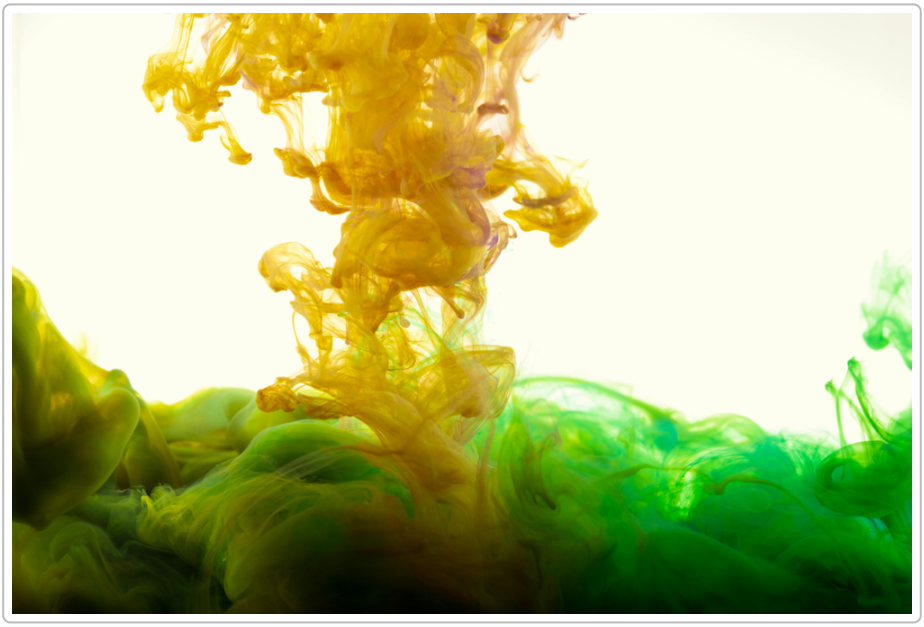
ChatGPT

Curriculum: Machine Learning for Realistic Water Simulation

Stage 1: Fluid Dynamics Fundamentals and Classical Simulation



Begin by learning basic fluid mechanics and classical simulation methods. Study incompressible flow (Navier–Stokes equations), shallow-water equations, and particle methods like Smoothed-Particle Hydrodynamics (SPH). Good resources include *Real-Time Fluid Dynamics for Games* (Jos Stam) and position-based fluid simulation (Macklin & Müller). Experiment with open-source tools (e.g. **SPlisHSPlasH** with Python bindings [1] or Blender/MantaFlow) to generate simple 2D data. For example, simulate a single water droplet or ripple on a flat surface to create a dataset of particle positions and velocities. Ensure projects are small enough to run on an RTX 2060 (e.g. a few hundred particles or a 64×64 grid). Key learnings include handling boundary conditions and stability (diffusion, viscosity).

- **Theory:** Navier–Stokes basics, SPH principles, stability in real-time fluids (Stam 1999).
- **Tools:** Python, NumPy/SciPy, **pysplishsplash** (Python SPH simulator) [1], MantaFlow or PhiFlow for 2D fluids.
- **Project:** Implement a 2D SPH droplet or wave. Use the simulator to produce training data (particle states over time). Visualize outputs (e.g. with Blender or Matplotlib). Confirm correct splashes or waves in the classical simulation before moving to ML.

Stage 2: 2D ML Surrogate using Convolutional Networks

Train a neural surrogate on simple 2D fluid data. Start by treating the fluid grid as an image: use Convolutional Neural Networks (CNNs) to predict the next velocity or pressure field. Study prior work like Tompson et al. (2017) *"Accelerating Eulerian Fluid Simulation with Convolutional Networks"* [2], which solves the pressure Poisson step with a CNN. Gather training data by running a classical 2D solver (e.g. MantaFlow) on modest grids (e.g. 64×64). Train the network on one-step predictions (supervised loss on fluid velocity). Evaluate visual realism (ripples, vortex) and frame rate. This stage uses less GPU memory (2D CNN on small grids), suitable for local training.

- **Theory:** Physics-informed CNNs for PDEs; supervised learning of flow fields [2]. Understand how convolutional filters can approximate differential operators.
- **Tools:** PyTorch or TensorFlow, PhiFlow (for differentiable baselines), MantaFlow or custom 2D CFD code for data. Use GPU for training on small 2D grids.
- **Project:** Train a CNN (e.g. U-Net) to predict one timestep of a 2D water surface. For example, given height or velocity at time t, predict time $t+\Delta t$. Visualize as animation. Confirm that the learned model produces plausible ripples and conserves basic fluid patterns. This builds intuition on data-driven fluid dynamics.

Stage 3: Graph Neural Networks for 2D Particle Fluids

Extend to particle-based models and Graph Neural Networks (GNNs). In a Lagrangian view, represent each water particle as a node in a graph, with edges connecting nearby particles. Study GNN theory and architectures (e.g. message-passing networks [3]). Key references: Li & Farimani's *"Learning Lagrangian Fluid Dynamics with Graph Neural Networks"* (ICLR 2021) [4], which decomposes advection, collision, and pressure via graph nets. Use PyTorch Geometric (PyG) or Deep Graph Library to build the model. Your GNN input: node features (position, velocity, etc.) and learned edge interactions. Train the GNN to predict particle accelerations or position updates from simulated data (generated in Stage 1). Aim for small 2D cases (hundreds of nodes) to fit GPU memory.

- **Theory:** Message-passing GNNs, graph representations of SPH. Review *Graph Network* models (Battaglia et al., 2018) and particle-based GNNs [4]. Understand inductive biases (e.g. permutation invariance, locality).
- **Tools:** PyTorch Geometric or DGL for GNNs, PyTorch for neural nets. Continue using pysplishsplash or similar to generate training data (positions, neighbor lists).
- **Project:** Implement a GNN fluid simulator on 2D particles. For example, use an edge MLP to compute forces between neighbors and a node MLP to update velocities. Train on the droplet dataset. Compare GNN rollout versus ground truth (splash shapes, stability). Ensure it generalizes to variations in droplet size or initial velocity.

Stage 4: Extending to 3D Fluid Simulations

Scale up to 3D fluids. Increase complexity gradually to 3D water volumes or waves. Continue with particle/GNN models or a hybrid grid-GNN approach. For example, use fewer particles (hundreds to a few thousand) due to GPU limits, or employ hierarchical GNNs. Papers like Kumar & Vantassel's GNS (2022) demonstrate 3D flows: a PyTorch Graph Network Simulator predicts accelerations for 3D fluid/granular flows 5,000×

faster than traditional MPM [5]. The user may now need cloud GPUs for larger training, but start with local GPU on small volumes. Also explore "MeshGraphNets" (Pfaff et al., 2021) [3] which handle arbitrary meshes – useful if representing water surface as a mesh.

- **Theory:** High-dimensional GNNs, 3D fluid dynamics. Read GNS [6] for generalizable GNN structure, and MeshGraphNets [3] for mesh adaptation. Understand how to embed 3D positions in graphs.
- **Tools:** PyTorch Geometric; consider PyTorch3D or custom for 3D meshes. Possibly use Taichi or UFL libraries to cross-check. Use cloud (AWS, GCP, or Colab Pro) when local GPU is insufficient.
- **Project:** Simulate a simple 3D fluid scenario (e.g. a water drop or a block of fluid). Train a GNN to predict short-term motion. Validate on small 3D data (e.g. 200–1000 particles). Check visual plausibility (wake, splash). Optimize model size (e.g. reduce hidden dims or batch size) to fit memory. This stage transitions from 2D visuals to 3D volumetric fluid behavior.

Stage 5: Complex Interactions – Splashes, Wind, and Obstacles

Introduce environmental forces and collisions. Model water interacting with objects or wind. Extend the GNN to include static scene elements: either add boundary nodes or use a mesh-based representation. MeshGraphNets propose splitting computations into *mesh-space* (internal fluid dynamics) and *world-space* (external forces/collisions) [7]. Similarly, include features like local wind velocity, gravity changes, or object positions. Experiment with obstacles (pillars, moving paddles) and forces (2D or 3D). For example, simulate a water block hitting a wall or being blown by a fan. Train the model on these scenarios and test generalization. This stage pushes towards visually realistic effects (splash shapes, droplets) and leverages domain knowledge (e.g. preserving incompressibility).

- **Theory:** GNNs with external interactions. Study how to incorporate collisions or boundaries, e.g. by adding nodes for obstacle surfaces. Review MeshGraphNets on passing messages in both mesh and world spaces [7] to handle collisions. Consider physics-informed losses (e.g. approximate divergence-free flow).
- **Tools:** Same GNN frameworks; extend datasets to include collision geometry. You may use Blender/MantaFlow to simulate fluid hitting objects for ground truth.
- **Project:** Create an animated scenario (e.g. water in a tank with a moving object). Use your GNN to predict the outcome. Compare to a physics sim (surface shapes, splash trajectories). Optionally, use reinforcement learning (e.g. Unity ML-Agents) to learn control policies for object motion that produce realistic fluid splashes. The goal is a rich training environment preparing for integration.

Stage 6: Deployment to Game Engine (Real-Time Inference)

Finally, integrate the trained ML simulator into a real-time game engine. Export the model (e.g. PyTorch → ONNX) and use engine-specific runtimes. In Unity, the Barracuda library supports ONNX models [8]. For example, add the `.onnx` file to Assets, load it with `ModelLoader.Load`, and run inference each frame to update particle states. For Unreal Engine, use its Neural Network Inference (NNI) plugin or Python API to run ONNX models. Optimize the model for speed: quantization (FP16), pruning, or a smaller architecture to achieve ~30+ FPS. Tools like NVIDIA TensorRT or Unity's Burst compiler can help. The final milestone is a

working demo: a Unity or Unreal scene with ML-driven water that reacts to user-controlled objects (e.g. a paddle or wind zones) in near-real-time.

- **Theory:** Model export and runtime constraints. Understand ONNX format and inference pipelines. Plan for latency and precision trade-offs.
- **Tools:** ONNX export in PyTorch; Unity Barracuda (NNModel) [8] ; Unreal Engine MLAdapter or ONNX plugin; possibly TensorRT or GPUs on console/PC. Use Unity's Particle System or custom shaders to visualize the ML-predicted fluid surface.
- **Project:** Deploy a proof-of-concept: for instance, a Unity demo where a GNN predicts next particle positions each frame under user input. Verify timing (profile inference) and visual fidelity. Document the workflow (training→export→engine) and prepare the model as a reusable API (e.g. C# script calling the model).

Throughout the curriculum, the focus is on incremental, verifiable progress from simple to complex. Early stages emphasize GPU-feasible tasks (small 2D data) and build intuition with citations such as Tompson et al. (CNNs for fluid) [2] and Li & Farimani (GNN fluids) [4] . Later stages leverage state-of-the-art GNN research [3] [5] for scalable fluid modeling. By the end, the user will have a deployable, ML-based water simulator capable of realistic splashes and interactions, suitable for integration into modern game engines.

Sources: Key references include GraphNet-based fluid simulators [4] [5] [3] and deployment guides [8] cited above. These inform the theoretical foundations and practical steps in each stage.

---

1 **GitHub - InteractiveComputerGraphics/SPlisHSPlasH: SPlisHSPlasH is an open-source library for the physically-based simulation of fluids.**
https://github.com/InteractiveComputerGraphics/SPlisHSPlasH

2 **[1607.03597] Accelerating Eulerian Fluid Simulation With Convolutional Networks**
https://arxiv.org/abs/1607.03597

3 7 **[2010.03409] Learning Mesh-Based Simulation with Graph Networks**
https://ar5iv.org/pdf/2010.03409

4 **Learning Lagrangian Fluid Dynamics with Graph Neural Networks | Papers With Code**
https://paperswithcode.com/paper/learning-lagrangian-fluid-dynamics-with-graph

5 6 **[2211.10228] GNS: A generalizable Graph Neural Network-based simulator for particulate and fluid modeling**
https://ar5iv.org/pdf/2211.10228

8 **Importing a trained model | Barracuda | 1.0.4**
https://docs.unity3d.com/Packages/com.unity.barracuda%401.0/manual/Loading.html