Physics-based Simulation for Aerial Additive Manufacturing Using Viscoelastic Expanding Fluids

Nevò Mirzai Hamadani, Yusuf Furkan Kaya, Mirko Kovac, *Fellow, IEEE,*

***Abstract*—Aerial Additive Manufacturing (Aerial AM) promises to enable on-site construction of large structures using drones for additive manufacturing. Real-world experiments, however, are costly and time-consuming. We present a physics-based simulation framework built on the GPU-native Genesis engine that models the deposition of viscoelastic materials, with curing and volumetric expansion represented as time-scheduled particle transitions. We select the Material Point Method (MPM) as the primary solver, as it better handles large deformations and maintains layer stability compared to SPH or PBD. Performance is evaluated across different scenarios under fixed numerical settings, averaged over repeated runs, reporting frames per second and the simulation-to-real time ratio (with and without rendering). Results show that runtime is primarily driven by particle count, with curing adding a modest overhead, expansion a stronger one, and the combination of both with drone geometry being the most demanding. Rendering constitutes a major bottleneck, while headless (no rendering) runs remain stable and scalable. Overall, the framework provides a reliable platform for Aerial AM studies and establishes a foundation for future work on drone control, reinforcement learning, and digital-twin applications.**

***Index Terms*—Physics simulation, Additive Manufacturing (AM), Drones, Viscoelastic fluids**

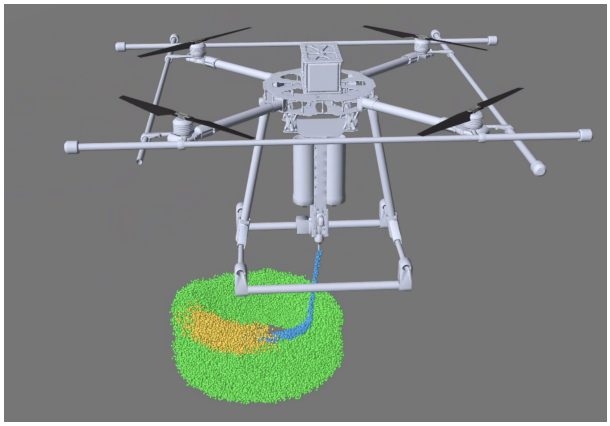


Fig. 1: Drone-based emission of an expanding and curing foam, forming concentric layers that build up a cylindrical structure.

I. Introduction

SINCE the early 2010s, additive manufacturing methods, especially 3D printing, have seen a rapid increase in popularity, supported by improvements in consumer printers and accessibility [1]. In parallel, drones have also seen widespread adoption during this period [2]. Aerial additive manufacturing (Aerial AM) is a new approach that unites 3D printing capabilities with aerial vehicles, therefore overcoming traditional spatial limitations that limit conventional 3D printers. The concept was first demonstrated by Zhang *et al.* using two drones, one for building and one for scanning the environment and the structure; with these, they were able to print free-standing large-scale structures [3]. Recently, Kaya *et al.* [4] critically reviewed the state-of-the-art and described how Aerial AM could, in principle, be extended to automated on-site construction, pointing to its potential for fabricating infrastructure in hard-to-reach locations.

Testing Aerial AM methods in the real world requires preparing the environment, drones, and printing materials, making it a costly and time-consuming process. Existing simulation frameworks are usually designed with a narrow focus: they specialize either in the rigid dynamics of the aerial vehicle or in the fluid dynamics of the extruded material. Because these simulators are tailored to a single domain, they rarely capture the rigid–fluid interactions that are central to Aerial AM. As a result, there is little research on frameworks that model both aspects simultaneously. Moreover, the limited work that does exist generally lacks support for GPU-parallelized simulation, which is essential for applications such as reinforcement learning–based drone control or real-time digital-twin observation. Developing such a simulation framework for Aerial AM would accelerate research in this area, reduce reliance on costly physical testing, and provide the tools needed to study structural stability of printed objects more effectively.

A closely related work is that of Berdica *et al.*, who integrated a Smoothed-Particle-Hydrodynamics (SPH) material simulation into Gazebo to study mobile 3D printing robots [5]. Their framework combines robot kinematics with viscoelastic-fluid deposition, but runs at only about 5 FPS on a desktop PC, making it impractical for closed-loop control, real-time simulation, or learning tasks. SPH also struggles with high viscosity ratios, contact discontinuities, and structural stability, reducing its realism for viscoelastic materials. In addition, Gazebo's CPU-focused architecture limits GPU parallelization. In this project, we pursue a similar vision, but for *aerial* robots and using the Material Point Method (MPM). Compared to SPH, MPM is more stable (it does not require prohibitively small timesteps), better captures large deformations and shape changes, and provides more reliable predictions of structural stability in printed parts. This makes it a more suitable foundation for simulating aerial additive manufacturing.

The authors are with the Laboratory of Sustainability Robotics, École Polytechnique Fédérale de Lausanne (EPFL), CH1015 Lausanne, Switzerland.

Beyond the work of Berdica, other studies have also contributed ideas that are useful for designing an Aerial AM simulator. Angelidis *et al.* developed an SPH-based framework for robotics that allows for real-time interaction with objects [6]. However, their simulator does not include flying robot dynamics and mainly focuses on liquid fluids, not viscoelastic ones. Libraries such as CRESSim-MPM, which use GPU acceleration, show that it is possible to simulate millions of material points efficiently by taking advantage of CUDA-based GPU computing. There are also new approaches based on graph networks, which learn to simulate materials much faster than classic FEM or SPH methods [7], [8]. Still, these machine learning methods require large amounts of accurate simulation data for training. Overall, the literature shows that creating a high-performance, all-in-one simulator for Aerial AM is both possible and needed.

In this work, we propose a simulation framework specifically designed for Aerial AM that is both fast and accurate, leveraging the new (2024) generative simulation platform Genesis [9]. Genesis achieves very high frame rates when compared with other simulation platforms such as Isaac Sim and Gazebo, and it natively supports both SPH and MPM fluid simulation. Its GPU-native architecture allows large-scale parallel simulation of environments for reinforcement learning policy training. We present a comparison between multiple different fluid emission simulations using MPM in the context of Aerial AM.

II. Method

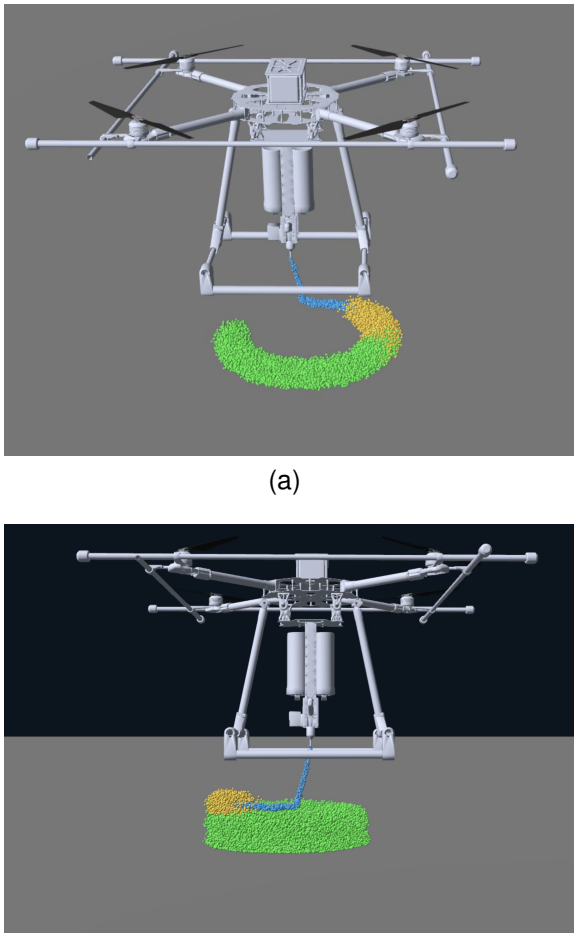
A. Task and physical assumption

Printing goal definition: The goal is to deposit a flowable medium in stacked layers to form a free-standing column. An idealized emitter follows a circular toolpath; each pass lays one closed ring, and successive rings build the tower height (Figure 2). The drone model is shown only to visualize nozzle pose; no flight dynamics or control is simulated. The material may change state during or after deposition (expansion and curing), but those effects are handled by the simulator's material model rather than by modifying the toolpath itself. This pattern generalizes to other layer shapes for walls and shells used in Aerial AM [3].

Material / fluid properties: We model the printed medium as a viscoelastic, expanding material suitable for deposition. In the liquid state, it flows and coalesces, while in the cured state, it supports load and resists deformation. The model exposes: density ρ, elastic parameters (e.g., E, ν), and, where used, plastic or yield thresholds. Curing is represented as a time-dependent transition that increases stiffness and reduces flow. Expansion is also time-dependent, and it is represented as controlled volumetric growth (via particle duplication) that preserves the total mass. Fluids are simulated using the Material Point Method (MPM); more details about this and other fluid models are provided in Section II-B. Temperature, humidity, and detailed reaction kinetics are not modelled explicitly.

Numerical / simulation scope: All simulations are three-dimensional under uniform gravity. The computational domain is defined as a bounded box large enough to contain the emitter trajectory and the deposited structure; particles leaving the domain are reflected by its walls. The developers of the simulation platform used in this work (Genesis, see Section II-C) are actively working on implementing boundless MPM for large-scale environments [9]. Until such functionality is available, simulations remain constrained to a finite box. Rigid–fluid interaction is modelled as solid contact with friction, while the environment is treated as kinematic. Numerical stability depends on spatial and temporal discretization; for MPM, it is mainly governed by the background grid resolution. In both cases, time step size and sub-stepping are tuned accordingly. Collisions among particles are resolved internally by the solver. All simulations are executed headless on a GPU, enabling many environments to run in parallel (batch rollouts) to support future reinforcement learning applications.

Output / evaluation metrics: Physical outcomes are primarily evaluated in qualitative terms by comparing the simulated behavior to real-world experiments (e.g., foam expansion, final layer formation, structural rigidity, and overall shape). Computational performance, on the other hand, is assessed quantitatively by comparing Sim/Real time ratio, frame-rate (FPS), number of particles and other parameters



(a)

(b)

Fig. 2: Formation process of concentric layers through by fluid emission from a drone.

Criterion	MPM	SPH	PBD
Paradigm	Particle–grid continuum	Lagrangian particles (kernel)	Constraint-based particles
Fidelity	High for deformables/viscous	Good for liquids	Plausible but not accurate
Stability / Δt	Medium; medium Δt	Low, smaller Δt	High, larger Δt
Computational cost	Mid-high	High	Low
Best for	**Realistic viscoelastoplastic fluids (this work)**	Realistic liquids	Real-time demos/games
Limits	Memory/cost; grid artifacts	High-viscosity, non-Newtonian	Low physical accuracy

TABLE I: Fluid simulation methods comparison: MPM, SPH and PBD.

(Section III).

B. Material models: MPM, SPH & PBD

We initially considered three complementary families of methods to simulate the deposition of a viscoelastic, expanding medium for Aerial AM: the Material Point Method (MPM), Smoothed Particle Hydrodynamics (SPH), and Position-Based Dynamics (PBD). Each involves trade-offs in fidelity, stability, and computational cost (Table I). Among these, MPM was selected as it can accurately capture the viscoelastic and curing behavior essential for Aerial AM. SPH and PBD are retained for comparison, to contrast their respective qualities with those of MPM, as they are both widely used but physically less accurate. By extending the MPM solver in Genesis, we implemented two process effects: *curing* (a time-dependent increase in stiffness/viscosity) and *expansion* (controlled volumetric growth modelled to conserve total mass).

1) Material Point Method (MPM): MPM represents matter with Lagrangian material points that carry *mass* m, *position* $\mathbf{x}$, *velocity* $\mathbf{v}$, *deformation gradient* $\mathbf{F}$ (state), and material parameters used in this work: *density* ρ, *dynamic viscosity* η, *elastic moduli* (E, ν), and (when plasticity is enabled) a *yield stress* σ_y. Forces and momentum are exchanged on a background Eulerian grid. This hybrid particle–grid formulation can handle large deformations and liquid–to–solid transitions, making it suited for viscoelastoplastic behavior and post-deposition structural stability.

Spatial resolution is governed by the grid cell size Δx and the particle spacing (often summarized as particles-per-cell); both must be chosen consistently to avoid under-resolution. The simulation time step (Δt) is restricted by standard stability bounds. In particular, the elastic (wave-propagation) CFL constraint and the viscous (diffusion) constraint are:

$$\Delta t \le \alpha_{\text{el}} \frac{\Delta x}{c}, \qquad c = \sqrt{\frac{K + \frac{4}{3}G}{\rho}}, \tag{1}$$

$$\Delta t \le \alpha_{\text{visc}} \frac{\rho \Delta x^2}{\eta}. \tag{2}$$

where K and G are the bulk and shear moduli

$$K = \frac{E}{3(1-2\nu)}, \qquad G = \frac{E}{2(1+\nu)},$$

and $\alpha_{\text{el}}, \alpha_{\text{visc}} \in (0, 1)$ are safety factors. We therefore choose

$$\Delta t \;\le\; \min\left(\alpha_{\text{el}} \frac{\Delta x}{c},\; \alpha_{\text{visc}} \frac{\rho \Delta x^2}{\eta}\right),$$

with sub-stepping applied when required (e.g., during curing or yielding events). In exchange for higher physical fidelity, MPM is compute and memory intensive and can exhibit grid-related artifacts if too few particles occupy a cell (see Table I).

2) Smoothed Particle Hydrodynamics (SPH): SPH is a particle-based, mesh-free Lagrangian method that models fluids by discretizing the material into moving particles. Each particle carries mass and state variables (e.g., position, velocity, density, pressure), and field values are estimated through weighted averages over neighboring particles within a smoothing radius h. Resolution is therefore set by the initial particle spacing and the *smoothing length* $\mathbf{h}$ (the typical number of neighbors). SPH works well mainly for the liquid phase and early deposition: it captures free-surface flow and coalescence well.

The simulation time step Δt is limited by wave speed and viscous constraints,

$$\Delta t \le C_{\text{CFL}} \frac{h}{c}, \tag{3}$$

$$\Delta t \le C_{\nu} \frac{h^2}{\nu}, \tag{4}$$

where c is the (numerical) speed of sound used in weakly compressible SPH, ν the kinematic viscosity, and $C_{\text{CFL}}, C_{\nu} \in (0, 1)$ are safety factors. In practice we set c sufficiently large to keep density variations small (weakly compressible assumption) and select Δt to satisfy both bounds.

3) Position-Based Dynamics (PBD): PBD advances particle positions by projecting them to satisfy constraints $C(\mathbf{x}) = 0$ each step. Resolution is set by particle spacing and the number of solver iterations per step. PBD's strengths are robustness and speed at larger time steps, which make it useful for real-time previews and interactive tuning. However, because it is constraint-driven rather than PDE-driven, its physical fidelity remains quite limited. We mainly treat it as a comparison method for qualitative behaviours (Table I).

4) Comparison between the 3 methods: MPM, SPH, and PBD each offer different balances of accuracy, stability, and speed (Table I). MPM combines particles with a background grid to give a continuum description of the material. This makes it well suited for capturing viscosity, elasticity, plasticity, and large deformations. Its main disadvantages are higher memory and compute cost, as well as possible grid artifacts if too few particles are used. SPH, on the other hand, represents fluids with particles that interact through smoothing kernels. It is very effective for free-flowing liquids but becomes less stable and more expensive when simulating highly viscous or non-Newtonian materials, often requiring very small time steps. PBD works by directly enforcing geometric constraints at each step. This makes it very fast and stable for large time steps, but it lacks physical accuracy and mainly produces visually plausible results.

For Aerial AM, we need to simulate a material that flows, expands, and then cures into a solid structure. This requires

correctly handling changes in viscosity, stiffness, and contact between layers. MPM is the best method for this purpose because it can naturally model curing as time-dependent material changes, conserve mass during expansion, and handle structural stability once layers are deposited. SPH would require very small time steps to remain stable in this setting, which would make it the most computationally expensive method, and it would still struggle to reproduce curing or elastic behavior in a realistic way. PBD could produce approximate shapes quickly, but it cannot predict important effects such as bead width, deformation under load, or collapse. Even though MPM is computationally demanding, it provides the level of accuracy needed to make reliable predictions for process design and control in Aerial AM.

C. Simulation platform: Genesis

In this work, the main simulation platform used is Genesis. Genesis is a recently developed (2024) physics simulation platform aimed at general-purpose robotics and embodied AI research. It provides a modular environment that integrates several physics solvers, including rigid body dynamics, the Material Point Method (MPM), Smoothed Particle Hydrodynamics (SPH), the finite element method (FEM), and others, allowing the simulation of rigid bodies, deformable materials, fluids, and articulated systems [9].

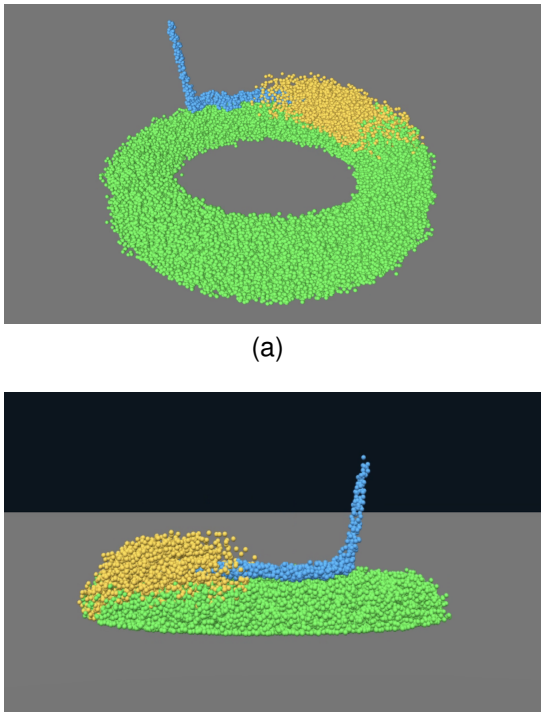
One of the main reasons for selecting Genesis is its native support for both MPM and SPH solvers within the same platform, which, in our research, is essential for comparing the two fluid simulation methods in the context of Aerial AM. Genesis is also designed to run efficiently on GPUs, enabling simulations at high speeds, especially when compared to other simulation platforms [9], which is particularly useful for reinforcement learning training and for running highly accurate, complex simulations.

Unlike some other platforms, such as Gazebo or Isaac Sim, Genesis does not offer a graphical user interface (GUI) for interacting with simulations. Instead, all simulations are set up and executed via Python scripts and command-line tools.

Genesis also plans to introduce a generative layer. A data engine and agent framework intended to turn natural-language prompts into simulation assets, tasks, scenes, camera motions, and even policy demonstrations. While the physics engine used here is open-sourced, access to the generative framework is planned to roll out gradually and was not available during this project; we therefore did not rely on it. If/when released, such tooling could accelerate environment and policy creation for Aerial AM by automating scene setup, reward functions, and large-scale data generation. [9]

D. Expansion and curing implementation

Both expansion and curing are modelled as time–scheduled, per–particle state transitions. When a particle is created (by emission or transition), its birth time is stored. At each simulation step, its age is computed, and once it exceeds a prescribed delay, the transition is triggered. To avoid exact synchronous behavior, which would be unrealistic, a slight random jitter is added to each particle's delay.



(a)

(b)

Fig. 3: Emission of a viscous fluid (blue) that expands (yellow) and cures (green) using the MPM method.

Curing is implemented as a one–to–one promotion of particles from a "fresh" pool of particles to a "cured" pool with different constitutive parameters (like viscosity, compressibility, etc). The particle's complete state (position, velocity, deformation history, etc.) is copied into the cured pool, and the original slot in the fresh pool is deactivated. In this way, momentum is preserved and no teleportation occurs. The cured pool can use increased viscosity, a non-zero yield stress, or even a completely different material law, such as an elasto–plastic model. To approximate continuous transitions more realistically, curing can also be performed in several stages with intermediate parameter values.

Expansion is instead a one–to–many process, where each particle is replaced by several particles in order to increase the simulated volume. For each particle that has reached its expansion delay, new particles are sampled inside a small sphere of radius proportional both to the particle size and to the expansion ratio, the radius is centered at the parent particle. The number of offspring is set by the target expansion ratio, with fractional parts handled probabilistically. Each new particle inherits the parent's velocity and is given a slight outward and random impulse to prevent static clumping and to provide the expansion plume-like motion. To conserve mass, the density of the expanded and downstream pools is scaled by the inverse of the expansion ratio. After duplication, the parent particle can be reused when needed.

Both curing and expansion can be applied incrementally in multiple passes. For curing, this means gradually changing material parameters over time, and for expansion, it means applying several smaller duplications rather than one large

jump. Randomness enters both in the transition times and in the spatial and kinematic perturbations of new particles, which reduces artifacts and mimics the probabilistic variability of real processes such as curing due to air or oxygen exposure.

All transitions use preallocated pools (carriers) of particles. When particles are promoted or expanded, their data are copied into these pools and activated with flags, so no new memory needs to be reallocated. Pool sizes are set large enough to accommodate the maximum expected particles, keeping curing and expansion scalable. An extended code implementation of the expansion and curing process is presented in Appendix IV-B.

Experimental protocol and metrics: We run three-dimensional simulations in Genesis (Section II-C) using the MPM solver (Section II-B). The domain is a bounded box with a rigid ground plane under uniform gravity. Before each run we fix: (i) the toolpath (circle center and radius), (ii) the number of layers and the pause between layers, (iii) nozzle standoff, tangential speed, and emission rate (chosen to yield a continuous, gap-free bead), and (iv) the material schedule: baseline fluid only, fluid + curing, fluid + expansion, or fluid + expansion + curing (Section II-D). If the robot model is included, its pose is kinematically attached to the toolpath with a fixed offset and a small clearance to avoid collisions, and its rigid physics can interact with the emitted particles.

For numerical settings we set the particle size and the background grid density, then choose the time step Δt and the number of substeps to satisfy stability constraints (Section II-B1).

During each run we log physical and computational measures. Physical measures include: bead width and layer height (sampled along the path), radial drift of the ring center, overall tower height, mass conservation (from total particle mass), and qualitative stability (no slump or collapse after the pause). When curing is enabled we also record the time to reach a target stiffness; when expansion is enabled we record the volume growth and the spread of offspring particles. Computational measures include: number of active and ever-activated particles, grid density (cells/m), Δt, substeps, wall time, frame rate (with and without rendering), and the Sim/Real time ratio. Success is defined as completing all layers with continuous beads, dimensional error within a small tolerance (height and radius), and a stable final geometry after the last pause. Results are summarized in Section III and in Table II.

III. Results

All simulations were executed with the Material Point Method (MPM) on a single RTX-5090. For strict comparability, the same numerical settings were used across scenarios: particle size $7 \times 10^{-3}\,\mathrm{m}$, time step $\Delta t = 10^{-3}\,\mathrm{s}$, 10 substeps, grid density $64\,\mathrm{cells/m}$, the same bounded domain of $1\,\mathrm{m}^3$, and—in most simulations—the same particle pool size. We deliberately fixed these to the values needed for the most demanding case ("(1) + Expansion + Curing + Drone" in Table II) so that performance differences reflect changes in modeled physics rather than solver retuning.

We use two performance metrics in Table II: frames per second (FPS), and the "Sim/Real time (s/s)" ratio, reported with and without rendering. The ratio grows as computations become more expensive (it indicates how many simulation seconds are needed per wall-clock second). Because FPS counts processed steps per second, it is useful when Δt and the number of substeps are fixed (as here), but it is not a good general estimator of throughput when numerics vary. Each scenario was run five times; we report the mean and standard deviation.

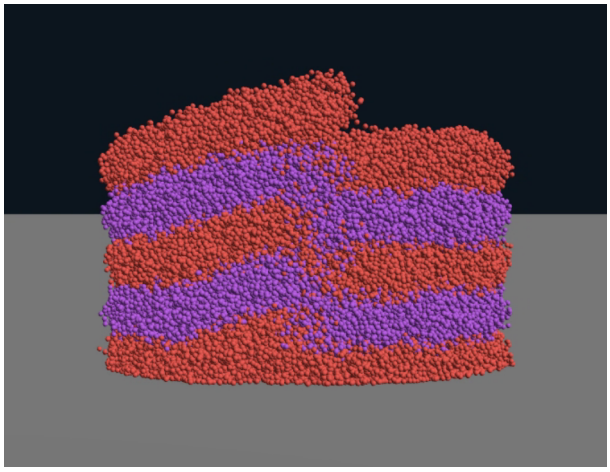


Fig. 4: Final cylindrical structure produced by depositing an expanding, curing foam. Alternating layer colours illustrate the layer-by-layer build-up.

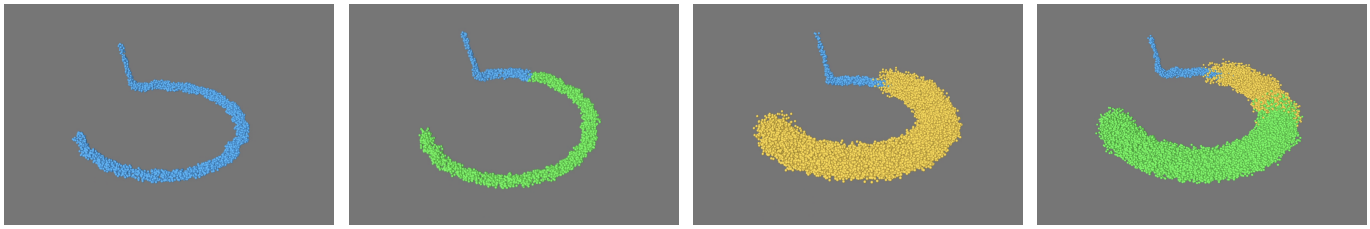
The baseline case, "(1): 5 layers printed structure," was run twice to isolate particle-count effects: once with the minimal particle pool (17k) and once with the worst-case pool (298k). With rendering enabled, increasing the pool roughly halves FPS (from ~ 64 to ~ 29) and about doubles the Sim/Real ratio (from ~ 16 to ~ 34 s/s). Headless, the change is negligible: FPS remains in the mid-80s and Sim/Real near 12 s/s. This contrast indicates that, at these settings, the renderer—not the solver—becomes the bottleneck as N grows.

We then add features incrementally. Relative to the large-N baseline, enabling curing causes only a small slowdown. Ten-times volumetric expansion is costlier, and combining expansion with curing increases cost further. Finally, adding the drone model produces the largest jump: rendered FPS drops into the high-teens and the Sim/Real ratio rises to ~ 60 s/s; headless stays around 60 FPS with Sim/Real near 17 s/s. This shows that extra scene geometry and interactions materially increase workload even without control logic.

In the resulting structure (Fig. 4), the layer-by-layer build-up appears realistic overall: rings coalesce, settle, and, after curing, support subsequent layers. At the same time, the structure is arguably "too precise" compared to field deposits: real foam application often shows uneven layer heights and slight tilts due to local over-fill, the way fresh material pushes and drags previously laid beads and its interaction with the surrounding air. In our model, volumetric growth is implemented by spawning particles around existing ones rather than by physically pushing neighbors outward; this simplifies mass conservation and stability but limits the emergence of the small leveling imperfections seen in practice. The small

Method	Simulation	# of Particles	Particle size (m)	Δt (s)	Substeps	FPS		Sim/Real time (s/s)	
						Rendering	No Rend.	Rendering	No Rend.
MPM	(1): 5 Layers Printed Structure	17k	7e-3	1e-3	10	63.7 $\pm$0.4	85.0 $\pm$0.9	15.7 $\pm$0.1	11.7 $\pm$0.2
	(1) + Curing	298k				29.2 $\pm$0.6	83.8 $\pm$0.6	34.2 $\pm$0.1	11.9 $\pm$0.1
	(1) + Expansion (x10)					29.2 $\pm$0.6	79.3 $\pm$0.7	34.2 $\pm$0.3	12.6 $\pm$0.2
	(1) + Expansion + Curing					27.0 $\pm$0.4	74.5 $\pm$0.7	37.0 $\pm$0.2	13.4 $\pm$0.1
	(1) + Expansion + Curing + Drone					25.8 $\pm$0.5	68.7 $\pm$0.5	38.8 $\pm$0.2	14.6 $\pm$0.1
						16.9 $\pm$0.3	60.0 $\pm$0.3	59.3 $\pm$0.2	16.7 $\pm$0.1

TABLE II: Comparison grid for MPM simulations in Genesis. Each simulation was repeated five times; the reported FPS and Sim/Real time values are given as averages with their corresponding standard deviations.



(a) Viscous emission (b) With curing (c) With expansion (d) Expansion + curing

Fig. 5: Genesis fluid emission comparison

"bumps" visible in the figure occur where each circular path starts and ends: by the time the ring closes, the material has already expanded, so the end segment deposits over partially swollen foam, creating a visible ridge. In future work, such artifacts could be reduced by adopting non-stop toolpaths or by simply avoiding full circular closures in the emission path.

Overall computational cost is driven by several coupled factors. Particle count N matters directly for rendering and indirectly for solver work. Grid density and domain volume set the background cost of particle–grid transfers and constitutive updates; with the grid fixed at $64\,\mathrm{cells/m}$, all scenarios pay the same grid bill. The time step Δt and the number of substeps primarily control stability and accuracy; the emission cadence is tied to Δt, whereas substeps refine the physics within each time step. Because we held these numerics fixed, Table II isolates the incremental cost of curing, expansion, and the drone.

Two practical takeaways follow. First, if throughput is the priority, disable rendering whenever possible: the headless rows in Table II are far less sensitive to N. Second, for fair scenario comparisons, keep numerics fixed; for production speed, tune per case, reduce the domain, lower grid density, increase Δt (within stability limits), and decrease substeps after verifying that no artifacts appear. These measurements are intentionally conservative because we locked numerics to the strictest stable settings across all cases; several scenarios could run stably with larger Δt, fewer substeps, or smaller domains, which would improve both FPS and the Sim/Real ratio. Further work will present per-scenario tuning studies to quantify sensitivity to Δt, grid density, and particle size.

IV. Conclusion

We introduced a physics-based simulation framework for Aerial Additive Manufacturing (Aerial AM) that combines viscoelastic fluid deposition with curing and volumetric expansion, running efficiently on the GPU-native Genesis engine. These process effects are implemented as scheduled particle transitions using preallocated pools, which makes the simulation stable even under large deformations. Among the fluid models taken into account, the Material Point Method (MPM) proved the most suitable for Aerial AM, as it can handle large shape changes, maintain printed layer stability, and predict structural response more reliably than SPH or PBD in our setting.

Our experiments, performed with fixed numerical settings for fair comparison, indicate that runtime is strongly influenced by particle count when rendering is enabled. In headless mode, however, the dominant factor is the added complexity of physical processes. Curing introduces only a modest computational overhead, expansion increases the cost more noticeably, and combining both with additional scene elements such as a drone results in the largest slowdown. Rendering itself has a substantial impact on performance, while headless runs remain stable and efficient, making them more suitable for large-scale data generation and reinforcement learning.

The simulator qualitatively reproduces expected Aerial AM behaviours, such as material settling, stiffening during curing, and volumetric growth during expansion. The expansion mechanism, however, is implemented through particle spawning around the existing fluid rather than by physically displacing material outward into surrounding air. Despite this simplification, the expanded medium subsequently exhibits realistic foam-like behavior, with visco-elastoplastic properties that are consistent with the intended material model.

Some limitations remain: simulations are still restricted to finite domains, aerial robots are only visualized rather than actively controlled, and the quantitative analysis focused primarily on MPM. Future work will integrate full drone dynamics and control, validate material parameters against experimental data, extend to larger-scale and multi-robot scenarios, and include a quantitative analysis of the framework's memory usage. In addition, the simulator will be configured

for training reinforcement learning policies for drone control. These steps aim to evolve the framework into a practical digital twin for Aerial AM, combining the accuracy required for structural prediction with the speed needed for learning and design exploration.

REFERENCES

[1] T. Wohlers and T. Campbell, *Wohlers Report 2015: 3D Printing and Additive Manufacturing State of the Industry*. Wohlers Associates, 2015.

[2] D. Floreano and H. H. Bülthoff, "Miniature flying robots: A new frontier," *Frontiers in Robotics and AI*, vol. 2, p. 18, 2015.

[3] K. Zhang, X. Li, P. Sharma, M. S. Anderson, and J. D. Brown, "Aerial additive manufacturing with multiple autonomous robots," *Nature*, vol. 609, no. 7928, pp. 709–717, 2022.

[4] Y. F. Kaya, A. Müller, R. K. Katzschmann, and M. Kovac, "Aerial additive manufacturing: Toward on-site building construction with aerial robots," *Science Robotics*, vol. 10, no. 101, p. eado6251, 2025.

[5] U. Berdica, E. Romano, L. P. Roversi, and G. Fiorentini, "Mobile 3d printing robot simulation with viscoelastic fluids," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Prague, Czech Republic, 2021, pp. 1234–1241.

[6] E. Angelidis, S. Petrova, and J. Schneider, "A smoothed particle hydrodynamics framework for fluid simulation in robotics," *Robotics and Autonomous Systems*, vol. 185, p. 104885, 2025.

[7] A. Sanchez-Gonzalez, J. Puig, P. Battaglia, R. Yue, J. T. Kipf, M. W. T. Michael, and P. Blunsom, "Learning to simulate complex physics with graph networks," arXiv:2002.09405, 2020.

[8] T. Pfaff, N. Gosala, G. Berseth, K. Hausman, and T. Lillicrap, "Learning mesh-based simulation with graph networks," arXiv:2010.03409, 2020.

[9] G. Authors, "Genesis: A universal and generative physics engine for robotics and beyond," December 2024. [Online]. Available: https://github.com/Genesis-Embodied-AI/Genesis

APPENDIX

A. Simulator platform selection: methodology and rationale

Criterion	Genesis	Isaac Sim	Gazebo
Fluid models (native)	PBD, SPH, MPM	PBD	None
GPU acceleration	Yes (all GPUs)	Yes (CUDA)	No, CPU only
RL training	Parallel envs; fewer tools	Parallel envs; rich RL/DR/ROS2	CPU-bound; slow setup
OS support	Linux, Windows & MacOS	Linux	Linux & Windows (WSL2)
API / tooling	Python-first, headless	Python + Omniverse/USD	Strong ROS/ROS2, C++/Python

TABLE III: Simulation platforms comparison.

Objective: Select a simulator that can (i) represent viscous/viscoelastic, expanding deposition with curing, (ii) run efficiently on GPU for parallel rollouts, and (iii) be scripted headlessly for reproducible experiments and potential RL training.

Stage A — From longlist to shortlist (qualitative screen): We began with a broad survey of robotics/graphics simulators: Isaac Sim, Gazebo, Mujoco, Genesis, Project Chrono, Webots, Unity, CoppeliaSim and RoboDK. We then reduced the set to three candidates based on native fluid models, GPU use, RL tooling, OS support, and scripting (see Table III): *Genesis*, *Isaac Sim*, and *Gazebo*. Genesis was included for its native SPH/MPM alongside PBD; Isaac Sim for its strong ecosystem (Omniverse/USD, PhysX, RL/ROS2 tools); Gazebo for its ROS/ROS2 integration despite relying on plugins for fluids.

Stage B — Hands-on tests (basic viscous extrusion): We then implemented the simplest viscous deposition scenario in the candidates we could exercise quickly:

- **Isaac Sim**: native PBD/PBF fluids. We obtained stable, real-time behavior suitable for previews; however, the constraint-based model remained qualitatively plausible rather than quantitatively faithful for dense or viscoelastic materials, and tuning cohesion/viscosity did not yield the curing/structural response we require. Stability with high viscosity using PBD also turned out to be problematic.
- **Genesis**: SPH and MPM minimal setups. SPH captured free-surface flow and coalescence for early deposition; MPM, with a viscoelastic (and optional elastoplastic) update and a curing schedule, maintained load-bearing layers and post-deposition stability in our tower tests.
- **Gazebo**: excluded mainly due to lack of native GPU-accelerated fluids and CPU-bound performance for particle methods, which would hinder large batches and RL-scale experiments (Table III).

Stage C — Small cluster comparison (matched scenarios): We ran a small matched comparison on our compute node: a 5-layer ring tower, then the same tower with expansion and curing toggled. Metrics included wall-time/FPS, particle/grid resolution limits before instability, and reproducibility (seeded runs). Isaac Sim offered strong tooling and visualization but its fluid model (PBD/PBF) was the limiting factor for viscoelastic accuracy. Genesis achieved higher effective throughput at comparable fidelity for our use case by allowing us to choose `MPM.Liquid` for fast liquid deposition tests and `MPM.Elastoplastic` for cured, load-bearing behavior, both in headless GPU batches.

Decision and rationale: We continued with **Genesis** as the primary platform because it (i) provides native *SPH and MPM* in addition to PBD, (ii) runs headless on GPU with parallel environments, (iii) exposes a Python-first API suitable for pipelines and RL, and (iv) reached the fidelity/performance trade-off we needed for viscoelastic, expanding deposition with curing. Isaac Sim and Gazebo remain valuable in other settings (visualization/Omniverse tooling and ROS/ROS2 integration, respectively), but for this project their fluid modeling (Isaac Sim) and GPU fluid throughput (Gazebo) were not aligned with our objectives. Consequently, all experiments in the main paper are conducted in Genesis; Section II-C details versions, settings, and hardware.

B. Implementation of expansion and curing

This simulation uses three preallocated *carriers*, each backed by a circular buffer: `P0_emit` (fresh droplets), `P0_expanded` (after expansion), and `P1_viscous` (after curing). We store, for every slot, the simulation step when a particle entered a carrier in the arrays `birth_emit` and `birth_expanded`. Small writer helpers insert contiguous blocks into a carrier and advance the head indices (for example `emit_head` and `expanded_head`). Two promotion functions then move particles forward in the pipeline when they are old enough.

Timing and jitter. At each step we compute the particle age in a carrier as `(step - birth) * dt`. A particle is eligible for the next stage when its age is greater than a user time constant plus optional random jitter:

- expansion wait: `TAU_EXPAND_S` with optional `TAU_EXPAND_JITTER`;
- curing wait: `TAU_TO_P1_S` with optional `TAU_TO_P1_JITTER`.

We build a boolean mask of eligible and currently active particles and collect their indices for promotion. This keeps the logic simple and fast.

Expansion (`promote_expand`). Expansion models controlled volume growth. For each eligible parent in `P0_emit`, we create K children written to `P0_expanded`. The child count K comes from `EXPANSION_RATIO`: we take its integer part and add one extra child with probability equal to the fractional part. New child positions are sampled inside a ball of radius `R_samp = EXPAND_RADIUS_MULTIPLIER * P_SIZE` centered at the parent. We keep only samples that:

1) remain inside the simulation bounds (`in_bounds_mask`);
2) stay above the ground by at least `GROUND_Z + COLLISION_MARGIN`.

Child velocities are the parent velocity plus a radial impulse and a small random kick:

```
v_child = v_parent + Vexp *
dir_blend + EXPANSION_EXTRA_NOISE *
noise
```

Here `Vexp` is sampled around `EXPANSION_VEL` with jitter `EXPANSION_VEL_JITTER`, and `dir_blend` mixes the normalized offset direction with a random unit direction using `VEL_RADIAL_WEIGHT`. We write the resulting blocks into `P0_expanded` via the circular buffer and stamp `birth_expanded = step` for the inserted range. We then deactivate the promoted parents in `P0_emit` and reset their `birth_emit` entries to `-1`. To approximately conserve mass when the particle count increases, the expanded carrier can use a reduced density `rho_expanded = RHO_P0 / EXPANSION_RATIO` when `MASS_COMPENSATE` is enabled.

Curing (`promote_to_p1`). Curing changes the material model after a waiting time. Eligible particles in `P0_expanded` are copied into `P1_viscous` with the same positions and velocities. The change in behavior comes from the target carrier using a different constitutive law (for example `ElastoPlastic` with parameters `E` and `nu`). As in expansion, we insert a contiguous block into the circular buffer, update counters, deactivate the sources, and clear their `birth_expanded` entries. This simple handoff captures stiffening after deposition without reinitializing state.

Capacity and safety. Carrier sizes are chosen before the run using `estimate_counts_sphere_per_step`, which predicts peak occupancies from Δt, total duration, droplet size, particle size, the expansion ratio, and two safety factors (packing efficiency and headroom). The circular writer `_push_block` performs wrap-around writes and calls small "stamp" callbacks to record `birth_*` and activation counters. Finally, domain bounds, ground clearance, and a light linear drag (`DRAG_LINEAR`) help keep the expanded population stable under large motion.