ChatGPT

Stage 1: Foundational 2D Fluid Simulation (Python/Matplotlib)

1. Fluid Theory Fundamentals

- **Incompressible flow & Navier–Stokes:** Read about incompressible fluids (density nearly constant for water) and study the Navier–Stokes (momentum) and continuity (mass) equations [1] [2]. *Outcome:* Understand how the 2D Navier–Stokes equations (with $\nabla \cdot u = 0$) govern fluid flow. *(Time: ~1–2 weeks)*.
- **Eulerian vs Lagrangian perspective:** Learn the two viewpoints: Eulerian (fixed grid storing velocity/pressure) vs Lagrangian (particles moving with fluid) [3] [4]. *Outcome:* Be able to contrast grid-based (Eulerian) and particle-based (Lagrangian) methods, and why each is used [4] [5]. *(Time: ~1 week)*.
- **Shallow water basics (optional):** (For extra insight) derive the 2D shallow-water equations (conservation of fluid column height and momentum) and code a simple 1D wave (dam-break) solver. *Outcome:* See a concrete example of a simplified fluid model (height+velocity). *(Time: ~1 week)*.
- **SPH intuition – smoothing kernels:** Study Smoothed-Particle Hydrodynamics (SPH) fundamentals. Understand that SPH treats fluid as particles with mass, using kernel functions to reconstruct continuous fields [6]. Read about how each particle's density is computed by summing neighbors' masses weighted by a smoothing kernel, and how pressure/viscosity forces arise from those densities [6] [7]. *Outcome:* Be able to explain the SPH algorithm steps (density sum, pressure force, viscosity force, integration) in words. *(Time: ~1–2 weeks)*.

2. Basic 2D Grid-Based Solver in Python

- **Grid setup and data structures:** Initialize a 2D grid of size N×N (plus ghost boundary cells) and allocate arrays for the velocity components (u,v) and pressure (or fluid density/dye). Apply boundary conditions (e.g. solid walls). *Expected outcome:* Code creates and visualizes an initial velocity/density field on a grid. *(Time: ~1 week)*.
- **Advection (semi-Lagrangian):** Implement a semi-Lagrangian advection step for the velocity and any scalar field (dye or fluid density). For each grid cell, trace backward along the velocity field to interpolate values from previous step. This moves fluid quantities without instability. For example, the Stable Fluids method uses exactly this approach [8]. *Expected outcome:* You can "push" dye/velocity through the field and see it transported on the grid. *(Time: ~1–2 weeks)*.
- **Diffusion and viscosity:** (Optional at first) add a diffusion step by solving a heat equation on the grid for viscosity effects. This typically involves iterating a linear solver (Jacobi relaxation) to diffuse velocity or density [9]. *Expected outcome:* Observe slight smoothing of the flow over time. *(Time: ~1 week)*.
- **Pressure projection:** Enforce incompressibility by solving a Poisson equation for pressure (the "pressure projection" step) [10]. Compute the divergence of the intermediate velocity, solve for pressure via Jacobi or a sparse solver, then subtract the pressure gradient from velocity so that $\nabla \cdot u \approx 0$. *Expected outcome:* A divergence-free velocity field that preserves mass. *(Time: ~1–2 weeks)*.

- **Time-stepping loop:** Combine steps into a simulation loop: apply external forces (e.g. gravity or a force field), advect velocities and scalars, project pressure each timestep, and update t+=Δt [10] [9]. After each step, use Matplotlib (`imshow` or `quiver`) to plot the fluid (e.g. velocity magnitude or dye density). *Expected outcome:* A working 2D fluid solver that shows flowing/spinning fluid (e.g. a vortex or dye swirling around) [10] [8]. *(Time: ~2–3 weeks total).*

3. Basic 2D SPH Simulation in Python

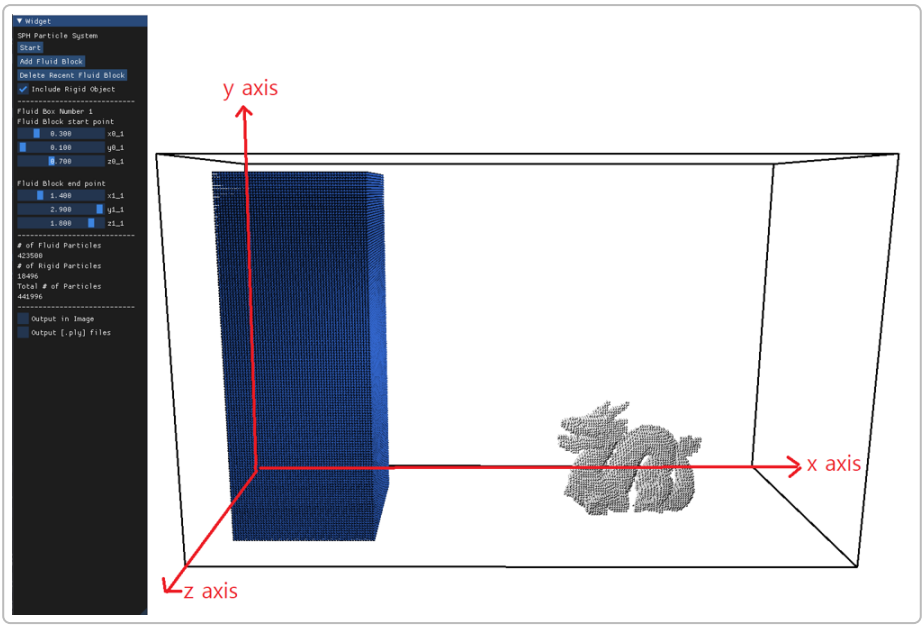

Illustration: An SPH fluid simulation (blue particles in a container) where each particle's motion is determined by smoothed pressure/viscosity forces [5].

- **Particle representation:** Create a list of fluid particles, each with mass, position (x,y), and velocity (vx,vy). Initialize them in a shape (e.g. a block of fluid). *Outcome:* Particles rendered as points in a 2D plane. *(Time: ~1 week).*

- **Kernel functions:** Implement smoothing kernels (e.g. the poly6 kernel for density, spiky kernel for pressure gradient). These compute weights W(r,h) based on distance. *Outcome:* A function W(r) that gives higher weights to nearby particles, as in SPH theory [6]. *(Time: ~1 week).*

- **Density and pressure:** For each particle *i*, sum over neighbors *j* within the smoothing radius: $\rho_i = \sum m_j W(r_{ij})$ [6]. Then compute pressure $p_i = k(\rho_i - \rho_0)$. *Outcome:* Array of densities and pressures for all particles. *(Time: ~1 week).*

- **Pressure and viscosity forces:** Compute forces on each particle: use the SPH pressure gradient formula (forces from neighbors' pressures) and a viscosity term based on relative velocities [7]. For example, pressure force on *i* is $\sum m_j (p_i+p_j)/(2\rho_j) \nabla W(r_{ij})$. *Outcome:* Acceleration vectors for all particles. *(Time: ~1 week).*

- **Integration (time-stepping):** Update velocities and positions using explicit Euler or a simple integrator: v_i += (force$_i$/m_i)Δt + $g\Delta t$; x_i += $v_i\Delta t$. *Outcome:* Particles move according to computed forces (e.g. a simulated "water drop" falling and splashing). *(Time: ~1 week).*

- **Boundary handling:** Add simple walls (reflecting or force-feedback) so particles stay contained. *Outcome:* Realistic fluid confinement. *(Time: ~0.5–1 week).*

- **Visualization:** Use Matplotlib to scatter-plot particle positions each frame (optionally color by density or

speed). *Outcome:* A visible 2D SPH fluid animation (e.g. dam break or steady flow) [5] [7]. *(Time: ~0.5–1 week)*.

4. Data Generation for ML Training

- **Frame export (grid):** Modify your grid-based solver to save simulation frames (velocity or density fields) to disk each timestep [10]. For example, use `matplotlib.pyplot.imsave` or `numpy.save` to record the 2D arrays. *Outcome:* A sequence of 2D array files (or images) showing fluid evolution. *(Time: ~0.5 week)*.
- **Snapshot export (SPH):** Similarly, export particle states from the SPH simulation (e.g. write out each particle's (x,y) to a file or image). *Outcome:* Time-series data of particle coordinates. *(Time: ~0.5 week)*.
- **Parameter sweeps:** Automate runs with varied initial conditions (fluid shapes, force values, viscosity) to produce a diverse dataset. *Outcome:* A library of simulation sequences for ML (saved as `.npy`, `.csv`, or image stacks). *(Time: ~0.5–1 week)*.

5. Optional Bonus – Unity Visualization

- **Unity import:** (After completing Python code) optionally write a Unity script to read your exported data (arrays or images) and visualize it. For instance, map grid data to a texture, or spawn Unity sprites/particles at the SPH particle positions each frame. *Outcome:* An interactive Unity scene displaying the fluid motion as a visual bonus. *(Time: ~1 week, bonus)*.

Each of the above steps builds on the last. By the end of Stage 1 you should have: written notes on fluid dynamics theory [1] [6], a working 2D grid-based fluid solver, a working 2D SPH solver, and exported datasets of their outputs. This modular roadmap lets you pause and resume easily at clear checkpoints (for example, *"Grid solver complete"* or *"SPH forces implemented"*).

Sources: Fluid simulation theory and algorithms [1] [4] [10] [9] [8] [6] [5] [7].

---

[1] [2] [3] [4] [10] **cg.informatik.uni-freiburg.de**
https://cg.informatik.uni-freiburg.de/intern/seminar/gridFluids_fluid-EulerParticle.pdf

[5] **GitHub - taehoon-yoon/SPH-Fluid-Simulation: Smoothed Particle Hydrodynamics implementation with Python**
https://github.com/taehoon-yoon/SPH-Fluid-Simulation

[6] **Basics of the Smoothed Particle Hydrodynamics (SPH) Method**
https://altair.com/blog/articles/Basics-of-the-Smoothed-Particle-Hydrodynamics-(SPH)-Method

[7] **Haoran Liang - SPH Fluid Simulation**
https://haoranliang.com/fluid-simulation

[8] **GitHub - murdock-aubry/fluids-playground**
https://github.com/murdock-aubry/fluids-playground

[9] **Fluid Simulation Python Projects**
https://matlabprojects.org/fluid-simulation-in-python/