Narrow Band FLIP for Liquid Simulations

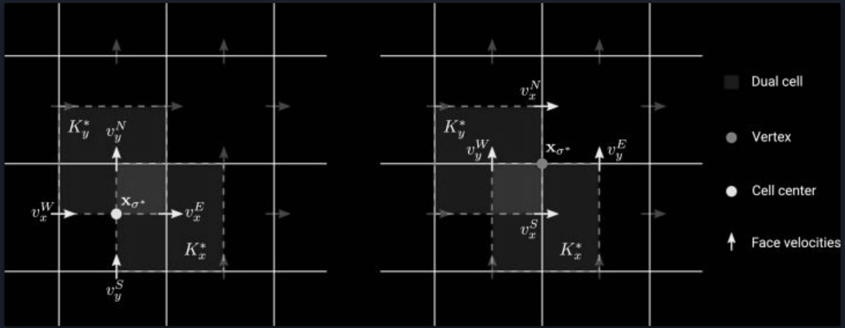
Baptiste Boulan

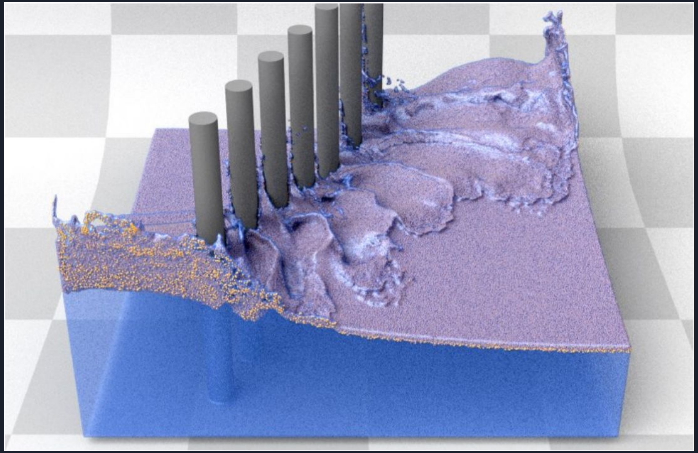
Summary

I. The idea of the paper
   A. The FLIP/PIC fluid simulation
   B. The narrow band
II. The implementation
   A. FLIP/PIC
      1. First version with Gauss-Seidel solver
      2. Second version with Conjugate gradient
      3. GPU acceleration
      4. Extension to 3D
   B. The narrow band optimization
      1. The level set computation
         a) The paper’s idea
         b) my version of it
      2. The narrow band optimization
III. Conclusion
   A. The limitation of my model
   B. What could be done in the future

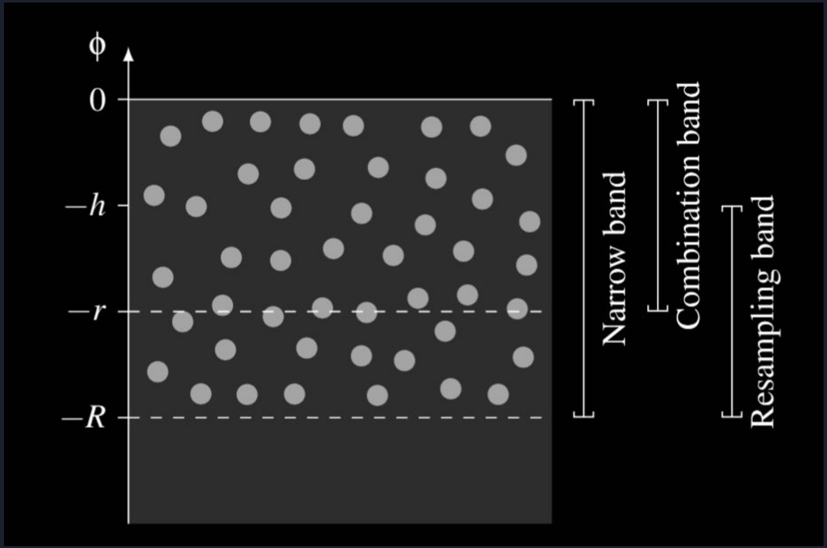
The idea of the paper

- Narrow Band FLIP for Liquid Simulations
  - Florian Ferstl & al
- Starting from a PIC/FLIP fluid simulation.
  - Mix of eulerian and lagrangian simulation.
  - Amazing results.
  - Moderate computational cost.
- BUT:
  - Most of the fluid is non visible and only participating a little to the simulation
- Idea:
  - Save some compassion time in the invisible parts by approximating it.





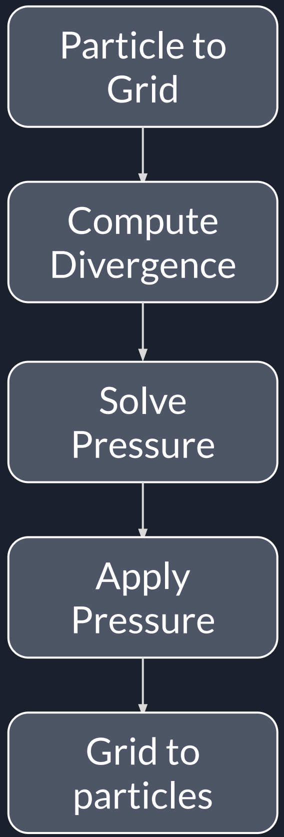
The narrow band



- Determining a narrow band just under the surface.
  - Split it in three main parts.
- Remove the particles outside the narrow band.
  - Eulerian simulation outside the band.
- Re sample in the lower narrow band.
  - Avoid resampling to close to the surface to avoid visual artifacts.

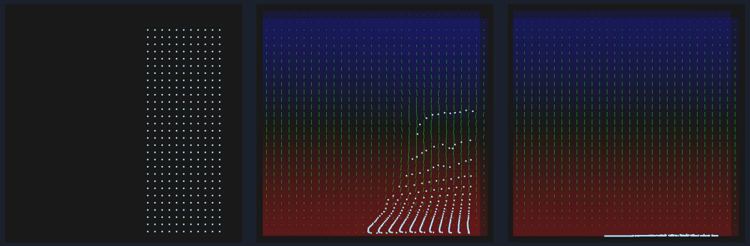
2D FLIP/PIC simulation: Gauss-Seidel Approach

- Solving the linear system by iteratively recomputing the pressure using the already existing pressures
- Pros:
  - Easy to understand
  - Easy to implement
  - The Jacobi method allows GPU adaption.
- Cons:
  - Slow convergence.
  - In our case, not enough.

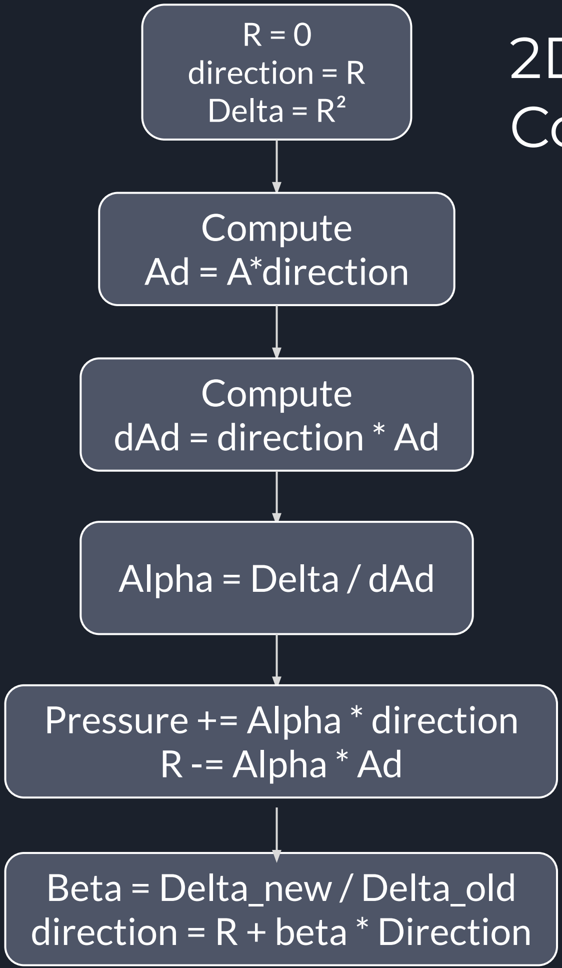


2D FLIP/PIC simulation: Gauss-Seidel Approach

- Solving the linear system by iteratively recomputing the pressure using the already existing pressures
- Pros:
  - Easy to understand
  - Easy to implement
  - The Jacobi method allows GPU adaption.
- Cons:
  - Slow convergence.
  - In our case, not enough.

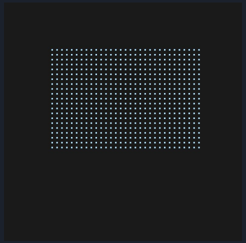
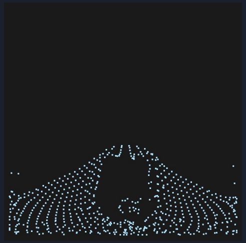


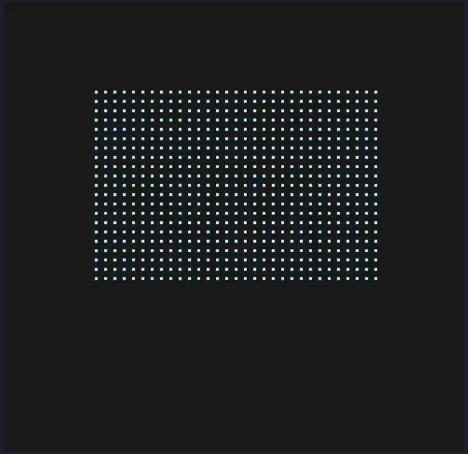
2D FLIP/PIC simulation: Conjugate Gradient Approach



- Solving the linear system by Gradient descent.
- Steepest gradient:
  - Easy to understand.
  - Simple to implement
- Conjugate gradient:
  - A bit trickier to understand
  - A bit more efficient
  - working wonderfully here !

2D FLIP/PIC simulation: Conjugate Gradient Approach

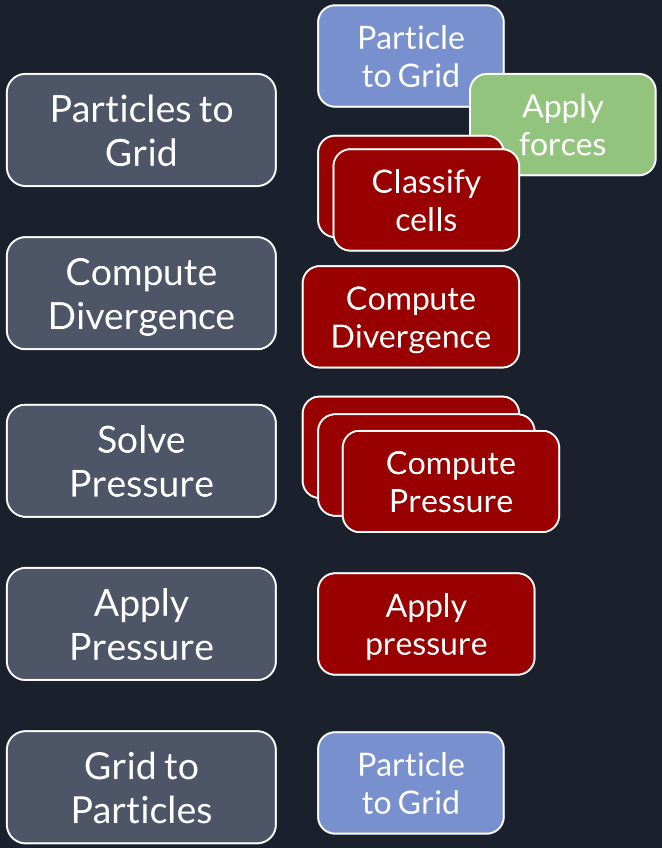





- Solving the linear system by Gradient descent.
- Steepest gradient:
  - Easy to understand.
  - Simple to implement
- Conjugate gradient:
  - A bit trickier to understand
  - A bit more efficient
  - working wonderfully here !

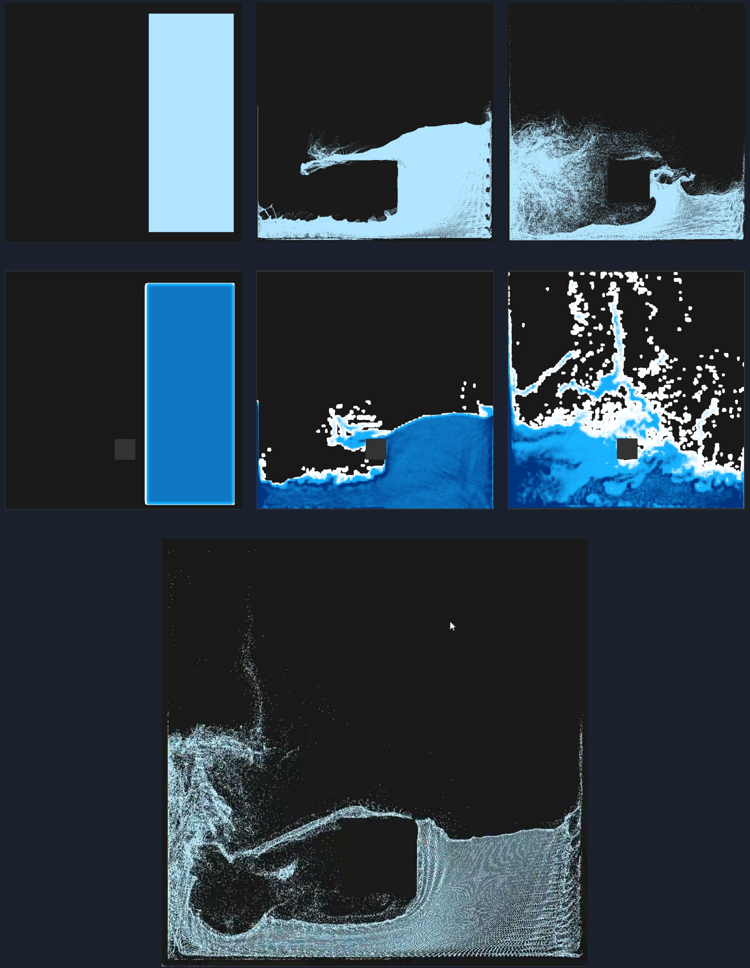
GPU acceleration

- Using to GPU to parallelize the computation linked to the particles, the velocities or the grid.
- Issues
  - Bottleneck with the incessant data transfer
  - Separate shaders trying to write on the same buffer
- Results:
  - CPU: 5 000 particles at 50 FPS
  - GPU: 500 000 particles at 50 FPS

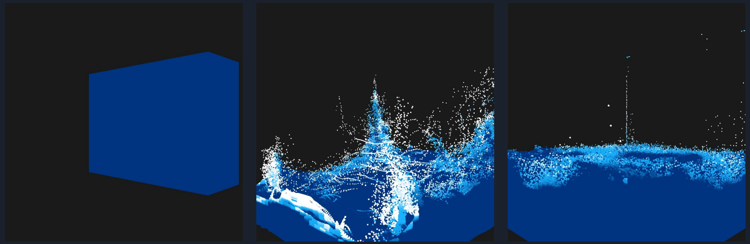


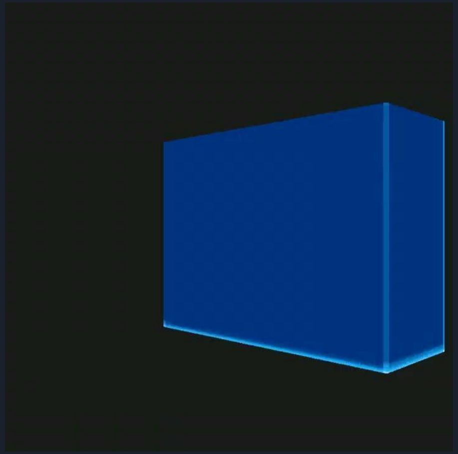
GPU acceleration

- Using to GPU to parallelize the computation linked to the particles, the velocities or the grid.
- Issues
  - Bottleneck with the incessant data transfer
  - Separate shaders trying to write on the same buffer
- Results:
  - CPU: 5 000 particles at 50 FPS
  - GPU: 500 000 particles at 50 FPS



Extension to 3D

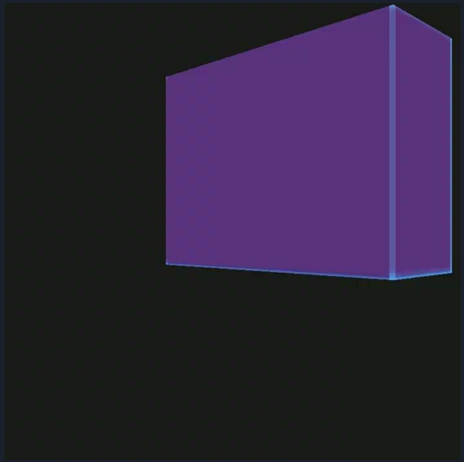


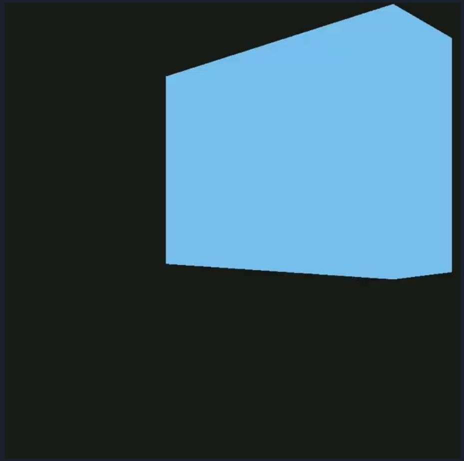


- Careful with the indices
- Increase the computational cost:
  - 2D: 128^2 grid = 16 384 cells.
  - 3D: 64^3 grid = 262 144 cells.
  - For 500 000 particles, the FPS rate drops down to 7.
- Some slight instability introduced

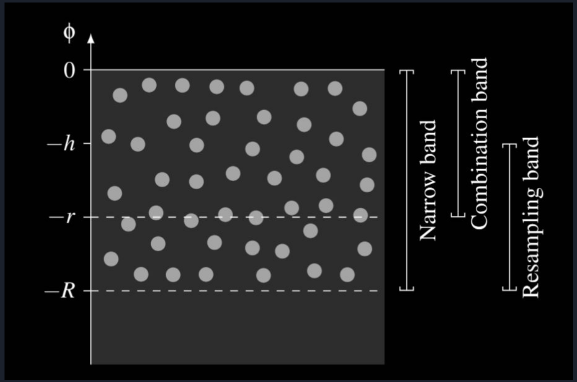
Level Set Computation

- Computation method from the paper:
  - Initialize with analytic formula
  - On every steps:
    - Advect the level set.
    - Offset it by the cell of a size
    - adjust with a particle based level set.
  - I has given me poor results.
- Computation method proposed by me:
  - Reset the level set every steps
  - Initialize the air cells with a particle based level set
  - propagate the distance to the surface from the air cells to the depth of the fluid



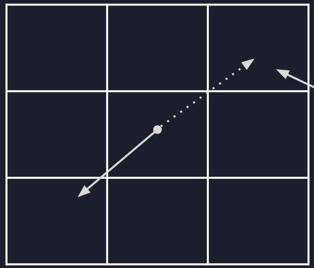


The optimization



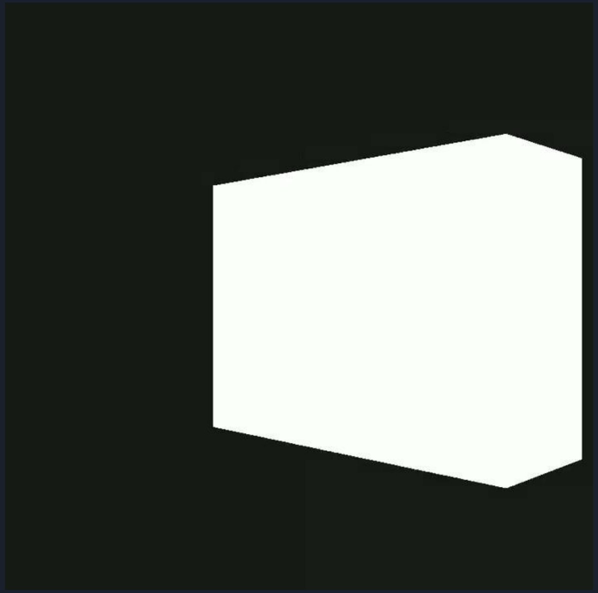
- The narrow band:
  - Only place where the particles are allowed to exist.
  - Outside of it, the particles are deleted and the simulation is eulerian only.
- The resampling band:
  - The lower part of the narrow band.
  - Where we resample region that have a too low amount of particles.
- The combination band:
  - The only place where we apply the semi-lagrangian logic.

u’ = u(x-u*dt)



Imaginary old position

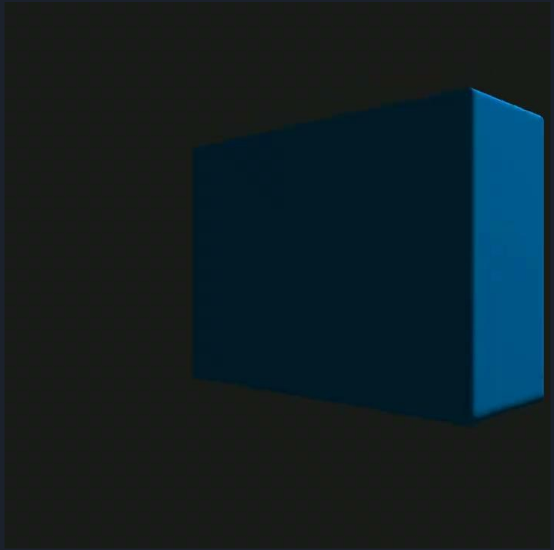
The optimization



- The narrow band:
  - Only place where the particles are allowed to exist.
  - Outside of it, the particles are deleted and the simulation is eulerian only.
- The resampling band:
  - The lower part of the narrow band.
  - Where we resample region that have a too low amount of particles.
- The combination band:
  - The only place where we apply the semi-lagrangian logic.

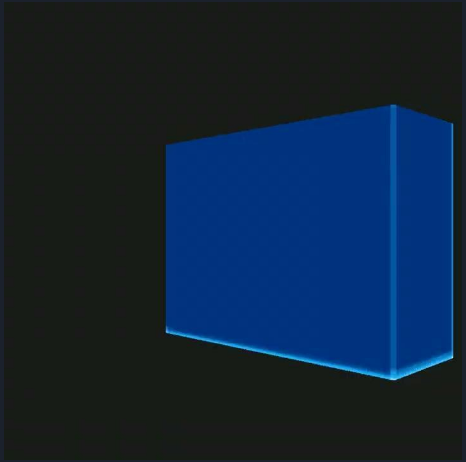
Marching cubes

- In theory the algorithm can handle about 18 million particles in a 128^3 grid at about one frame per second.
  - However in practice the GPU struggles to display it all.
- I therefore used a simple marching cubes algorithm to render the surface.
  - Using the level set as field of value.
  - still struggling near the edges.



Conclusion

- Limitations:
  - Results clearly not as good as the originals.
  - Only saving time if there is a large amount of hidden fluid.
- Futur work:
  - Improvement could be made on my cheap level set.
  - surface tension would look nice.



Thanks :)