Planning in Dynamic Environments: A Comparative Study of A* and RRT* Replanning in a 3D Fluid Particle Simulation

Anthony Angeles, Xuanyu Zhu
Santa Clara University

***Abstract*— Autonomous robots must navigate environments that are dynamic and unpredictable, requiring frequent replanning under strict time constraints. This problem lies at the intersection of motion planning, real-time computation, and control theory, where planners must balance responsiveness with path stability. A planner that is too slow to recompute fails to avoid collision, while one that updates too frequently produces unstable behavior. We present a 3D path-planning simulator in which a robot navigates a dense, continuously evolving Smoothed Particle Hydrodynamics (SPH) fluid environment, augmented with static and dynamic obstacles. The system performs continuous replanning using either A* (grid-based) or RRT* (sampling-based) on a live occupancy representation.**

Across 1,142 benchmark trials spanning varying particle densities and obstacle configurations, RRT* consistently outperformed A* on all primary metrics. RRT* achieved 2–3× faster time to goal, and 10–24× lower per-replan latency, while maintaining comparable path quality and stability. In contrast, A* frequently failed to find valid paths in dense environments due to excessive grid occupancy, leading to high failure rates and large latency spikes.

These results highlight that in highly dynamic, high-density environments, planners with bounded computation time (RRT*) can outperform optimal grid-based methods (A*), where responsiveness is more critical than per-plan optimality.

I. Introduction

Planning in dynamic environments remains a central challenge in robotics, where obstacles may move, appear, or disappear over time. In such settings, a single precomputed path is often insufficient, and robots must continuously replan in response to environment changes. Planners need to trade off the balance between responsiveness and stability: react quickly while maintaining consistent, usable trajectories.

In this work, we study this trade-off in dense, continuously evolving environments. We built a 3D robot path-planning simulator called `plannersim` in which a robot navigates from a random start position to a goal inside a bounded volumn filled with a live Smoothed Particle Hydrodynamics (SPH) fluid. The fluid particles act as dynamic obstacle whose positions change every simulation frame under the forces of pressure, viscosity, and gravity. In addition to this particle-based noise, we introduce two types of structured obstacles: static axis-aligned boxes and dynamic boxes that move and bounce within the environment.

The planner is required to continuously recompute a collision-free path from the robot's current position to the goal, updating its stored path whenever a new plan is ready. We implemented two canonical path-planning algorithms for comparison: A* (a grid-based, deterministic algorithm that is optimal within the discretized space) and RRT* (a sampling-based algorithm that provides asymptotic optimality with bounded per-call computation). Both planners run against the same live occupancy map derived from particle positions and obstacle geometry at each replanning call.

While A* and RRT* have been extensively studied in static and moderately dynamic settings, their performance in highly dense, rapidly evolving particle-based environments remains less explored. In such conditions, the performance of grid-based planners such as A* may become unpredictable when large portions of the state space are occupied, while sampling-based methods may better adapt to irregular and transient free space. This motivates the question: how do these planners compare when frequent replanning is required in a dense, dynamic environment?

We evaluate both algorithms across three scenarios—particle-only, static obstacles, and dynamic obstacles—while varying particle density from sparse to highly congested conditions. We measure performance using time to goal, planning latency, success rate, and path consistency.

Our central questions are: (1) Which algorithm achieves faster replanning, and does this translate to faster goal arrival? (2) How does each algorithm behave as the environment complexity increases (more particles, moving obstacles)? (3) How consistent are the paths produced across successive replanning steps?

Across 1,142 benchmark trials, we find that RRT* outperforms A* on nearly every metric: it is 10–24× faster per planning call, reaches the goal roughly 3× sooner on average, and maintains comparable path consistency. Counterintuitively, A*'s grid-based approach is crippled in this domain by the density of the particle cloud, which causes the majority of replan attempts to return no path. In contrast, RRT*'s fixed iteration budget provides stable and predictable replanning behavior.

II. Related Work

A. Smoothed Particle Hydrodynamics

SPH, introduced by Lucy (1977) and Gingold & Monaghan (1977), approximates a continuous fluid using discrete interacting particles. Each particle's density and pressure are computed via a kernel-weighted sum over neighbors within a smoothing radius H. Force contributions include a pressure gradient term (Spiky kernel) and a viscosity term (Viscosity Laplacian kernel). Modern robotics uses SPH-like representations for crowd simulation, deformable-terrain

navigation, and fluid obstacle modeling [1]. While SPH has been used to model fluids and deformable environments, its use as a dynamic obstacle field for evaluating replanning algorithms remains relatively unexplored.

B. A* in Dynamic Environments

A* [2] is a best-first search on a graph that combines the cost-to-come g with a heuristic estimate h of cost-to-go. In static environments it is optimal and complete. For dynamic environments, the standard approach is to re-invoke A* from scratch whenever the map changes, sometimes called Replanning A*. More sophisticated variants such as D* Lite [3] incrementally repair the previous plan rather than rebuilding from scratch, reducing computation when only local changes occur. Lu et al. [4] formally characterized this worst-case sensitivity of LPA*, the incremental A* variant, showing that vertex expansions can spike dramatically when a blocked cell creates a local dead-end near the source, and proposed a multiscale quadtree decomposition (m-LPA*) that reduces worst-case vertex expansions from $O(n^2)$ to $O(n)$. Our implementation uses straight replanning A*. However, these approaches typically assume structured or sparsely changing environments. Their behavior in dense, rapidly changing occupancy fields remains less well explored.

C. RRT and RRT*

The Rapidly-exploring Random Tree [5] grows a tree of collision-free configurations by randomly sampling the configuration space and steering the nearest tree node toward each sample. RRT* [6] adds a rewiring step that guarantees asymptotic optimality: as the number of samples grows, the solution cost converges to the optimum. In dynamic environments, RRT* is particularly attractive because its random sampling naturally handles irregular obstacle shapes and the computational cost per call is bounded by a fixed iteration budget, which makes its latency predictable. While RRT* is well suited for continuous and high-dimensional spaces, its performance under frequent replanning in dense dynamic environments has not been extensively evaluated.

D. Replanning Strategies

Anytime algorithms [7] return a best-effort solution quickly and refine it over time. Our system uses a simpler model: a background planning thread is triggered every 10 simulation frames, and the main simulation thread swaps in a newly computed path whenever one becomes ready (double-buffered path exchange using atomic flags). This sllows us to directly compare algorithm latency and responsiveness under identical conditions.

III. Problem Definition

A. Environment

The world is a closed cubic box of side length L (default $L = 5.0$ world units at the baseline of 1,500 particles, scaled as $L = 5.0 \cdot \sqrt[3]{N/1500}$ so that particle density remains constant as N grows). The interior is populated with N fluid particles simulated by SPH at a fixed timestep of $\Delta t = 0.02\,\mathrm{s}$.

The SPH particles act as dynamic obstacles whose positions evolve continuously under physical forces including pressure, viscosity, and gravity. Besides, we consider two forms of structured geometry: axis-aligned static boxes and dynamic boxes that move with fixed velocities and reflect off environment boundaries.

We evaluate three environment configurations:

- **Scenario 1 – Particles Only.** The only obstacles are the SPH fluid particles themselves. This allows for simulated noise to act as if the robot were planning in a crowded environments with pedestrians and vehicles.
- **Scenario 2 – Static Obstacles.** In addition to fluid particles, a set of axis-aligned box obstacles is placed at fixed positions.
- **Scenario 3 – Dynamic Obstacles.** Same as Scenario 2, but the box obstacles translate and bounce off the environment boundaries with fixed velocity vectors.

These scenarios progressively increase environmental complexity, from purely stochastic motion to structured and adversarial obstacle dynamics.

B. Occupancy Model

A uniform grid of cubic cells (cell size 0.2 wu) is overlaid on the environment. For A*, a grid cell is marked *occupied* if: - any SPH particle lies within a radius of $0.7\times$pathCellSize $= 0.14$ wu from the cell center, or - an obstacle intersects the cell. This effectively introduces a conservative obstacle inflation, causing obstacles to appear larger in the discretized grid. As a result, A* tends to favor paths that remain centered within free space rather than closely following obstacle boundaries. For RRT*, collision checks on continuous space using a tighter radius of $2 \times$ particleRad $= 0.10$ wu (`isOccupiedWorld`).

Thus, the two planners operate under different collision-checking resolutions: A* uses a conservative wider radius with obstacle inflation, while RRT* evaluates collisions in continuous space with a tighter radius. This reflects inherent differences between grid-based and sampling-based planners, where grid methods often incorporate safety margins through discretization.

C. Robot and Goal

Two particles are designated as *robot* and *goal*. They are selected randomly after an initial settling phase, with a minimum separation of $0.5 \times L$. The robot particle is steered toward its current waypoint using a proportional velocity controller (`robotSteerForce` = 50, `robotMaxSpeed` = 2.0). The goal is considered reached when the robot is within $4\times$ the particle radius. We distinguish the two by using the colors green and red to mark them clearly in a pool of white particles.

D. Replanning and Metrics

A background thread invokes the path planner every 10 simulation frames. An atomic flag (`g_plannerRunning`)

TABLE I

METRICS OF INTEREST

Metric	Description
Time to goal (s)	Wall-clock time until the goal is reached
Avg. planning time (ms)	Mean latency per replanning call
Total replans	Total number of replanning calls per run
Successful replans	Number of calls returning a valid path
Failed replan rate	Fraction of replanning calls that return no path
Path divergence	Mean distance between consecutive planned paths
Avg. path length	Mean Euclidean path length

ensures that a new invocation is not launched until the previous one has completed. Each invocation reads the current occupancy map and returns a waypoint sequence from the robot's current position to the goal.

Metrics recorded per invocation include: planning time (ms), path length (world units), number of waypoints, path divergence from the previous plan, and whether a valid path was found. The primary metrics of interest are summarized in Table I.

IV. APPROACH

A. A* (Grid-Based Replanning)

The planning grid has dimensions $\texttt{pathGridDim}^3$, where $\texttt{pathGridDim} = \max(25, \lfloor L/0.2 \rfloor)$ scales with the box size. At the 1,500-particle baseline this yields $25^3 = 15{,}625$ cells; at 25,000 particles ($L \approx 12.75$ wu) it reaches $63^3 = 250{,}047$ cells—a $16\times$ increase that compounds A*'s latency at high particle counts.

A* is run with full 26-connectivity (face, edge, and corner neighbors). Move cost is Euclidean distance between adjacent cell centers (1.0 for face, $\sqrt{2}$ for edge, $\sqrt{3}$ for corner). The heuristic is the Euclidean distance to the goal cell.

After a path is found, a greedy path smoother post-processes the waypoint list: it performs iterative visibility tests to skip intermediate waypoints connected by a clear line-of-sight segment, reducing waypoint count and smoothing robot motion. Because A* is deterministic, it always returns the shortest path within the current occupancy snapshot (subject to grid resolution). However, it must search the entire free space between start and goal, making it sensitive to the density of occupied cells.

Listing 1 shows the inner neighbor-expansion loop. The triple $[-1, 1]$ loop generates all 26 surrounding voxels; the $(0, 0, 0)$ case is pruned by the closed-set check. Move cost is computed as the exact Euclidean distance between voxel centers (1, $\sqrt{2}$, or $\sqrt{3}$), so the g-score reflects true Euclidean distance rather than hop count.

```
for (int dx = -1; dx <= 1; dx++) {
 for (int dy = -1; dy <= 1; dy++) {
  for (int dz = -1; dz <= 1; dz++) {
    int nx = current.x+dx, ny = current.y+dy,
        nz = current.z+dz;
    if (nx<0||nx>=pathGridDim||...) continue;
    if (isOccupied(simState, nx, ny, nz))
      continue;
    // 1, sqrt(2), or sqrt(3)
    float moveCost =
      sqrt(dx*dx + dy*dy + dz*dz);
    float tentativeG =
      gScores[currentTuple] + moveCost;
    ...
  }
 }
}
```

Listing 1. A* 26-connected neighbor expansion (`pathFinding.cpp`)

B. RRT* (Sampling-Based Replanning)

Each RRT* call runs for a fixed budget of 2,000 iterations. Key parameters: step size 0.3 wu, goal attraction probability 10%, rewire radius 0.6 wu. A KD-tree over the current tree nodes (rebuilt every 50 new nodes) accelerates nearest-neighbor queries from $O(n)$ to $O(\log n)$.

When a node is added, all existing nodes within the rewire radius are checked; if routing through the new node reduces their cost, their parent pointer is updated. Goal detection uses a proximity threshold of 0.3 wu. The resulting path is also passed through the same greedy smoother as A*. Because the iteration budget is fixed, RRT*'s per-call latency is bounded regardless of environment complexity, giving it a predictable real-time profile.

Listing 2 shows the two-pass parent-selection and rewiring procedure. The first pass selects the minimum-cost parent among all near nodes; the second re-parents any near node whose cost decreases by routing through the new node. Both passes gate on `isPathClear` to reject collision-blocked edges.

```
// Pass 1: choose lowest-cost parent
int bestParent = nearestIndex;
float bestCost = tree[nearestIndex].cost
               + distance3D(..., newX,newY,newZ);
for (int idx : nearNodes) {
  float c = tree[idx].cost
          + distance3D(tree[idx]..., newX,newY,newZ);
  if (c < bestCost &&
      isPathClear(simState, tree[idx]..., newX,...))
  { bestParent = idx; bestCost = c; }
}
tree.push_back(
  RRTNode(newX,newY,newZ,bestCost,bestParent));

// Pass 2: rewire near nodes through new node
for (int idx : nearNodes) {
  float rc = bestCost
    + distance3D(newX,newY,newZ, tree[idx]...);
  if (rc < tree[idx].cost &&
      isPathClear(simState, newX,..., tree[idx]...))
  { tree[idx].parent = newNodeIndex;
    tree[idx].cost   = rc; }
}
```

Listing 2. RRT* parent selection and rewiring (`pathFinding.cpp`)

C. Replanning Architecture

Planning runs in a dedicated background thread decoupled from the simulation loop. Three shared variables coordinate the exchange: `g_plannerRunning` (atomic bool) prevents concurrent planner invocations; `g_pendingPath` (vector, mutex-guarded) is a staging buffer for the new path; and `g_pathReady` (atomic bool) signals the main thread to swap in the new path. The main thread checks `g_pathReady` each simulation frame and, if set, atomically replaces `simState.currentPath` with `g_pendingPath`.

Listing 3 shows the handoff. The planner thread writes its result under `g_pathMutex` and sets `g_pathReady`; the main thread performs a single `exchange(false)` — if it wins the swap it takes the lock once to replace `currentPath` and append the replan event to the metrics log. There is no busy-wait and no double-lock.

```
// Main / idle thread -- called each frame
if (g_pathReady.exchange(false)) {
  std::lock_guard<std::mutex> lk(g_pathMutex);
  if (g_pendingEvent.pathFound) {
    simState.currentPath = g_pendingPath;
    simState.pathUpdateCounter++;
  }
  simState.replanLog.push_back(g_pendingEvent);
}
```

Listing 3. Lock-free path handoff between planner and simulation threads (`main.cpp`)

V. Implementation Details

The simulator is implemented in C++17 with OpenGL/GLUT rendering. CUDA acceleration is optionally available for the SPH physics pipeline, enabled via the `USE_CUDA` compile flag. Five kernels run back-to-back each frame, one thread per particle: `computeCellIdx` assigns particles to grid cells; a Thrust radix sort reorders particles by cell so that `buildCellRanges` can mark contiguous `[cellStart, cellEnd)` ranges; `computeDensityPressure` and `computeForces` then walk those ranges to accumulate SPH kernel contributions with coalesced global-memory access; and `integrate` applies the Euler step and boundary reflection. Particle state is held in pinned (`cudaMallocHost`) host buffers to maximise DMA throughput on the per-frame host↔device transfer. The pathfinder, robot controller, and obstacle logic remain on the CPU; only the $O(N{\cdot}k)$ neighbour kernel is offloaded. The CPU-only binary (`plannersim-cpu`) was used for all benchmarked results to ensure reproducibility across machines without GPU access.

SPH parameters: smoothing radius $H = 0.15$, particle mass $= 0.05$, rest density $\rho_0 = 1000$, gas constant $k = 2000$, viscosity $\mu = 0.1$. SPH neighbors are found via a spatial hash grid (cell size $= H$), searching the 27 neighboring cells of each particle's cell.

Spatial grid: A hash-based spatial grid (cell size $H = 0.15$) partitions all particles into a flat array of buckets. Each particle is assigned to a bucket via:

```
int gridHash(int x, int y, int z) {
  return (x * 92837111)
       ^ (y * 689287499)
       ^ (z * 283923481);
}
```

Listing 4. Spatial hash function (`particle.cpp`)

Each axis coordinate is multiplied by a distinct large prime, spreading its bits across the full 32-bit range before XOR combination. XOR mixes without carries, so two cells that share one or two coordinates map to distant buckets rather than clustering. Neighbour lookup checks only the 27 cells within ± 1 in each axis; since cell size equals H, no particle outside those 27 cells can contribute to the kernel. Each insertion and lookup is therefore a single multiply-XOR-mod: $O(1)$ average per particle, making the per-frame SPH step scale approximately linearly with N.

Benchmark: A shell script (`benchmark.sh`) sweeps particle counts from 500 to 25,000 in steps of 1,500 (i.e. $500, 2000, 3500, \ldots, 24500, 25000$ — 18 counts), across all three scenarios and both algorithms, running 5 headless trials at each combination: $18 \times 3 \times 2 \times 5 = 540$ runs per full sweep. The 1,500-particle step matches the simulator's default particle count, so each increment on the scaling plots corresponds to one additional "base configuration" worth of fluid. Results are appended to `results.csv` (14 columns) and analyzed with `analyse.py` (matplotlib/pandas), which generates bar charts, box plots, and the scaling curves in Fig. 3.

VI. Results

All numbers below are from successful runs (goal reached) across 1,142 trials in the benchmark.

A. Time to Goal

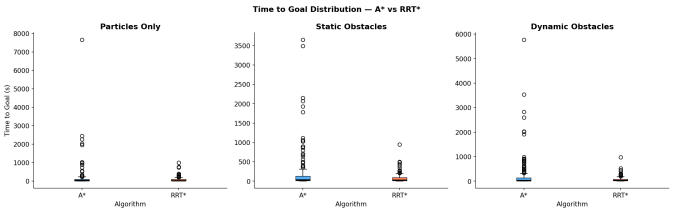

Fig. 1. Time-to-goal distributions (s) for A* and RRT* across all three scenarios. A*'s interquartile range is far wider and its outliers extend to 7,600 s, while RRT*'s distribution remains compact in all conditions, confirming substantially lower variance in navigation time.

TABLE II
Time to Goal by Scenario

Scenario	A* (s)	RRT* (s)	Speedup
Particles Only	171.85 ± 654.67	73.12 ± 123.39	$2.35\times$
Static Obstacles	198.10 ± 496.60	69.03 ± 107.85	$2.87\times$
Dynamic Obstacles	205.97 ± 601.20	67.67 ± 104.94	$3.04\times$

RRT* consistently reaches the goal 2–3× faster than A*. Notably, A*'s standard deviation exceeds its mean in all three scenarios, indicating occasional runs with catastrophically long navigation times. RRT* is substantially more predictable.

B. Planning Time

TABLE III
Average Planning Time per Replan Call

Scenario	A* (ms)	RRT* (ms)	Ratio
Particles Only	466.2	40.6	$11.5\times$
Static Obstacles	569.9	47.9	$11.9\times$
Dynamic Obstacles	1159.3	47.8	$24.3\times$

A*'s planning time grows sharply from Scenario 1 to Scenario 3 ($2.5\times$), as moving obstacles fragment the free space

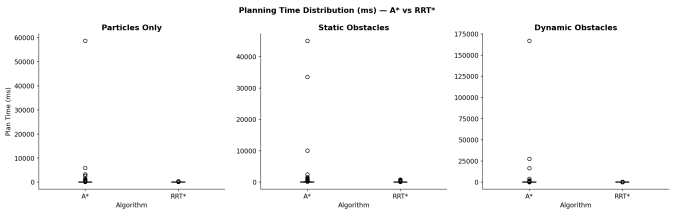

Fig. 2. Planning time distributions (ms) for A* and RRT* across all three scenarios. Note the differing y-axis scales: A*'s worst-case latency reaches ∼302,000 ms in the dynamic-obstacles scenario, while RRT* never exceeds 2,072 ms.

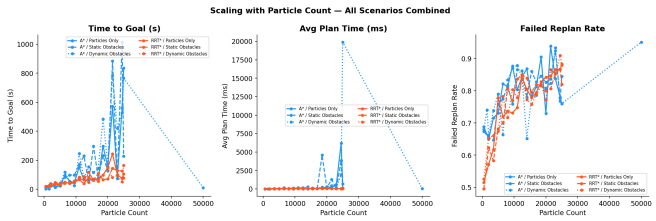

Fig. 3. Scaling behavior with particle count across all scenarios. Left: time to goal (s). Center: average planning time (ms). Right: failed replan rate. Blue lines are A*, orange are RRT*. A*'s planning time spikes sharply above 18,000 particles while RRT*'s remains flat, and both algorithms' failure rates converge toward 1.0 at high particle counts.

and force A* to explore larger regions. RRT*'s planning time is essentially constant across scenarios (40–48 ms)—a direct consequence of its fixed iteration budget.

C. Scaling with Particle Count

Fig. 3 shows how time to goal, average planning time, and failed replan rate evolve as particle count increases from 500 to 25,000 across all three scenarios.

Time to goal grows monotonically for A* as particle count increases, with a sharp spike near 20,000–25,000 particles before dropping. This drop does not reflect improved performance: at these densities A* so frequently fails to find any path that the robot makes little forward progress, and the few successful runs that complete take extremely long. Beyond 25,000 particles A* succeeds so rarely that the mean is computed over a very small sample and is statistically unreliable. RRT* time to goal grows modestly and remains well below A* at all particle counts tested.

Average planning time for A* remains near zero for low particle counts (the environment is sparse enough that searches terminate quickly) but spikes dramatically above 18,000 particles, reaching over 20,000 ms per call in the dynamic-obstacles scenario. RRT*'s planning time is essentially flat across all particle counts, confirming that its fixed 2,000-iteration budget decouples latency from environment density.

Failed replan rate rises for both algorithms as particle count increases, reflecting the growing fraction of occupied path-grid cells. A*'s failure rate tends to be slightly higher than RRT*'s at moderate particle counts, but both converge toward near-total failure (>90%) above 20,000 particles. Crucially, A* pays a much higher latency cost for each failed attempt than RRT* does.

D. Replanning Activity

Both algorithms trigger a similar number of total replanning calls per run (∼105–130, measured as `replanLog.size()`), with RRT* slightly higher. Successful replans (calls that returned a valid path, tracked separately as `pathUpdateCounter`) are a small fraction of this total for both algorithms. The failed replan rate is high for both (∼76–79%), meaning the majority of replan calls return no valid path. This reflects the dense SPH particle cloud: at typical particle counts, most of the path-cell space is occluded. A* responds to failure with a complete grid search (all nodes expanded until the open set is exhausted), explaining the much longer planning times. RRT* terminates after 2,000 iterations regardless, keeping latency bounded.

E. Path Quality

TABLE IV
PATH LENGTH AND DIVERGENCE BY SCENARIO

Scenario	A* len	RRT* len	A* div	RRT* div
Particles Only	0.789	0.755	0.205	0.177
Static Obstacles	0.759	0.757	0.148	0.185
Dynamic Obstacles	0.815	0.662	0.168	0.180

Note: `avgPathLength` is averaged over all replan calls including failed ones (which contribute zero length), so values reflect mean path length weighted by success rate (≈23%), not the mean length of a successfully found path. The latter is approximately 3.4 wu for A* and 3.3 wu for RRT* in the particles-only scenario.

Path lengths are similar between the two algorithms. Path divergence (mean nearest-distance between consecutive plans) is also comparable (0.15–0.21), suggesting both algorithms produce paths of similar spatial stability. A* shows marginally lower divergence in static scenarios, as its deterministic grid search returns the same shortest path when the environment has not changed. RRT*'s divergence is slightly higher because each call produces a new random tree.

VII. DISCUSSION

Why does A* take so long? A key observation is that A* incurs significantly higher latency in dense particle environments. The SPH fluid creates a highly occluded environment. When A* cannot find a path (failed replan), it exhausts its entire open set before returning failure. Because `pathGridDim` scales with box size, the grid the search must exhaust grows from $25^3 = 15{,}625$ cells at 1,500 particles to $63^3 = 250{,}047$ cells at 25,000 particles, leading to a substantial increase in worst-case search effort. This explains the large latency spikes observed at high particle densities (10,000+ particles). This behavior is consistent with the worst-case analysis of Lu et al. [4], who showed that LPA* vertex expansions can blow up when blocked cells create local dead-ends near the source—the same failure

mode observed here. In contrasrt, RRT* avoids this problem because it operates under a fixed iteration and terminates after 2,000 samples (∼48 ms) regardless of whether a path exists. As a result, its latency remains stable even when the environment becomes highly cluttered.

Why is the failed replan rate so high for both? The SPH particles and obstacles collectively block a large fraction of path-grid cells at any given frame. The robot's current position (a particle itself) may temporarily be surrounded by neighbors, making even short-range paths infeasible. This is an inherent property of using a particle-based fluid as the obstacle representation. It should also be noted that the two algorithms use different occupancy check radii (0.14 wu for A*, 0.10 wu for RRT*), meaning A* operates against a more conservative obstacle map, which increases the number of blocked cells and further reduces feasible free space. This leads to a higher failure rate compared to RRT*, which performs collision checking in continuous space with a tighter radius.

Path consistency. Despite the high replanning rate, path divergence is modest (∼0.15–0.20 wu). This suggests both planners tend to rediscover geometrically similar routes, even when the exact waypoints shift. Although individual particle positions change each frame, the fluid's macroscopic density field—and therefore the large-scale topology of free corridors—evolves slowly relative to the replanning rate. A*'s slightly lower divergence in static scenarios reflects its determinism: when the occupancy map is unchanged between two replan calls, A* returns bit-for-bit identical paths.

Winner per Metric and Scenario (blue = A* | orange = RRT*)

	Time to Goal	Plan Time	Failed Replan Rate	Path Length	Path Divergence
Particles Only	RRT*	RRT*	RRT*	A*	RRT*
Static Obstacles	RRT*	RRT*	RRT*	A*	A*
Dynamic Obstacles	RRT*	RRT*	RRT*	A*	A*

Fig. 4. Per-metric winner across all scenarios (blue = A*, orange = RRT*). RRT* wins on the three latency-sensitive metrics in every scenario; A* produces marginally shorter paths in all cases but at the cost of substantially longer navigation times overall.

Trade-off summary. These results highlight a fundamental trade-off in dynamic path planning. Grid-based planners such as A* offers theoretical grid-optimality and deterministic behavior, but its latency is unbounded and grows with obstacle density. RRT* sacrifices optimality guarantees within a single call but provides bounded, consistent latency—which in a dynamic environment is arguably more valuable than per-call optimality.

Limitations. The robot uses a simple proportional velocity controller; a more sophisticated model-predictive controller would better exploit replanning speed. The occupancy model is rebuilt from scratch each frame; an incremental update would further reduce A*'s overhead. Additionally, the high failure rate suggests the environment may be too dense for reliable navigation—adaptive particle density or a fallback "wait-and-replan" behavior could improve success.

VIII. Conclusion and Future Work

We presented a 3D replanning simulator in which a robot particle navigates a live SPH fluid environment toward a goal, continuously replanning with either A* or RRT*. Across 1,142 benchmark trials spanning three scenarios and 18 particle counts (500 to 25,000 in steps of 1,500, plus an explicit endpoint at 25,000), RRT* outperformed A* on every primary metric: 2–3× faster time to goal, 10–24× faster per-replan latency, and substantially lower variance in navigation time. Both algorithms produced comparable path quality and divergence.

These results highlight a fundamental trade-off in dynamic path planning. In dense, rapidly changing environments, a planner with bounded computation time (RRT*) is more effective than one with optimality guarantee (A*). The ability to reliably return *a* feasible plan within a fixed time budget overweights the benefit of computing the *optimal* plan at an unpredictable cost.

Directions for future work include: (1) **Incremental/anytime replanning**—D* Lite or m-LPA* [4] would allow A* to amortize work across replanning calls, potentially closing the latency gap with RRT* by exploiting the multiscale quadtree structure to bound worst-case vertex expansions; (2) **Trajectory-level smoothing**—a spline-based or model-predictive trajectory layer wcontrol could improve motion stability and reduce sensitivity to frequent replanning updates; (3) **Adaptive occupancy**—a density-field threshold rather than raw particle positions could yield a more stable occupancy map, reducing the failed replan rate; (4) **Learned heuristics**—a neural-network heuristic for A*, trained on past planning queries in this domain, could speed up A* while preserving its optimality guarantee; and (5) **GPU-accelerated planning**—the CUDA backend already accelerates SPH; extending it to parallelize RRT* sampling or A* node expansion could enable real-time planning at much higher particle counts.

References

[1] J. J. Monaghan, "Smoothed particle hydrodynamics," *Annual Review of Astronomy and Astrophysics*, vol. 30, pp. 543–574, 1992.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[3] S. Koenig and M. Likhachev, "D* lite," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2002, pp. 476–483.

[4] Y. Lu, X. Huo, O. Arslan, and P. Tsiotras, "Incremental multi-scale search algorithm for dynamic path planning with low worst-case complexity," *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 41, no. 6, pp. 1556–1570, 2011.

[5] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Department of Computer Science, Iowa State University, Tech. Rep. TR 98-11, 1998.

[6] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[7] D. Ferguson and A. Stentz, "Anytime, dynamic planning in high-dimensional search spaces," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2007.