particle with some temperature T that moves through the air. In the Lagrangian you always look at your temperature that does not change. In a Eulerian point of view, you look at the temperature at a certain point in space. So when the air moves past, then also the temperature changes, even if the temperature of every single particle is constant.
These two viewpoints allow us to reformulate the Navier-Stokes equations for different use cases. This brings us to the two main method types, grid based methods and particle based methods. In general, grid based methods are more accurate but also harder to program. On the other hand, particle based methods might be easier to implement but are often not that accurate.
That brings us to hybrid methods which mainly boil down to the Particle-in-Cell method or the Fluid Implicit Particle method, which we will discuss later.

5.1 Grid Based Simulations

5.1.1 From Reality to Simulation. Now let’s start with grid based simulations. At first we have to somehow go from reality to simulation. In the computer it is impossible, to perform a continuous simulation. Therefore we have to discretize time. We do this with so called CFL condition.

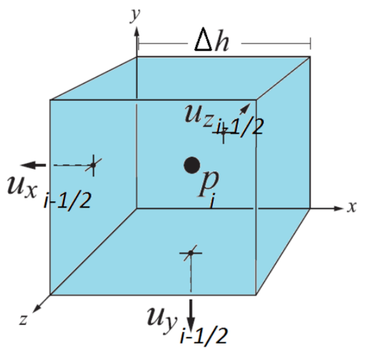

Figure 3: Visualization of a MAC grid cell [2].

$$\Delta t = \frac{\Delta h}{\vec{u}_{\max}} \quad (4)$$

Put simply, we don’t want any quantity q in the grid to move farther than the width of one grid cell per time step. To ensure this we divide the cell width by the maximum velocity that exists in the field to retrieve the size of our time step. As mentioned previously, in visual effects time often matters a lot. That’s why the derived time step often gets scaled up with a factor to minimize the amount of time steps we need.

$$\Delta t = k_{CFL} \cdot \frac{\Delta h}{\vec{u}_{\max}} \quad (5)$$

Another improvement was mentioned by Robert Bridson in [3]. His equation also accounts for body forces that act on the fluid and might increase the maximum velocity that is currently in the field.

$$\vec{u}_{\max} = \max(|\ \vec{u}\ |) + \sqrt{\Delta h\ |\ \vec{g}\ |} \quad (6)$$

5.1.2 The MAC-Grid. One important improvement in computational fluid dynamics was the so called MAC-Grid introduced by Harlow and Welch in [6]. It is a staggered grid. Staggered means that the different quantities are stored at distinct positions throughout the grid. As shown in figure 3, the pressure for example is stored in the center of the grid cell, also often called voxel. Only the normal components of the velocity are put at the corresponding face of the grid cell. This overall structure allows for an accurate central differences calculation but is not really useful for anything else. It complicates things quite well because you always have to perform some interpolation to even get the velocity value at a discrete grid point!

5.1.3 Operator Splitting. So now let’s look on how to perform the actual simulation. To integrate over the full Navier-Stokes equation or even the Euler equation would be way to hard and complicated. That’s why something called operator splitting is performed. We divide the original formula into separate parts and solve them one after each other individually.

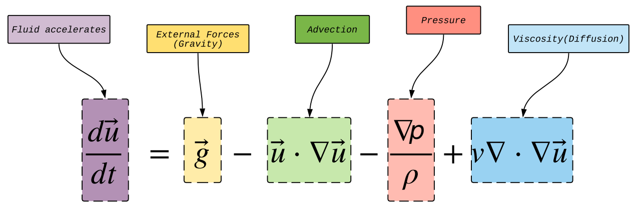
Fluid accelerates
External Forces (Gravity)
Advection
Pressure
Viscosity(Diffusion)
$\frac{d\vec{u}}{dt} = \vec{g} - \vec{u}\cdot\nabla\vec{u} - \frac{\nabla p}{\rho} + \nu\nabla\cdot\nabla\vec{u}$

Figure 4: Operator Splitting

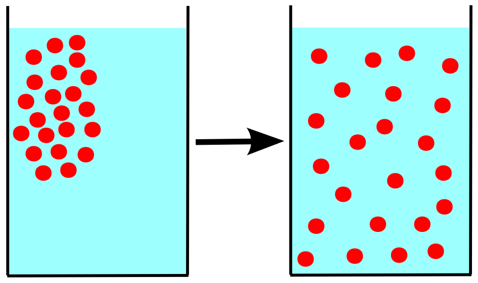

Figure 5: Diffusion [9]

```
dertermine $\Delta t$
$\vec{u}^A = \text{diffuse}(\vec{u}^n, \Delta t, \vec{u}^n)$
$\vec{u}^B = \text{advect}(\vec{u}^n, \Delta t, \vec{u}^A)$
$\vec{u}^C = \vec{u}^B + \Delta t \cdot \vec{g}$
$\vec{u}^{n+1} = \text{project}(\Delta t, \vec{u}^C)$
```

As one can see in the pseudo-code above, each step calculates the new values for the velocity field and are fed into the next step until we have the final result of the overall equation. Now let's have a look at the different steps.

5.1.4 Diffusion. Diffusion only applies in viscous fluids and describes the particle movement inside of the fluid due to viscosity. In general it tries to minimize the differences in velocity of nearby particles. Generally speaking it is driven by the gradient of concentration and can therefore also by applied to head diffusion for example.

5.1.5 Advection. A very important step in every fluid simulation is the advection step. In general, advection is the bulk motion of the substance and its properties. Often semi-Lagrangian advection is used. It's called that way, because we think of the process if there was a particle. Imagine we want to calculate some quantity for the next time step at position x_G. To do so, we put a fictional particle there and ask the question, where was the particle in the previous time step? So we first calculate the position where the particle would have been before and use the value at this position. Of course we will have to interpolate, because this point might not lie exactly at a grid point.

$$q_G^{n+1} = \text{interpolate}\left(q^n, \vec{x}_P\right) \tag{7}$$

To retrieve the previous location of this fictional particle we have to do some integration. The easiest method would be Forward Euler (8).

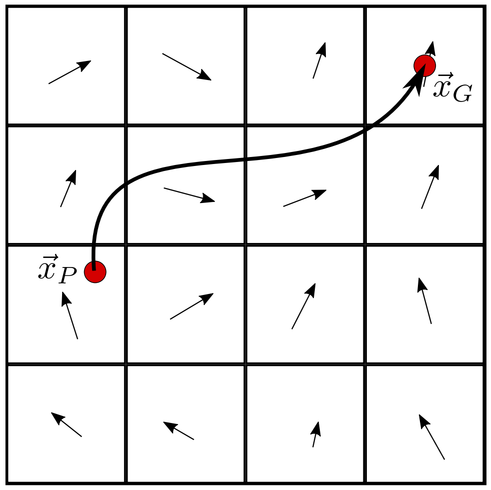

Figure 6: Semi-Lagrangian Advection

We simply look at the velocity at the current position and calculate the previous from there.

$$\vec{x}_P = \vec{x}_G - \Delta t \cdot \vec{u}\left(\vec{x}_P\right) \tag{8}$$

This method is not very robust and might later lead to artifacts and an inaccurate simulation. That is why often some more sophisticated methods like Runge-Kutta of higher order are used. Here we don't use one single velocity to estimate the previous location. Instead we go multiple smaller substeps and sample the velocity field at multiple points to retrieve a better estimation of the result.

5.1.6 Forces. In the next step we apply all our body forces to the velocity field. This often only boils down to gravity being the only force in the simulation. But with Newtons first law of motion we could convert any force that acts on the fluid into an acceleration value. At the end, all accelerations are applied to the velocity field.

$$\vec{a} = \frac{\vec{F}}{m} \tag{9}$$

5.1.7 Pressure Projection. Next comes the pressure projection. In this step, the new pressure values are calculated for every grid cell. Our new pressure values have to ensure that the incompressibility condition is met and the velocity field stays divergence free. It is called projection, because this is usually done with a system of linear equations. These are put into a matrix that forms a transformation. If we would apply this matrix over and over again, the result would not change anymore, because we are already divergence free. This is then called a projection. This step usually takes the most of our time in the simulation and should therefore be optimized a lot.

5.2 Particle Based Simulation

5.2.1 Smoothed Particle Hydrodynamics. In particle based simulations, the whole data is stored on particles. So each particle itself contains its values for velocity, pressure or maybe other quantities like temperature for example. One common method is the so called Smoothed Particle Hydrodynamics (SPH). This method is often used because it is easy to implement.
Because we only have a point cloud of particles, we always have

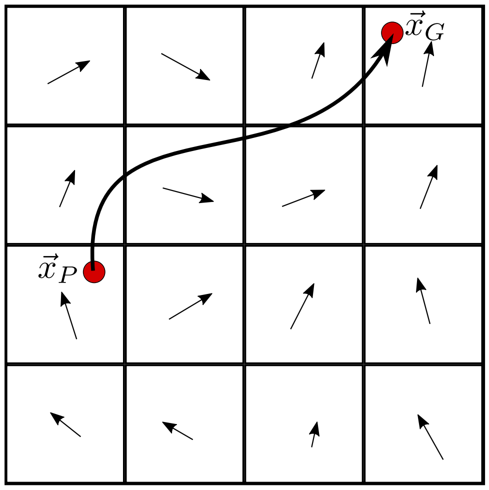

Figure 7: Particle Advection

to perform some sophisticated interpolation of these and need to implement acceleration structures for them. So accuracy might suffer from this. For accurate results a very high number of particles is needed. These often leads to the same performance as with grid based approaches.
Another disadvantage is that we don't want to render a point cloud in the final output most of the times. So we have to first perform some preparation for rendering. The simplest option is to map the particle data onto a discrete grid and perform volume rendering on this data. The next step would be to use this grid data to construct a mesh from it. This can be done with the marching cubes algorithm for example. Another possible method is the use of so called meta-balls.

5.2.2 Particle-in-Cell. Now let's have a look at a hybrid approach that should combine the advantages of both worlds. In section 5.1.5 we performed the so called semi-Lagrangian advection where we always calculated the quantities for a new discrete grid point. This might not be accurate enough. To perform a more precise advection, we use particles instead and get achieve a more accurate result, as depicted in figure 7.
The overall algorithm starts with storing all the quantities in particles. Then for each time step we transfer the quantities to the grid and integrate all non-advection terms, as with any other grid based method. After that we interpolate the results from the grid back to the particles. Then we perform the particle advection, moving them according to their new velocity values. Because we perform an interpolation in every time step there happens a lot of smoothing with can lead to unwanted results. The next method tackles this problem.

5.2.3 Fluid Implicit Particle. The Fluid Implicit Particle (FLIP) method is a variation of the previously discussed PIC method. In PIC we interpolate the grid data and store it in the particles. This leads to smoothing. In this method, only the change of the quantities on the grid are being interpolated and then added to the values stored in the particles. So interpolation only occurs once and does not accumulate and smooth the final value. This leads to virtually no numerical dissipation. The only downside is that this might lead to noise in the simulation. To circumvent this, we can introduce a factor that acts like a slider between pure PIC and pure FLIP results.

6 TETRAHEDRAL MESHES

6.1 The Method

Now we want to have look at an interesting paper [4]. It uses meshes for the simulation process. One cool thing about this method is that it supports an adaptive simulation. This is very important nowadays. Image a huge scene with a small boat in the middle of it. With a convectional uniform grid, you have to calculate everything and get the same resolution everywhere. But only the boat is our interesting part where things happen. They also use a staggered approach meaning that the pressure and other values are stored in the center of each tetrahedron. Only the normal components of the velocity are stored at each face.
So now have a look at the actual algorithm. In every step, the current surface is used to generate a tetrahedral mesh. This is done with the isosurface stuffing algorithm as we see later. This mesh is used to perform the actual fluid simulation via semi-Lagrangian contouring. After all fluid calculations have been done, we use the newly created surface for our next step. This has the advantage that here already is a surface that we could render.

6.2 Isosurface Stuffing

Now let's look at the isosurface stuffing algorithm [7]. Put simply, it is a very fast and numerically robust algorithm. This is very important because especially in finite element methods it is vital to have a good mesh. Having even a single badly shaped tetrahedron will cause immense errors. That's why this algorithm got introduced. On top of that it is easy to implement. Especially in fluid simulations when we have to perform computations over and over again, so it is very important to have a quick algorithm. This can be done with precomputed stencils like for example in the marching cubes algorithm. One major drawback of isosurface stuffing is, that it does not preserve sharp edges. This might be a huge problem in the area of engineering where you typically have hard-surface structures. For us in computational fluid dynamics this is no problem as we do not have any sharp edges in a fluid simulation.

Now let's have a look at an example. We use the graded version of the algorithm which uses an octree to help with the construction of a graded mesh. As one can see in the left part of figure 8, an octree gets built to assure finer resolution near the surface. Next we drop all the grid points that are too far away from the surface. Then we compute the intersection points of the edges that cross the isosurface. Finally we snap the outside lying points onto the surface and we are getting a graded tetrahedral mesh.

In figure 9 one can clearly see the graded internal structure of the generated mesh. The surface has smaller tetrahedrons than the inside of the mesh.

7 POWER PARTICLES

7.1 What are Power Particles?

Now we want to present another approach based on so called power particles. It combines the advantages of the previous mesh based

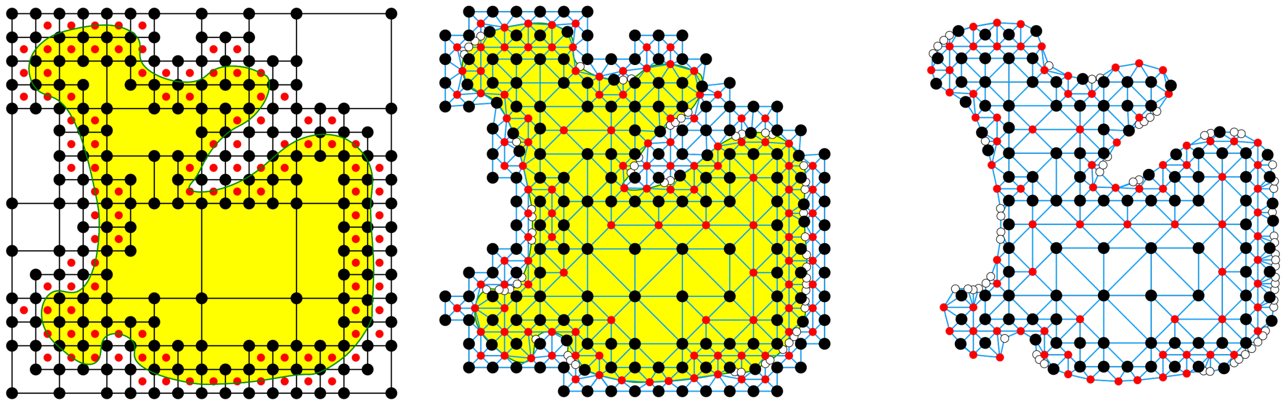

Figure 8: Isosurface Stuffing [7]

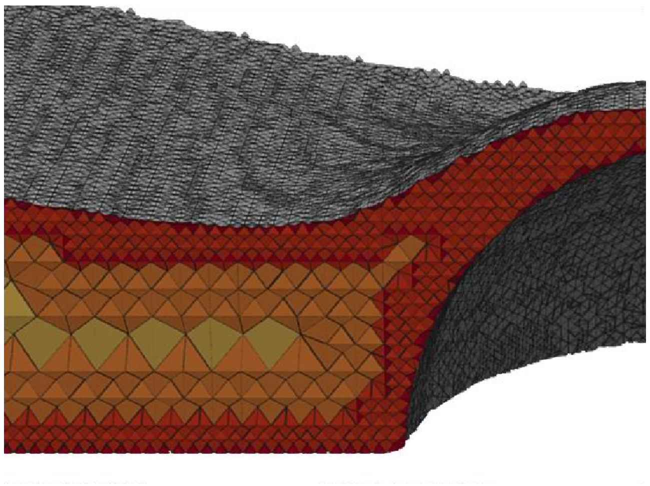

Figure 9: Example Mesh [4]

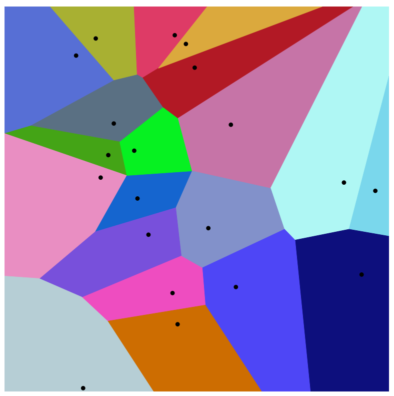

Figure 10: Voronoi Diagram

approach with a particle based simulation. So what are power particles? Some of you might already have heard of Voronoi diagrams. Given a set of points in a two dimensional plane, we divide the

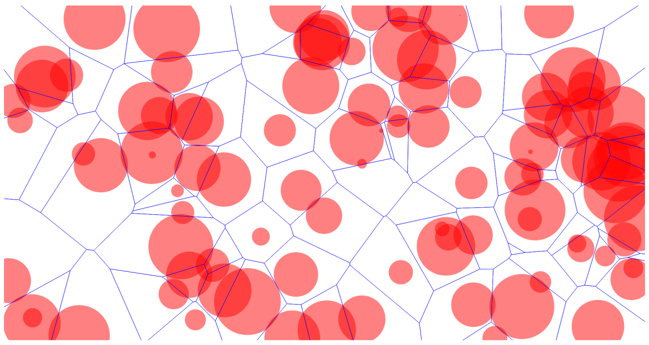

Figure 11: Power Diagram [5]

plane into areas such that for every point in that area, the corresponding input point is the closest one. So we partition the plane according to distances. Power diagrams extend this a little bit by adding a weight value to each of the input points. As depicted in figure 11 this weights can be represented as circles. In addition to that for calculating the partitioning, the square of the distance and the weights are used as one can see in the following formula.

$$v_i = \left\{x \in \Omega \mid d(x, q_i)^2 - w_i \leq d(x, q_j)^2 - w_j, \forall j\right\} \tag{10}$$

So what are power particles then? Instead of two dimensional power diagrams, we now get volumetric parcels in three dimensions. Each particle, now called cite, gets a weight that controls the volume of the power particle. These are then further used for the new approach.

7.2 The Method

Now let's look at the basic steps in the algorithm. We first start with n cells of volume V and mass m. In every simulation step we proceed as in most other methods. We first update the velocity values, compute pressure and apply the pressure forces to the velocities of each particle. Then we advect the particles through space. After all these steps, we adjust our volumes to ensure incompressibility of our fluid. This can be done by recalculating the weights of each particle. One interesting thing about this algorithm is, that it can

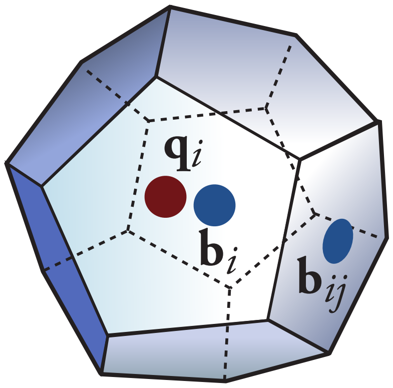

Figure 12: Power Particle

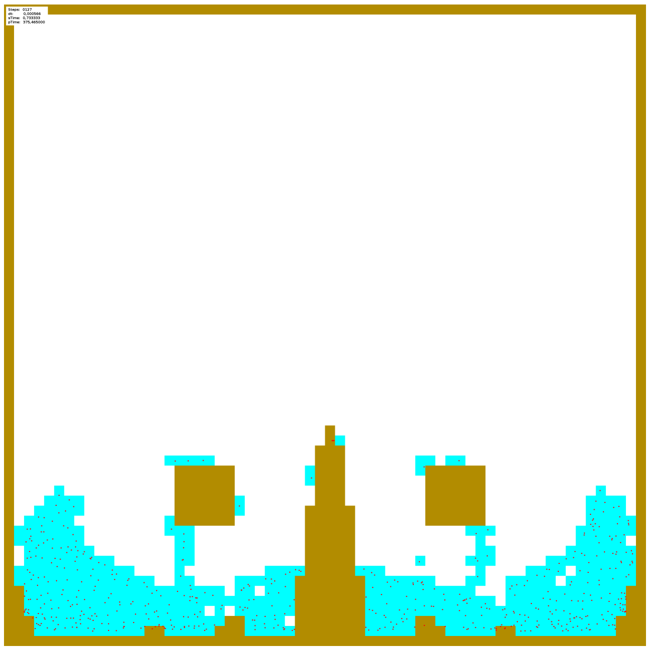

Figure 13: Simple Fluid Simulation

easily be adjusted to compressible flow. This only requires some changes in the pressure projection step and the need for updating the volume of each particle. These new volumes can then be again adjusted via the weights. So what are the advantages of this approach? Mainly because we are dealing with volumetric particles, the algorithm achieves accurate pressure projection. On top of that, the method can be adapted to also support compressible fluids. And one very cool addition is the support of multi-phase flows. This means the algorithm can handle liquids with different densities in one simulation. Some oil on top of water would be an excellent example for this.

8 OWN RESEARCH

In the course of our research we also tried to develop our own very simple fluid solver. In its current state it is only 2D, as Robert Bridson suggests in his book [3]. This simplifies the development and makes debugging easier if something is not working. This would be much more difficult in three dimensions.
We also want to point out, that these red particles that you can see in figure 13 are not FLIP particles. Those are simply marker particles to indicate whether a grid cell contains any fluid or not. This is needed for the pressure projection step to set the values in the corresponding equations correctly.
We implemented this in Java because this is the language we are most familiar with. We knew that the pressure projection step is the most computational intensive task, because of the huge matrix and the way it gets solved. But the time it takes to simulate only one second of simulation is currently far too bad. This might be due to the use of the simple conjugate gradient method to solve for pressure as well as the use of Java. But it is more likely that the way we implemented it is not that efficient as well.
In either way we are going to continue our work on it and try to implement new features. Our next steps are going to concentrate on level sets and maybe the extension to FLIP. One of the bigger steps would be to switch entirely to C/C++ for development to achieve faster simulation times and the compatibility to state of the art software. Also we would like to see if we could use an octree data-structure to allow us adaptive simulations which would be very nice.

REFERENCES

[1] Autodesk. since 1982. . Autodesk. https://www.autodesk.com
[2] C. Braley and A. Sandu. 2009. Fluid Simulation For Computer Graphics: A Tutorial in Grid Based and Particle Based Methods.
[3] Robert Bridson. 2015. *Fluid Simulation for Computer Graphics, Second Edition.* Taylor & Francis. https://books.google.at/books?id=7MySoAEACAAJ
[4] Nuttapong Chentanez, Bryan E. Feldman, François Labelle, James F. O'Brien, and Jonathan R. Shewchuk. 2007. Liquid Simulation on Lattice-Based Tetrahedral Meshes. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Diego, California) *(SCA '07)*. Eurographics Association, Goslar, DEU, 219–228.
[5] Jason Davies. 2020. Power Diagram. https://www.jasondavies.com/power-diagram/ [Online; accessed 24-November-2020].
[6] Francis H. Harlow and J. Eddie Welch. 1965. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. *Physics of Fluids* (1965). https://doi.org/10.1063/1.1761178
[7] François Labelle and Jonathan Richard Shewchuk. 2007. Isosurface Stuffing: Fast Tetrahedral Meshes with Good Dihedral Angles. *ACM Trans. Graph.* 26, 3 (July 2007), 57–es. https://doi.org/10.1145/1276377.1276448
[8] Industrial Light & Magic. since 1975. . Industrial Light & Magic. https://www.ilm.com
[9] Wikipedia contributors. 2020. Diffusion — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Diffusion&oldid=988462292 [Online; accessed 24-November-2020].