2IMV15 Simulation in Computer Graphics - Rigid Bodies and Fluid

Boris Rokanov (1396331), Georgi Kostov (1396773), Tar van Krieken (1244433)

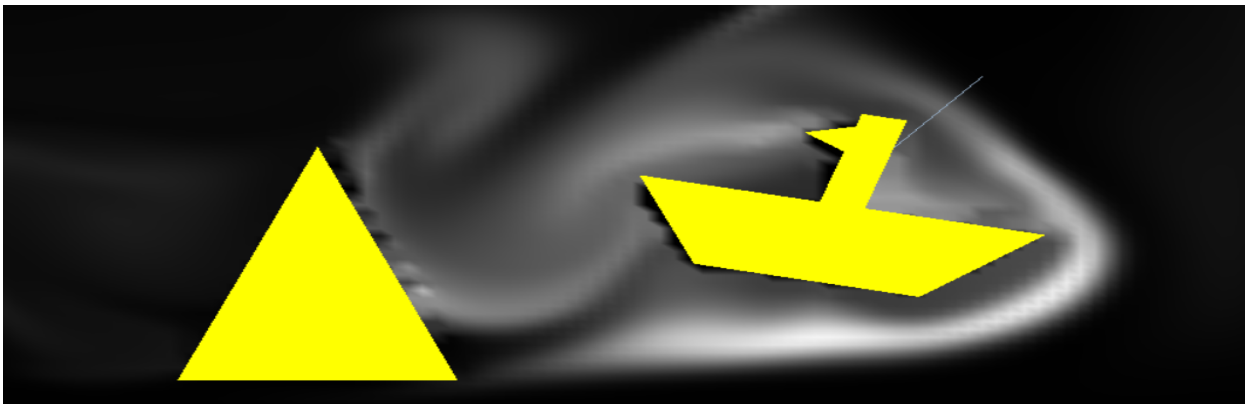


Fig. 1: An example scene in our fluid simulation generated using rigid bodies.

Abstract— This paper extends the fluid simulation suggested by Jos Stam [3]. We show how we implemented vorticity confinement, fixed and moving solid bodies, rigid bodies, particles and two way coupling between the different types of bodies and the fluid.

Index Terms—Fluid Simulation, Particle Systems, Vorticity, Rigid Bodies, Solid Objects, Collision

1 Introduction

Fluid mechanics is a field of physics concerned with the movement, flow and properties of fluids. It has a variety of use cases in many scientific areas, such as hydraulics, biology and others. In computer science particularly, fluid mechanics is often used to build simulations that visualize fluid interactions in a quick, effective and realistic way.

Video effects and games can rely on those fluid simulations to increase realism as discussed in the article by Jos Stam [3]. The author suggest a way to approximate the diffusion and advection, which allows for a fast and life-like implementation. The diffusion function defines the way density is spread from higher to lower dense regions. Similarly, the advection shows how the flow and currents of the fluid affect the velocity in a specific region. Additionally, by using a projection function which forces the fluid to conserve its mass, the simulation can display complex flows during runtime.

Our project extends the C++ code suggested by the aforementioned paper by adding additional features. In particular, we explain how we created vorticity confinement, fixed and moving solid objects, rigid bodies and particles in the fluid simulation. At the end, we discuss the limitations of our approach and suggest possible improvements.

2 Vorticity Confinement

Vorticity confinement is a mathematical technique used to increase the live duration and realism of a smoke simulation. Essentially, it increases the forces that create vortex in the fluid, thus giving small vortex-like movement even to stale parts of the simulation.

As discussed in another one of Jos Stam's papers [2], the first step to creating the vorticity is to calculate the curl ω for all cells of the simulation:

$$\omega = \nabla \times u$$

- *Boris Rokanov. E-mail: b.m.rokanov@student.tue.nl.*
- *Georgi Kostov. E-mail: g.t.kostov@student.tue.nl.*
- *Tar van Krieken. E-mail: t.m.k.v.krieken@student.tue.nl.*

This formula represents the forces that try to spin the flow of the fluid. Next, we calculate the normalized vorticity location vector N for each cell the following way:

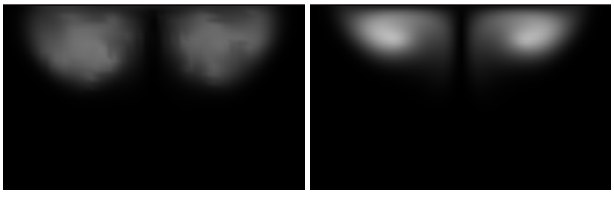
$$N = \frac{\eta}{|\eta|}, \text{ where } \eta = \nabla|\omega|$$

This vector shows how cells with lower vorticity point to cells with higher vorticity. Therefore, in the end we can define our whole vorticity confinement function per cell as:

$$vort\_conf = \varepsilon \cdot p(N \times \omega)$$

where $\varepsilon > 0$ denotes a scaling factor, while p denotes the projection function that ensures the mass of the fluid is preserved.

In the following figures you can see the difference when the vorticity confinement is applied (Figure 2a) agains when is not (Figure 2b). These two pictures are captured at the exact same moment of time. When the vorticity confinement is applied, we can clearly spot the vortexes that are appearing in the fluid. This brings more realistic behaviour to the smoke in our simulation.



(a) Vorticity confinement applied (b) No vorticity confinement applied

In Appendix A, we showed the Vorticity Confinement effect captured at different timesteps of the simulation.

3 Fixed Objects

In order to allow for fixed objects, a boolean boundary array was defined. This array marks each cell of the fluid simulation as either a boundary cell or not. Any boundary between a boundary and non-boundary cell is then interpreted as a boundary, as illustrated in Figure 3. This boundary is then considered at various stages of the fluid simulation process.

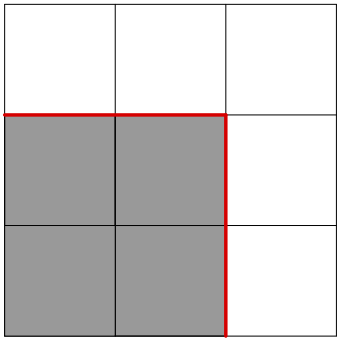


Fig. 3: Boundary interpretation

When advection is performed, any cell that is a boundary is skipped. Additionally for any non-boundary cell a check is executed to determine whether the cell at the sample point is a boundary. In case it is a boundary, the intersection point is approximated by performing a binary search. This process works as follows. Given the start point s and the sample point p, we define a test point t starting at $t = (p+s)/2$ and a direction $d = p - s$. We then iterate $i = 2$ up til an arbitrary maximum (e.g. 8) which determines the precision. In each step we check whether t is in a boundary cell. If it is in a boundary, we perform the following update: $t := t - d/2^i$. When it is is not a boundary, we perform this update instead: $t := t + d/2^i$. This process is illustrated in Figure 4. Under the assumption that the distance was less than 1 cell, this process allows us to get a good approximation of the intersection position. Finally this position is used instead of the original sample position.

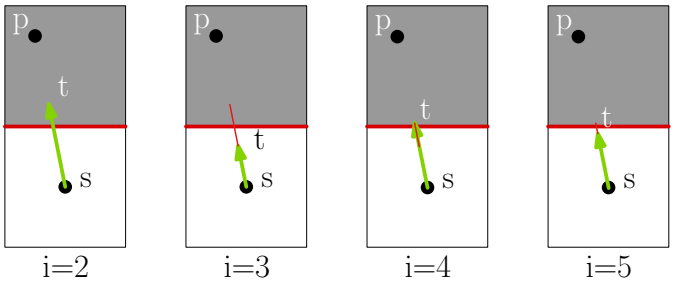


Fig. 4: Boundary path clipping, before each iteration i

In the diffusion step the values of the boundary cells are ignored as illustrated in Figure 5. This means that if a cell is adjacent to 1 boundary cell, this value is not considered when averaging, and the formulas are updated accordingly. For instance when updating the density ρ, if there is a wall to the right, we would get the following formula:

$$\rho_{i,j} := \rho_{i,j} + a \cdot (\rho_{i-1,j} + \rho_{i,j-1} + \rho_{i,j+1} - 3 \cdot \rho_{i,j})$$

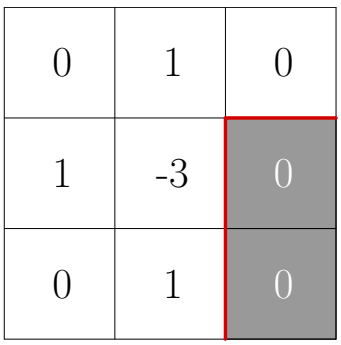
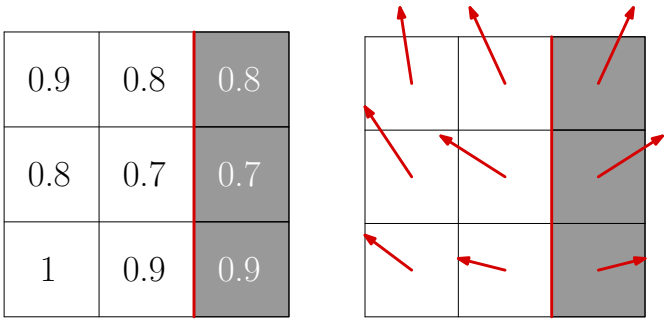


Fig. 5: Boundary diffusion

Finally we have to ensure that the boundary conditions are in these new boundary cells. This means that the velocity in the direction of the normal at the boundary should be 0 and the density at the boundary should also be equal to the density within the non-boundary cell. To achieve this, the velocity of a neighboring cell is copied and mirrored with respect to the normal, such that when averaging the velocity is 0 in the normal direction. Similarly for the density, we can simply copy the value without any alterations. This technique only works when a boundary cell is only involved in 1 boundary however. Figure 6 illustrates how these values are copied to boundary. In the other situations we average the values from all non-boundary neighbors, but here some accuracy is lost.



(a) Density copying (b) Velocity mirroring

Fig. 6: Boundary conditions realization

4 Moving Solid Bodies

For moving solid bodies we simply define each body by a shape and its location. Then in each iteration of the simulation all boundary grid cells are first cleared, and then all solid bodies are projected onto this grid. This is done by iterating over each cell that falls into the axis aligned boundary box of the shape at the given solid body's location, and checking whether its center point is contained in the shape. If it's contained in the shape, we can mark it as a boundary on our grid.

Next we introduce a new grid: the boundary velocity grid bv. This grid will store two-dimensional velocities for each cell on the grid. This grid represents the velocity that a boundary cell has. When trying to ensure the boundary conditions, this grid is now also considered. Consider for instance the situation as illustrated in Figure 7, where we have a boundary cell at position (i, j) with a non-boundary cell above it such that a horizontal boundary is formed. Then we can use the following assignment for the velocity v^y on the y-axis for the boundary cell:

$$v^y_{i,j} = -v^y_{i,j+1} + 2 \cdot bv^y_{i,j}$$

This assignment ensures that the the speed at the boundary will indeed be the speed requested by the boundary velocity grid, with respect to the boundary's normal vector.

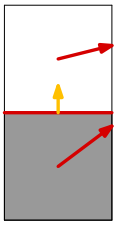


Fig. 7: Boundary velocity

Now when projecting the solid bodies to the boundary grid, we can also project their velocity to the boundary velocity grid. This results in the fluid being moved in a way that looks quite convincing. Additionally one may choose to only project the movement velocity projected onto the normal of the boundary of the solid shape as illustrated in Figure

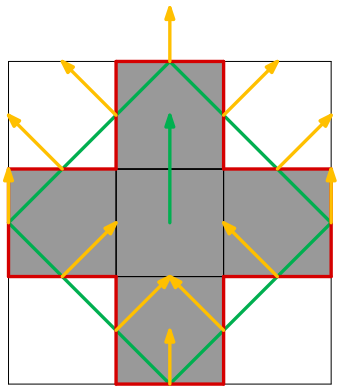


Fig. 8: Boundary velocity considering normals

8. This way fluid will not stick to the side of a moving solid body. Instead of fully removing the projected velocity perpendicular to the boundary's normal, we can also take a proportion of it. This way we can model an amount of drag that the surface has. Note that when multiple segments of a shape go through the same boundary, the boundary will average their velocity.

5 Moving rigid bodies

For the moving rigid bodies the particle system was used as a foundation. Additionally inertia, angular velocity, and angle scalars were added. In three-dimensional space a matrix is required to accurately represent the inertia, since there are 3 axis of rotation that influence each other. However when reducing to two-dimensional space, only a single axis of rotation remains. Because of this, it suffices to only use scalars to represent the inertia, angular velocity, and angle [1].

A helper function is used to calculate the center of mass in order to shift shapes such that they rotate around their center of mass, as well as the inertia given a desired mass. This function takes a resolution as input, and divides the boundary box of the shape into cells of this size. Each cell is sampled to test whether it is in the shape. If it is in the shape, we use it to calculate the average, and we also add it to the inertia. This is done in such a way that the grid approximation of the shape will be in accordance with the regular inertia formula:

$$i = \sum_n m_n \cdot r_n^2$$

We perform steps in a similar way as was done in the particle simulation. We chose the mid-point method for this. We only had to accommodate the new degrees of freedom, and everything else works identical to what was done in the particle simulation. The constraints were however fully removed, since adding constraints proved to difficult when dealing with impulse based collisions.

6 Colliding contacts

All shapes in the simulation are defined as simple polygons where all points are defined in counterclockwise order. To determine whether two shapes A and B are colliding at a given position, a naive check is performed that requires $\mathcal{O}(n \cdot m)$ computation time, where n and m are the number of points in A and B respectively. For either shape we iterate through its points, and for each point p check whether it is contained in the other shape. This is done by counting how many segments are intersected by a horizontal half-line ending at p as illustrated in Figure 9. If this number is odd, p must lie within the other shape. This technique will not catch all possible collision. Figure 10 illustrates an example of a collision that would not be detected. However if the time step is small enough the intermediate state before this type of collision will be detected properly. After we determined what point p intersected the shape, we perform an intersection test between the line formed by p and its position in the previous timestep p' together with all segments of the other shape. This way we can determine at what

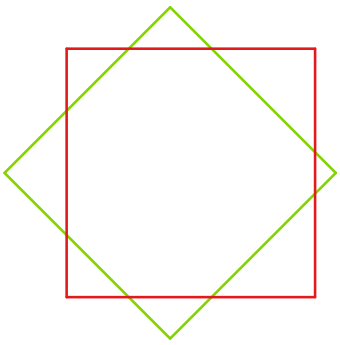


Fig. 9: Point intersection check

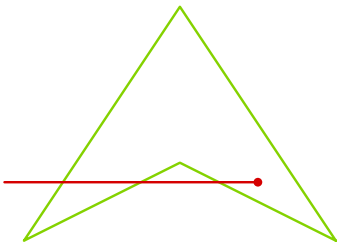


Fig. 10: Non-detected collision between polygons

point of the other shape collision occurred, and what the corresponding normal direction is.

The simulation iterates through all possible pairs of rigid bodies. For each pair a coarse check is first performed by checking whether the axis aligned bounding boxes of the shapes overlap. For any two rigid bodies whose bounding boxes overlap, the more precise check is performed on their shapes as described before.

We use a binary search similar to the one described for boundary checking in the fluid simulation to get an approximation of the furthest sub-timestep we can perform before collision occurred. We also store the collision data of the smallest sub-timestep where collision was still detected. This data is then used to adjust the linear and angular velocity of both bodies. Consider a collision between rigid bodies A and B. Using the collision algorithm describe before we can find corresponding collision points r_A and r_B relative to the rigid bodies' frame of reference as the normal direction n. From the r_A we can derive collision position $p_A(t)$ and velocity $\dot{p}_A(t)$ at a given time t. For this the rigid body's position x, velocity $\dot{x}$, angle a and angular velocity $\dot{r}$ are used to derive the following definitions using rotation matrix $R(\theta)$ for a given angle θ:

$$p_A(t) = R(a(t)) \cdot r_A + x$$
$$\dot{p}_A(t) = \dot{a}(t) \cdot \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} R(a(t)) + \dot{x}$$

Now let p_A^+ and p_B^+ specify the points after collision correction and p_A^- and p_B^- the same points before collision correction. Given a bounce factor b between 0 and 1, we make sure that the following equation holds:

$$n \cdot (\dot{p}_A^+ - \dot{p}_B^+) = -b \cdot (n \cdot (\dot{p}_A^- - \dot{p}_B^-))$$

This requires solving of a linear equation. The exact formula is not very interesting, and thus will not be described here, but all involved steps can be found in the code.

Finally after reverting the simulation to the last timestep before collision occurred and adapting the velocities accordingly, the simulation continues to finish the remainder of the timestep. This happens iteratively, since another collision may occur when trying to perform the remainder of the target timestep size.

We additional support collision-ignore groups that can be added to rigid bodies such that collisions with other rigid bodies in this group

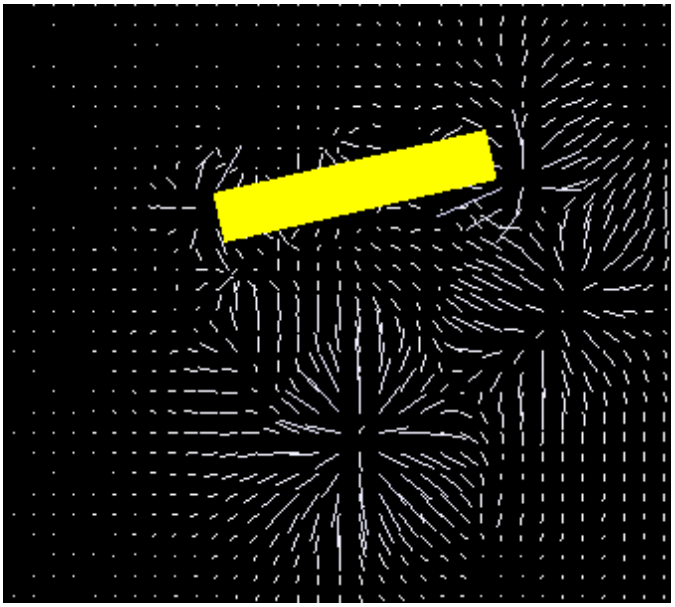


Fig. 11: The pressure field after an external force was applied.

are ignored. This way certain bodies can freely pass through each other, which is particularly useful when trying to add joints between bodies.

7 Two way coupling

To achieve two-way coupling, a pressure field is extracted from the fluid-simulation. This is done by taking the pressure fields computed in a single step by Stam's algorithm and summing them into single output pressure field. Stam's algorithm usually only uses the computed pressure field to adjust the velocity field, such that the fluid acts in an incompressible way. Having this field to know where pressure would have been applied is however useful for us in order to determine the interaction with the rigid bodies. Figure 11 visualizes what the pressure field may look like.

Just like for moving bodies, the rigid bodies project their shape onto the boundary grid to form boundaries that approximate their shape. Then for each rigid body, its boundary is approximated. This is done by taking a step size s, and sampling all points that are distance s apart on the boundary. For each point p we also extract the boundary's normal direction n. We then perform the velocity transfer as described in section 4, but for the velocity of point p rather than the velocity of the rigid body. We may however get multiple samples per grid cell, which will all get summed up in the boundary velocity grid. Hence we also keep track of the number of samples per cell in order to compute the average for each cell when all bodies are projected.

We also use point p and its normal n to transfer force to the rigid body. This uses the relation between pressure and force:

$$pressure = force/area$$

Since we are in two-dimensional space, we use the step size s instead of the area. The pressure field can now be used to derive how much force to apply at point p. For this we also have to consider our normal n in order to only apply the proportion of the force exerted in this direction. Once again however, we can use a drag scalar to also transfer some of the force exerted in the perpendicular direction. The force applied at point p will also partially convert into torque, depending on the direction of the force and the location of p relative to the origin of the body. Figure 12 shows how the samples are used to transfer velocity from the rigid body, and force to the rigid body.

8 Particles

Particles are simply treated as rigid bodies with a single point as its shape. This single point can simply be considered a degenerate polygon

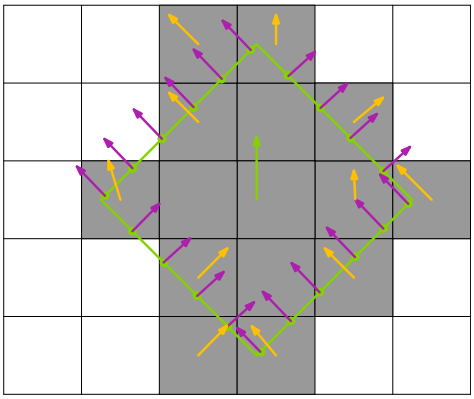


Fig. 12: Rigid body boundary sampling

with an area of size 0, and is treated as such in our implementation. This way we also get collisions between particles and other rigid bodies for free. This however does pose a little problem when it comes to our two-way coupling, since these point polygons do not really have normal directions when their boundary is sampled. To solve this, particles are treated as circles when it comes to sampling, and a sample is returned in 8 uniformly distributed directions. Each of these samples provides the same location, but specifies another normal direction.

Springs forces were adapted so they work on rigid bodies in general, allowing you to even specify the attachment point relative to the body's origin. This feature is not used for the particles since no force should be transformed into torque here, but this behavior can be seen when dragging rigid bodies using the mouse (which makes use of a special version of the spring force). Using these springs, a cloth is generated. This cloth automatically has two-way coupling with the fluid, since particles are simply implemented as rigid bodies, for which we already established the two-way coupling.

9 Results and evaluation

In order to evaluate our implementation, we performed multiple experiments, some of which can be seen in Appendix B. We conclude that our approach offers a lot of freedom in the types of rigid bodies that can be modelled, as shown in Figure 13. The spring forces and collision ignore groups can add to this by constraining bodies together to emulate joints. The interaction with the fluid simulation also behaves rather well and looks convincing.We even found that the interaction with the fluid had the expected result when modelling a dart. This dart was made to be long with a lot of surface area, and has the center of mass in the front. Then due to the drag with the fluid that it moves through, it tends to point face forward into the direction it is moving.

Fig. 13: Rigid body with a complex shape

However, we also found a number of issues with the approach. First off, it does not scale particularly well. The collision detection between two shapes is rather inefficient, and the boundary boxes are also checked for any pair of shapes. The simulation also is able to get stuck in certain situations. It will get stuck when even the smallest fraction of a timestep results in a collision. The handling of the collisions should ensure this

does not happen, but we think that resolving one collision may cause another in a way that a feedback loop is formed. We also believe that the density is slowly dissipating over time. This is likely due to density not fully being moved by the applied boundary velocity before being set to 0 when the rigid body occupies a new cell. And lastly we noticed that the interaction with the fluid does not behave as expected when it comes to the mass of the object. It appears that when gravity is applied, bodies with larger masses slow down due to interaction with the fluid faster than those with small masses. We would have expected the opposite to happen, because bodies with large masses should be able to overcome drag more easily. This behavior may be caused by some sort of feedback loop between the rigid body influencing the speed of the fluid, and that same fluid exerting a force on the rigid body.

10 Conclusion and future work

In conclusion, our approach for simulating rigid bodies in fluids is not perfect, but does create decently convincing results. It is rather flexible and can be used to create simple scenes, but it should not be relied on when interested in physically accurate interactions.

In the future we would like to investigate what causes the density to dissipate over time, and possibly try resolving this. Similarly it would be worthwhile to analyze what causes the unexpected behavior with respect to the masses. The realism of the simulation would likely benefit a lot by finding out the exact cause of this and resolving it if possible. Lastly we would like to try an alternative approach for collision handling. The current approach is not very efficient, and sometimes even results in the simulation fully coming to a halt. A common approach for simulations appears to be making use of the Separating Axis Theorem. When this theorem is used, collision can be resolved without rolling back the simulation. This could result in less realistic simulations and even jitter, but this still seems preferable over the simulation fully halting.

11 Contributions

Student	**Tasks**	Hours
Boris Rokanov	Implemented Vorticity Confinement Implemented solid boundaries Generated demo scenes for showcases Added voiceover in the demo	62
Georgi Kostov	Helped with vorticity confinement Worked on moving solid bodies Generated demo scenes for showcases Edited the demo video	60
Tar van Krieken	Helped with fluid boundaries Implemented colliding rigid bodies Came up with and implemented the two-way coupling approach	70

Table 1: Personal contribution.

References

[1] M. J. Baker. Physics - dynamics rotation - inertia tensor. https://www.euclideanspace.com/physics/dynamics/inertia/rotation/index.htm. Accessed: 2022-06-27.

[2] R. Fedkiw, J. Stam, and H. W. Jensen. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 15–22, 2001.

[3] J. Stam. Real-time fluid dynamics for games. In *Proceedings of the game developer conference*, volume 18, page 25, 2003.

APPENDIX A VORTICITY CONFINEMENT DURING TIME STEPS

The following Figures clearly show the effect of the Vorticity confinement on the fluids during differnt time steps.

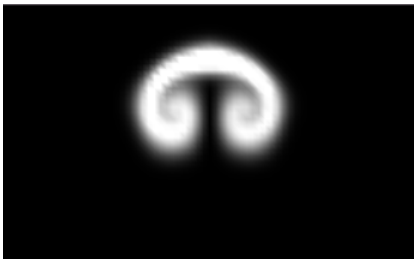


Fig. 14: With Vorticity Confinement at time step 1

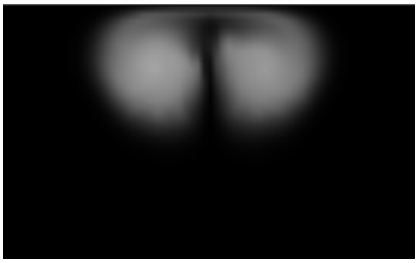


Fig. 15: With Vorticity Confinement at time step 2

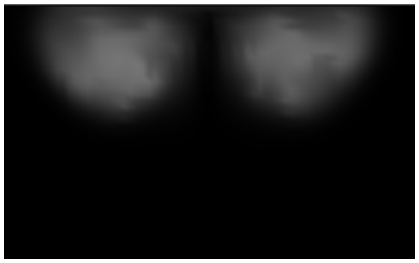


Fig. 16: With Vorticity Confinement at time step 1

In the three Figures below we can see the baseline implementation of the fluid. Note that these pictures are taken at the same timestep respectively to the ones above them.

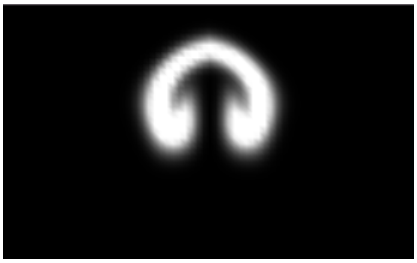


Fig. 17: Without Vorticity Confinement at time step 1

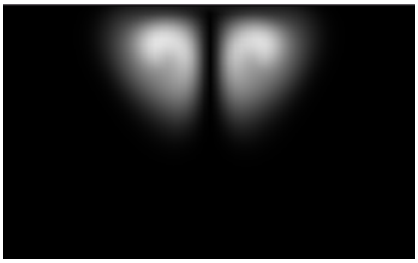


Fig. 18: Withoutout Vorticity Confinement at time step 2

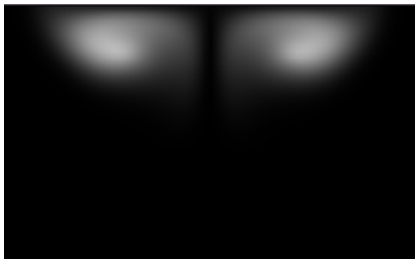


Fig. 19: Without Vorticity Confinement at time step 1

APPENDIX B EXAMPLE SCENES

Here we showcase some of the more interesting demo scenes we created using our project. Note that the green points in some of the Figures show collision points between bodies and/or particles.

Fig. 20: A car model made from solid objects. Note that smoke does not pass trough the car

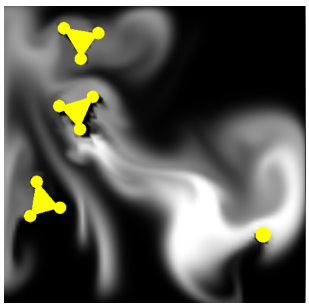


Fig. 21: Rigid bodies and fluid interaction

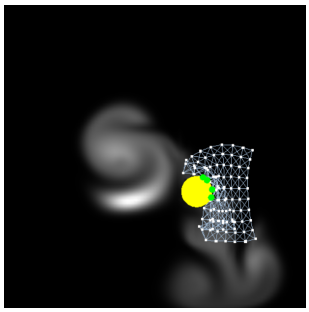


Fig. 22: Rigid bodies, cloth and fluid interaction

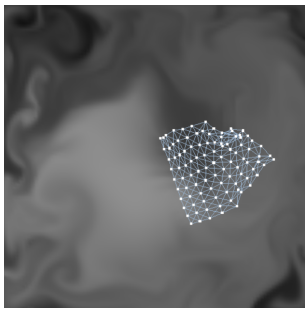


Fig. 23: Particle cloth simulation and fluid interaction

The next three Figures show collision of rigid bodies with how rigid bodies that have the same mass:

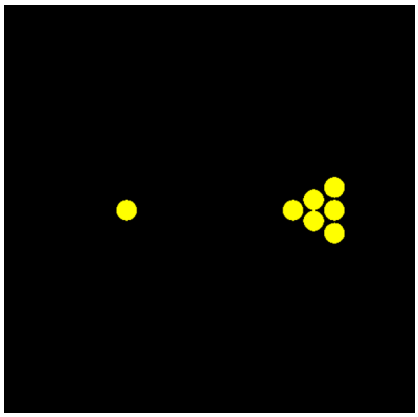


Fig. 24: Initial position of the rigid bodies

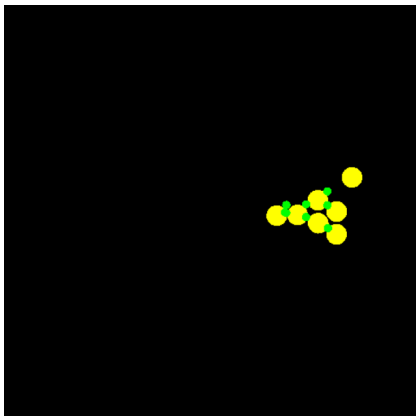


Fig. 25: Collision between the rigid bodies

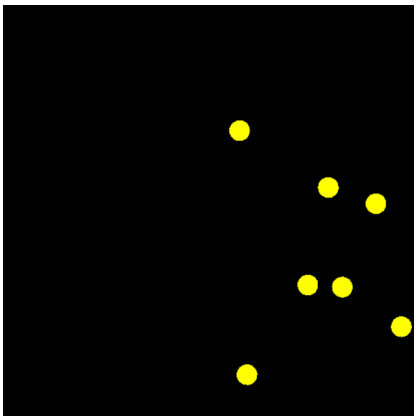


Fig. 26: Final position of the rigid bodies