3D Vicsek Swarm Model: Collective Behavior in Three-Dimensional Space

Andrea Simone Costa

October 11, 2025

1 Introduction

This project extends the 2D Vicsek flocking model to three-dimensional space, providing a more realistic representation of collective motion in bird flocks, fish schools, and aerial drone swarms. While the fundamental principles remain the same - local alignment with neighbors plus noise leads to global coherent motion - the 3D implementation introduces additional complexity in directional sampling, visualization, and behavioral rules.

The simulation implements four behavioral forces: alignment (matching neighbor directions), separation (maintaining personal space), cohesion (attraction to group centers), and boundary avoidance (repulsion from domain edges). These rules, inspired by the classic Boids model by Craig Reynolds (1987) and the Vicsek model (1995), produce emergent flocking patterns that resemble natural swarm behavior.

2 Mathematical Framework

2.1 3D Direction Representation

Unlike 2D where directions are simply angles $\theta \in [0, 2\pi)$, 3D directions are unit vectors:

$$\hat{\mathbf{d}} = (d_x, d_y, d_z), \quad |\hat{\mathbf{d}}| = \sqrt{d_x^2 + d_y^2 + d_z^2} = 1 \tag{1}$$

These can be parameterized using spherical coordinates:

$$\hat{\mathbf{d}} = \begin{pmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{pmatrix} \tag{2}$$

where $\theta \in [0, \pi]$ is the polar angle (angle from the positive z-axis) and $\phi \in [0, 2\pi)$ is the azimuthal angle (angle in the xy-plane).

2.2 Uniform Random Directions

To generate uniformly distributed random directions on the unit sphere, we cannot simply sample θ and ϕ uniformly, as this would oversample near the poles. The correct approach uses the inverse cumulative distribution function:

$$\theta = \arccos(1 - 2u), \quad u \sim \mathcal{U}[0, 1] \tag{3}$$

$$\phi = 2\pi v, \quad v \sim \mathcal{U}[0, 1] \tag{4}$$

This ensures uniform coverage of the sphere's surface.

2.3 Alignment Rule

Each agent i calculates the average direction of all neighbors (including itself) within the interaction radius r:

$$\hat{\mathbf{d}}_{\text{align},i} = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \hat{\mathbf{d}}_j \tag{5}$$

where $\mathcal{N}_i = \{j : |\mathbf{r}_j - \mathbf{r}_i| < r\}$ is the set of neighbors. The result is then normalized:

$$\hat{\mathbf{d}}_{\text{align},i} \to \frac{\hat{\mathbf{d}}_{\text{align},i}}{|\hat{\mathbf{d}}_{\text{align},i}|} \tag{6}$$

2.4 Separation Force

To prevent overlap and clustering, a repulsive force is applied when agents are closer than a separation threshold d_{sep}:

$$\mathbf{F}_{\text{sep},i} = s_{\text{sep}} \sum_{j \in \mathcal{N}_i, d_{ij} < d_{\text{sep}}} \frac{d_{\text{sep}} - d_{ij}}{d_{\text{sep}}} \frac{\mathbf{r}_i - \mathbf{r}_j}{d_{ij}} \tag{7}$$

where $d_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$ and s_{sep} is the separation strength parameter.

2.5 Cohesion Force

Cohesion creates attraction toward the center of mass of nearby agents, promoting group cohesion:

$$\mathbf{C}_i = \frac{1}{|\mathcal{N}_i^{\text{coh}}|} \sum_{j \in \mathcal{N}_i^{\text{coh}}} \mathbf{r}_j \tag{8}$$

where $\mathcal{N}_i^{\text{coh}}$ is the set of agents within cohesion radius r_{coh}. The cohesion force is then:

$$\mathbf{F}_{\text{coh},i} = s_{\text{coh}}(\mathbf{C}_i - \mathbf{r}_i) \tag{9}$$

In this implementation, $r_{\text{coh}} = 2.5 \times r$, meaning agents are attracted to a larger group than they align with - a biologically realistic feature.

2.6 Order Parameter in 3D

The order parameter measures the degree of alignment:

$$\Phi = \frac{1}{N} \left| \sum_{i=1}^{N} \hat{\mathbf{d}}_i \right| = \frac{1}{N} \sqrt{\left(\sum_i d_{i,x}\right)^2 + \left(\sum_i d_{i,y}\right)^2 + \left(\sum_i d_{i,z}\right)^2} \tag{10}$$

As in 2D, $\Phi = 0$ indicates complete disorder and $\Phi = 1$ indicates perfect alignment.

3 Implementation Details

3.1 Velocity Heterogeneity

Unlike the standard Vicsek model where all agents move at the same speed, this implementation introduces realistic velocity variation. Each agent is assigned an individual speed:

$$v_i = v_0(1 + \epsilon_i), \quad \epsilon_i \sim \mathcal{U}[-0.15, 0.15] \tag{11}$$

where v_0 is the base speed. This $\pm 15\%$ variation creates more natural-looking dynamics, as agents occasionally overtake each other or lag behind.

3.2 Stochastic Noise Application

Rather than adding noise to every agent at every time step, the simulation uses stochastic noise application. Each agent has probability $p = 1/N_{\text{interval}}$ of receiving a random directional perturbation at each step. This creates smoother trajectories with occasional stochastic kicks, resembling biological decision-making better than constant noise.

The noise vector is a uniformly sampled 3D unit vector (using the method described in Section 2.2), scaled by the noise level η:

$$\hat{\mathbf{d}}_{\text{noise}} = \eta \, \hat{\mathbf{n}}_{\text{random}} \tag{12}$$

3.3 Synchronous Updates

To ensure fairness and avoid artifacts from update ordering, all agent directions are computed based on the current state, stored temporarily, and then applied simultaneously:

1. For each agent, calculate new direction based on current positions and directions
2. Store new direction in temporary array
3. Apply all new directions simultaneously
4. Update positions based on new directions

3.4 Three.js Visualization

Agents are rendered as 3D cone meshes (resembling arrows) pointing in their direction of motion. The cone orientation is achieved using quaternion rotation:

```
const up = new THREE.Vector3(0, 1, 0);
const quaternion = new THREE.Quaternion();
quaternion.setFromUnitVectors(up, directionVector);
mesh.setRotationFromQuaternion(quaternion);
```

The color of each agent is determined by mapping the direction vector components to RGB channels:

$$\text{Red} = \frac{d_x + 1}{2}, \quad \text{Green} = \frac{d_y + 1}{2}, \quad \text{Blue} = \frac{d_z + 1}{2} \tag{13}$$

This creates visually distinctive colors for different directions, making it easy to identify aligned subgroups.

4 Behavioral Patterns

4.1 Toroidal Vortices

One striking pattern that can emerge is a toroidal vortex: agents circulate in a doughnut-shaped pattern. This occurs when:

- Cohesion is strong enough to keep the group together
- Boundary avoidance creates inward-turning forces

- The combination produces a self-sustaining circular flow

Toroidal structures are observed in some biological systems, such as fish schools avoiding predators.

4.2 Fragmentation and Merging

At intermediate noise levels ($\eta \approx 0.3 - 0.5$), the swarm exhibits dynamic fragmentation and merging:

- The group spontaneously splits into multiple subgroups
- Subgroups move in different directions
- Occasionally, subgroups collide and merge back together
- The cycle repeats, creating complex spatiotemporal patterns

This behavior resembles fission-fusion dynamics observed in many social animals.

4.3 Effect of 3D vs 2D

Compared to 2D, the 3D model exhibits:

- **Reduced collision frequency**: Agents have more space to maneuver around each other
- **Lower critical density**: The minimum density needed for coherent flocking is lower in 3D
- **More complex trajectories**: The additional dimension allows for spiraling, looping, and weaving patterns impossible in 2D
- **Increased computational cost**: Neighbor finding in 3D requires checking a larger volume

5 Comparison with Biological Systems

5.1 Bird Flocks

Starling murmurations are perhaps the most spectacular example of 3D collective behavior. The simulation captures key features:

- Rapid alignment propagation (information spreads through the flock via local interactions)
- Density-dependent coordination (tighter groups show higher order)
- Dynamic shape changes (the flock morphs and flows without a fixed structure)

However, real starlings use topological (nearest-neighbor) rather than metric (fixed-radius) interactions, which makes the system more robust to density variations.

5.2 Fish Schools

Pelagic fish (like herring or tuna) school in 3D, often forming compressed ellipsoids or dynamic formations. The separation and cohesion forces in the model capture:

- Preferred spacing between individuals
- Attraction to the group center
- Coordinated turning maneuvers

The model does not include hydrodynamic interactions (water flow effects), which play a significant role in real fish schools.

5.3 Insect Swarms

Some insects (like midges or locusts) form swarms without fixed spacing. The cohesion-dominated regime of the model ($s_{\text{coh}} > s_{\text{sep}}$) produces cloud-like swarms that resemble insect aggregations.

6 Software Architecture

6.1 System Overview

The 3D Vicsek implementation extends 2D with Three.js for 3D rendering and spherical coordinate sampling:

```
index.ts (Entry Point)
    |
    v
VicsekModel3D (AgentScript Model3D)
    |-- turtles.create(300) -> 3D agents
    |-- step() -> Directions → Positions → Order parameter
    |
    +-- updateDirections() -> Alignment + separation + cohesion
    +-- generate3DNoise() -> Uniform sphere sampling
    |
    v
SwarmVisualization3D (Three.js)
    |-- Scene + PerspectiveCamera + WebGLRenderer
    |-- OrbitControls for interaction
    |-- Cone meshes with direction-based RGB coloring
```

6.2 Core Modules

VicsekModel3D (`vicsekModel3D.ts`): Extends `Model3D` for 3D space. Each agent has direction vector (d_x, d_y, d_z) with $|\mathbf{d}| = 1$. Implements four forces: alignment (mean direction), separation (repulsion), cohesion (attraction to center of mass), boundary avoidance. Individual velocity variation: $v_i = v_0(1 + \epsilon)$ where $\epsilon \sim \mathcal{U}[-0.15, 0.15]$.

SwarmVisualization3D (`swarmVisualization3D.ts`): Creates `THREE.Scene` with cone meshes for agents. Uses `OrbitControls` for camera. Cone orientation via quaternion: `setFromUnitVectors(up` `directionVector)`. Color mapping: RGB $= ((d_x + 1)/2, (d_y + 1)/2, (d_z + 1)/2)$.

6.3 Execution Flow

Each frame:

1. **Direction Update**:
   - Find neighbors within `interactionRadius`
   - Mean direction: $\bar{\mathbf{d}} = (1/N_{\text{neighbors}}) \sum_j \mathbf{d}_j$
   - Separation: $\mathbf{F}_{\text{sep}} = \sum_{d_{ij} < d_{\text{sep}}} s(\mathbf{r}_i - \mathbf{r}_j)/d_{ij}$
   - Cohesion (radius $2.5 \times r$): $\mathbf{F}_{\text{coh}} = s_{\text{coh}}(\mathbf{C} - \mathbf{r}_i)$ where $\mathbf{C}$ = center of mass
   - Boundary avoidance: repel from walls within distance 3.0
   - Stochastic noise (prob 1/20 per tick): add random 3D unit vector scaled by η
   - Normalize: $\mathbf{d} \to \mathbf{d}/|\mathbf{d}|$
2. **Position Update**: $\mathbf{r} \to \mathbf{r} + v_i \mathbf{d}_i$, clamp to boundaries
3. **Order Parameter**: $\Phi = (1/N)|\sum_i \mathbf{d}_i|$
4. **Three.js Rendering**:
   - Update cone positions: `mesh.position.set(x, y, z)`
   - Update orientations: quaternion from direction vector
   - Update colors: RGB from direction components
   - `controls.update()`, `renderer.render(scene, camera)`

6.4 Uniform 3D Sampling

Generate random direction uniformly on sphere (inverse CDF method):

```
theta = acos(1 - 2*u)  // u ~ U(0,1), polar angle
phi = 2*PI*v           // v ~ U(0,1), azimuthal angle
direction = {
  x: sin(theta)*cos(phi),
  y: sin(theta)*sin(phi),
  z: cos(theta)
}
```

Naive uniform sampling of θ and ϕ would oversample poles.

6.5 Three.js Integration

Setup:

```
// Scene and camera
scene = new THREE.Scene()
camera = new THREE.PerspectiveCamera(75, aspect, 0.1, 1000)
camera.position.set(worldSize, worldSize, worldSize)

// Agent cones
coneGeometry = new THREE.ConeGeometry(0.15, 0.5, 8)
for each agent:
  material = new THREE.MeshBasicMaterial({ color })
  mesh = new THREE.Mesh(coneGeometry, material)
  scene.add(mesh)

// OrbitControls
controls = new OrbitControls(camera, renderer.domElement)
controls.enableDamping = true
```

Quaternion orientation:

```
up = new THREE.Vector3(0, 1, 0)
quaternion = new THREE.Quaternion()
quaternion.setFromUnitVectors(up, directionVector)
mesh.setRotationFromQuaternion(quaternion)
```

This rotates the cone (default pointing up) to align with agent direction.

6.6 Cohesion Implementation

Cohesion radius $r_{\text{coh}} = 2.5 \times r_{\text{align}}$ ensures agents attracted to larger group than alignment range. Center of mass:

```
C = (1/N_coh) * sum(r_j for j in cohesion_neighbors)
F_coh = s_coh * (C - r_i)
```

Force strength $s_{\text{coh}} = 0.2$ provides gentle attraction without overwhelming alignment.

7 Conclusion

The 3D Vicsek model successfully captures essential features of collective motion in three-dimensional space. By extending the classic 2D model with cohesion forces, velocity heterogeneity, and realistic noise application, the simulation produces diverse behavioral patterns including toroidal vortices, fragmentation-merging cycles, and coherent flocking. The Three.js visualization with interactive camera controls allows intuitive exploration of 3D swarm dynamics. This project demonstrates how simple local rules - alignment, separation, cohesion - can generate complex emergent behavior resembling natural biological systems, providing insights into self-organization, collective decision-making, and active matter physics in three dimensions.