INF585 - Computer Animation - Final Project

Marie Audouard

***Abstract*—In this project, I implement a Particle-In-Cell model to represent the flow of sand, based on the research paper [1].**

INTRODUCTION

In this project, I wanted to reimplement the paper [1]. However, as I worked on it, I realized the complexity of the task, and decided to only implement a Particle-In-Cell (PIC) model. Once this was done, I decided to also implement the surface reconstruction that is mentioned at the end of the paper. The final code is available here.

I. EXPLANATION OF THE PIC ALGORITHM

A. Theory

The idea of the PIC algorithm is to model fluids by mixing the advantages of a particular approach and of a grid-based approach. The outline of the algorithm is the following:

1) Calculate the speed on the grid by interpolation, taking into account the nearby particles;
2) Update those speeds to account for gravity (and any other forces you might want to add);
3) Make the grid incompressible by using a Gauss-Seidel approach (iterating several times over the grid to enforce having $div(\vec{v}) = 0$);
4) Update the speed of the particles by interpolating the speed of the cell in which it is;
5) Move the particles along that speed by integrating discreetly: $p_{new} = p_{old} + dt \times \vec{v}$;
6) Finally, ensure that no particle is going through walls.

Those steps can then be started again for the next time step.

B. Implementation details

For all the implementation, I used the class [2] of the course that this project is a part of, and the website [3].

For the implementation, I decided to use a Marker And Cell (MAC) grid, which means that in the middle of each face of the grid is stored the normal speed. In particular, for the (i, j, k) cell of the grid, we have that $v_x(i, j, k)$ is the inwards speed of the left face of the grid, and symmetrically along y and z. This is what this grid looks like in 2D:

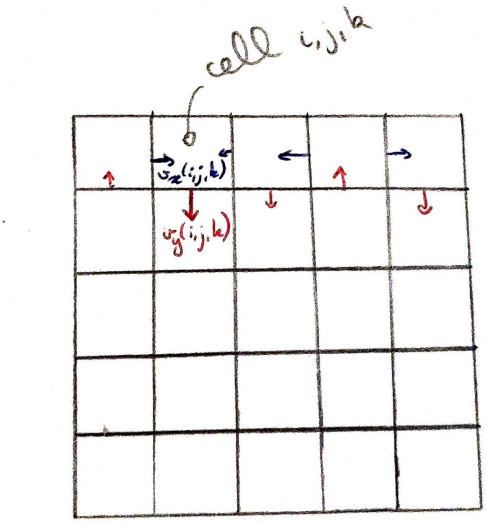

Fig. 1. MAC grid

Then, with this, I was able to calculate the divergence in each cell, the following figure describe the calculations in 2D:

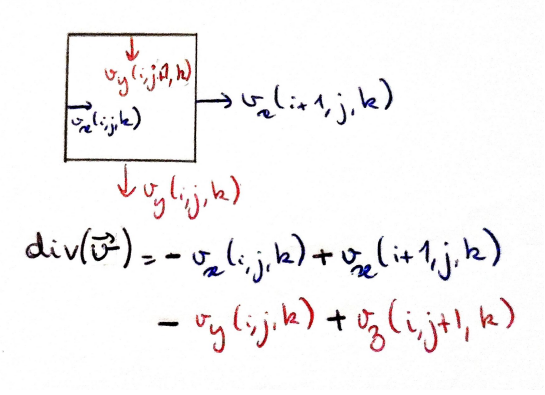

Fig. 2. Calculation of the divergence for one div cell

In 3D, another symmetric term is added for v_z. It is important to note that I only take into account

the neighbors that exist, and the ones where i (resp. j or k) is strictly positive because the speed is null inside the ground and walls.
Then, I set the divergence to zero by modifying the speeds: $v_x(i+1,j,k) = v_x(i+1,j,k) - div(\vec{v})/nb\_neighbors$ (resp. adding the div term for the (i,j,k) terms), for every one of the neighbors. I do this several times for all cells as per the Gauss-Seidel method, in order to have the divergence be zero in every cell at the end.

Then, my first approach of the PIC was working. However, I had a lot of particles that superposed, since I only treated the collisions with the walls and ground.
To attempt to solve this problem, I tried to impact the divergence of the cells: I calculated the particle density in the cell, and I updated the divergence as: $div(\vec{V})_{artificial} = div(\vec{v}) - \alpha(\rho - \rho_0)$, with ρ_0 the rest density of a cell and α a stiffness coefficient. Even if it improved slightly the result, it did not bring a major change (see section III).

To have a final display with fewer holes, I used a bigger radius for display than for the calculations.

II. Surface reconstruction

For the surface reconstruction mentioned in [1], I first had to calculate the implicit function. The expression given by the authors of the paper is:

$$\phi(x) = |x - \bar{x}| - \bar{r}$$

with $\bar{x}$ a weighted average of the positions of the particles that are found within a sphere of radius R of x, and $\bar{r}$ their weighted radii. Since my particles all have the same radius, we have an immediate simplification: $\bar{r} = r$. Then, for $\bar{x}$, with the definition of the paper, I have:

$$\bar{x} = \sum_{x_i \text{ such that } \|x_i - x\| < R} w_i x_i$$

with

$$w_i = \frac{k(\|x_i - x\|/R)}{\sum_j k(\|x_j - x\|/R)}$$

and k is the following kernel: $k(s) = \max(0, (1-s^2)^3)$.

Then, in order to display the reconstructed surface, I used marching cubes. To implement them, I adapted the code from [4].

III. Results

This is the final result of my program when using surface reconstruction:

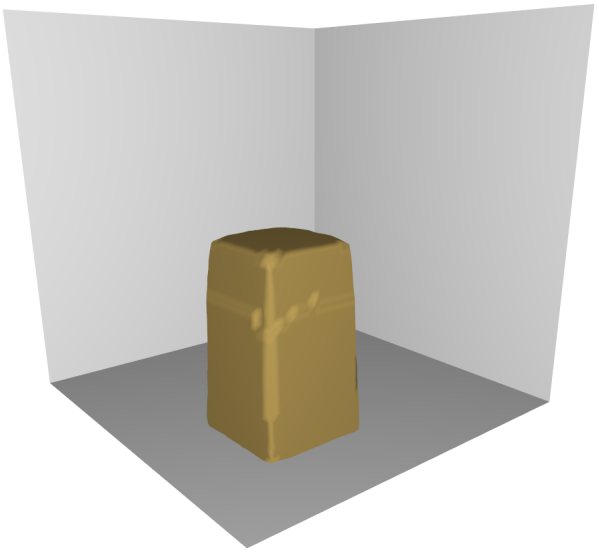

Fig. 3. Start of the animation

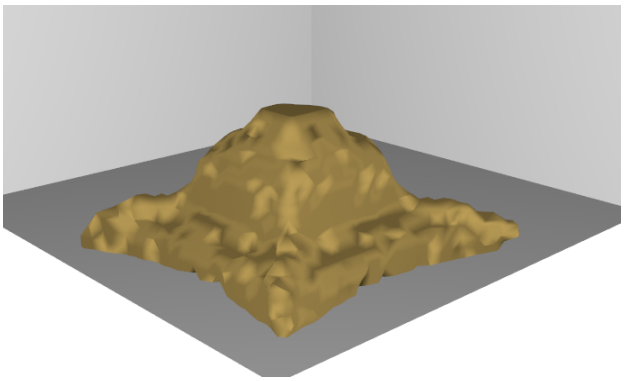

Fig. 4. Middle of the animation

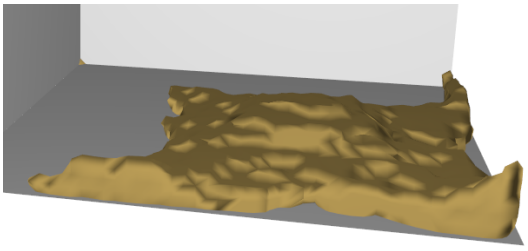

Fig. 5. End of the animation

We can notice that there are some strange artifacts at the angles, the particles seem not to be going down with gravity. This problem is even more visible at the end of the animation, when the particles hit the walls.

And this is the look with the particles:

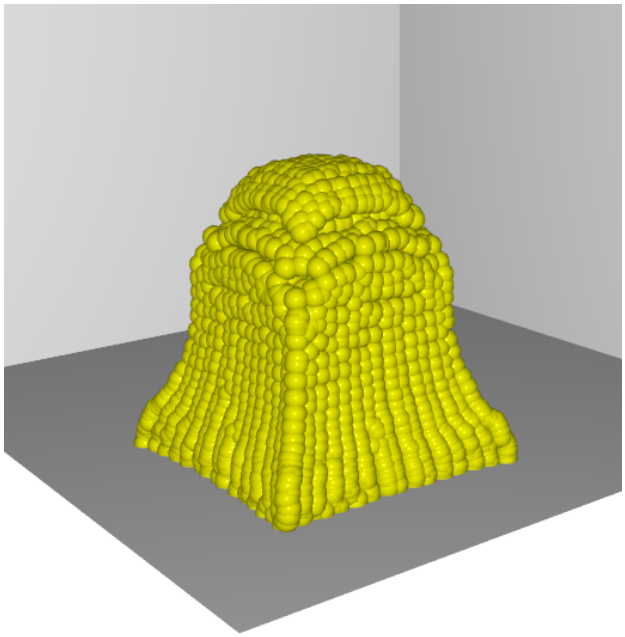

Fig. 6. Animation displaying the particles rather than the surface reconstruction

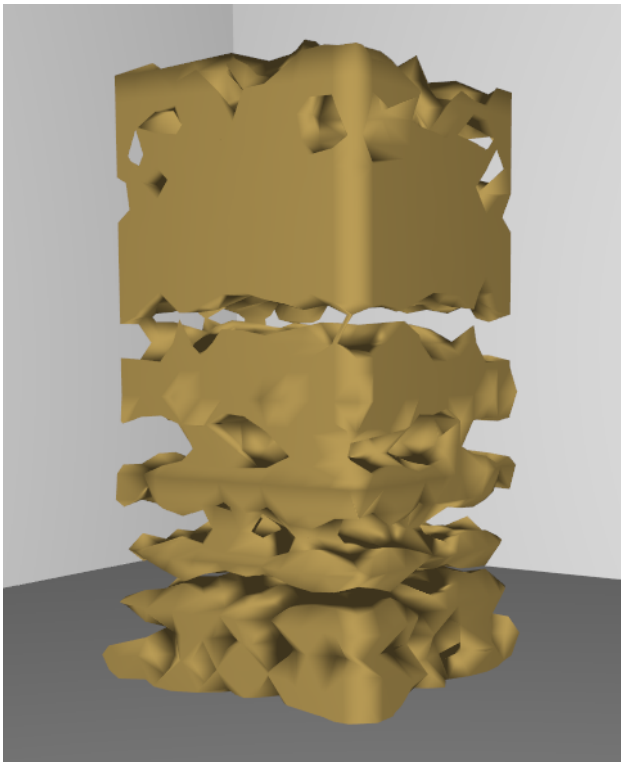

Fig. 7. $r = h$ with h the real radius of a particle

It is very clear that the surface reconstruction yields a much smoother result, even if to get a better result it would be better to have even smaller particles and smaller grid cells.

To choose the value for the radius for the average spacing (since the paper says to set R to twice that radius), I experimented with different values:

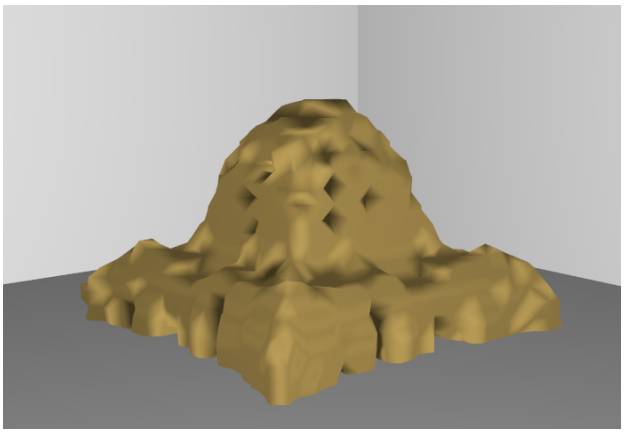

Fig. 8. $r = 1.5h$

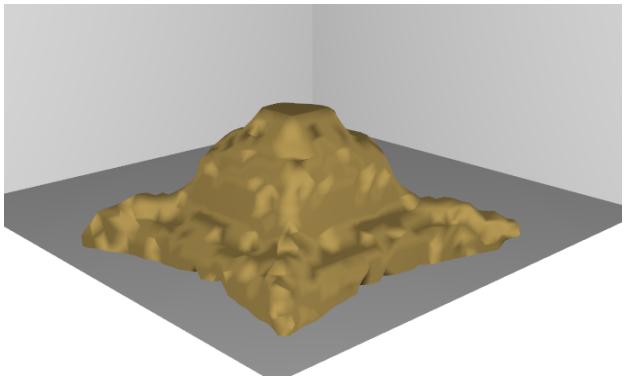

Fig. 9. $r = 2h$ which is what was kept

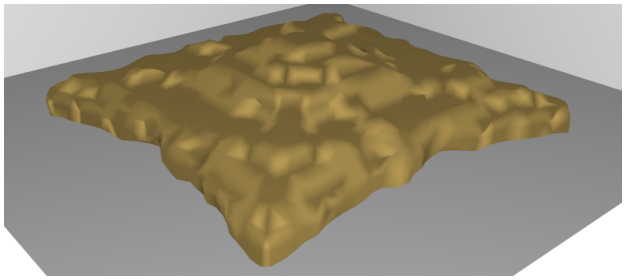

Fig. 10. $r = 2.5h$

It is clear that radii under $2h$ are too small and cause holes to appear in the structure. Radii over $2h$ are good, I decided to pick $R = 2h$ since it is the threshold value, and as such, keeps the maximum level of detail.

I also experimented with different values for the stiffness coefficient α for the divergence correction:

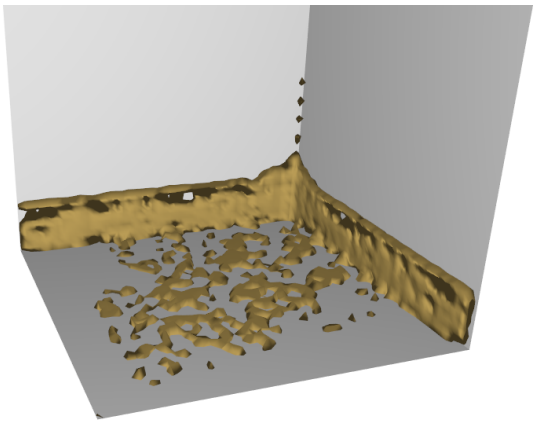

Fig. 11. $\alpha = 1$

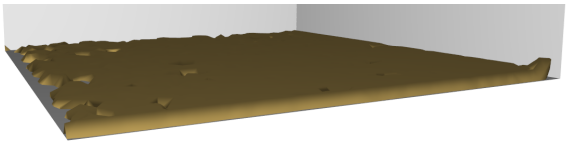

Fig. 12. $\alpha = 0.1$

The sand tends to climb up the walls when the value for α is too high. I picked $\alpha = 0.1$ because the effect was not significant for this value. This is not visible in an image, but the effect of an α greater than zero (i.e. activating the density correction) reduces the compression of the fluid, which makes it spread out a bit faster and gives a more realistic result.

When looking at the details in the animation, it is possible to notice problems in the movement of the particles, of which I do not know the origin:

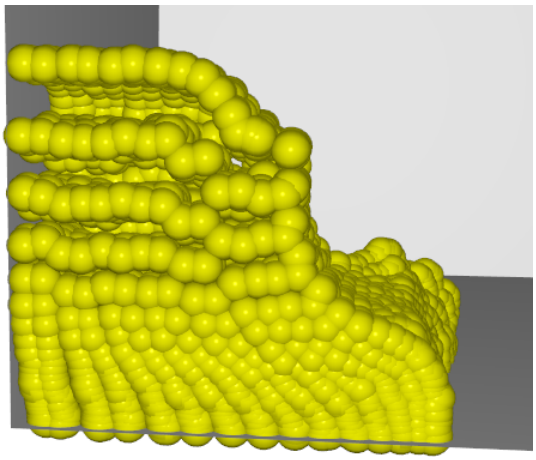

Fig. 13. Apparition of layers between the particles

Conclusion

During this project, I was able to implement a PIC fluid simulator, and to add a surface reconstruction algorithm which provides a better rendering than only the particles.
However, this algorithm could still be much improved upon: there is the problem of the layers, as well as optimizing more the algorithms in order to accelerate the program (I have between 10 and 15 frames per second on my computer using an NVIDIA GeForce RTX 3050 Laptop GPU). The model could also be improved by implementing more functionalities from the paper [1] to yield more realistic results. To avoid the interpenetration of particles, it could be a good idea to replace the

density correction by a collision detector that would ensure that the particles do not enter each other (just like for the walls).

REFERENCES

[1] Yongning Zhu and Robert Bridson. *Animating Sand as a Fluid*, in *ACM Transactions on Graphics 24(3):965-972*, 2005, https://www.cs.ubc.ca/~rbridson/docs/zhu-siggraph05-sandfluid.pdf

[2] Damien Rohmer. *Animating fluids II*, for the course *INF585 - Computer Animation* at École Polytechnique, https://imagecomputing.net/course/2023_2024/inf585/lecture_slides/20_fluids_II/html/content/04_picflip/01_pic_flip/index.html

[3] Ten Minute Physics. *18 How to write a FLIP Water Simulator*, https://matthias-research.github.io/pages/tenMinutePhysics/index.html, video, code and slides accessed 26 February 2024.

[4] Damien Rohmer. *Marching Cube - Dynamic update* in the example code for the *CGP Library* https://imagecomputing.net/cgp/04_examples/index.html, accessed 24 March 2024.