Simulación basada en agentes y estructuras espaciales

Eduardo Graells-Garrido

30 de marzo de 2026

1 Simulación basada en agentes

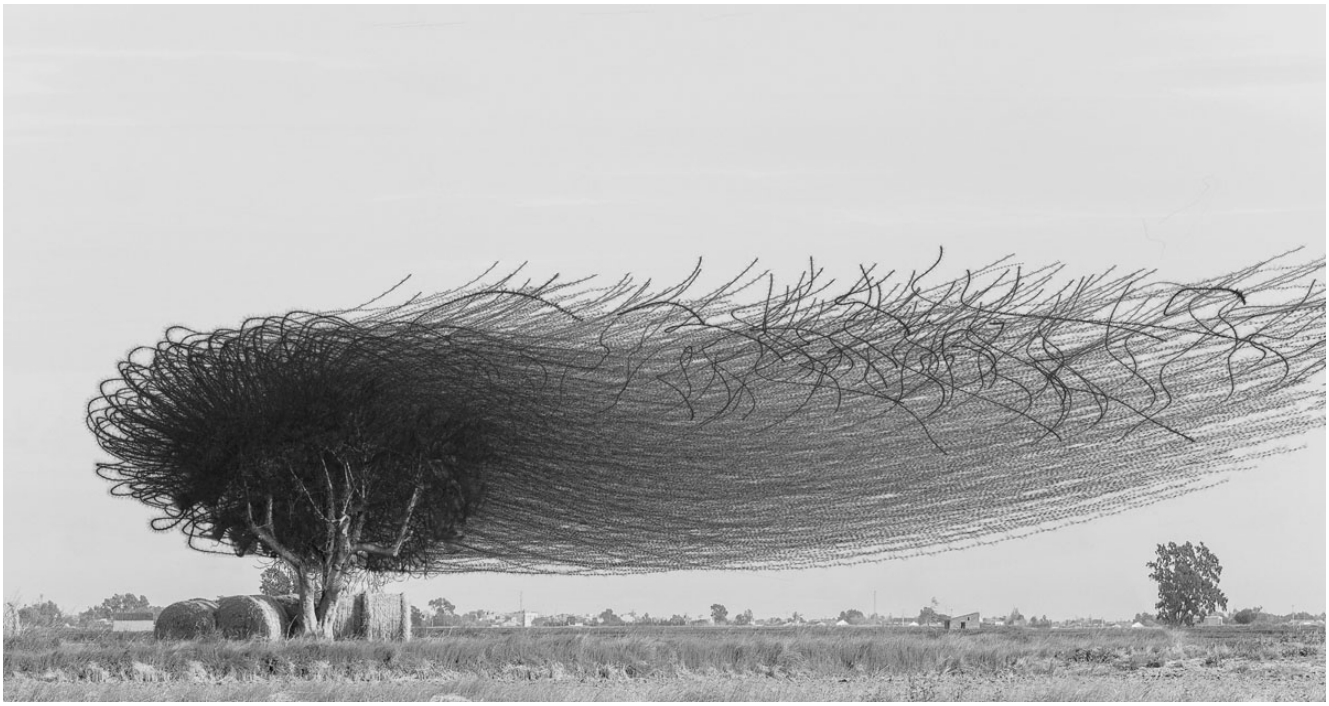


Figura 1. *Ornitografía #152*. Fuente: Xavi Bou https://xavibou.com/ornithographies/.

El fotógrafo Xavi Bou capturó el movimiento de aves a lo largo del tiempo en su serie de *ornitografías* (Fig. 1). Esta palabra combina dos raíces griegas: *ornitho-*, que significa ave o pájaro, y *-grafía*, que refiere a la escritura o registro visual. Se traduce como "escritura o registro de aves". Las ornitografías de Bou revelan patrones que no podemos apreciar a simple vista: el vuelo de las aves, que percibimos como una secuencia de momentos discretos, se transforma en curvas fluidas que muestran la continuidad del movimiento. Esta visualización nos conecta con el concepto central que exploraremos hoy: cómo comportamientos individuales simples pueden generar patrones complejos (y hermosos) cuando se observan en conjunto a lo largo del tiempo.

En las unidades anteriores estudiamos sistemas dinámicos que evolucionan según ecuaciones deterministas, desde atractores extraños hasta sistemas de partículas. Ahora exploraremos sistemas donde las entidades no solo siguen leyes físicas, sino que también toman decisiones autónomas e interactúan entre sí, lo que genera comportamientos colectivos emergentes.

La simulación basada en agentes[1] (*Agent-Based Modeling* o ABM) es un paradigma de modelamiento computacional que se distingue por su enfoque "de abajo hacia arriba" (*bottom-up*). A dife-

[1] No confundir con los "agentes de IA" populares hoy.

rencia de otros enfoques de simulación que describen el comportamiento global de un sistema mediante ecuaciones diferenciales u otros formalismos matemáticos (“arriba hacia abajo” o *top-down*), ABM modela el sistema a nivel individual, donde cada entidad (agente) actúa de manera autónoma siguiendo reglas sencillas. Se utiliza en el campo de **vida artificial**, que reúne leyes de comportamiento y métodos computacionales [1].

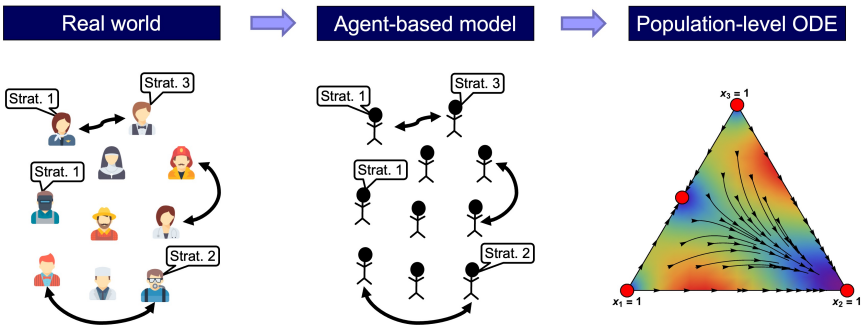


Figura 2. ABM simula a nivel individual, basándonos en reglas y entidades de la realidad, mientras que modelos basados en EDO tienden a describir el comportamiento general. Fuente: *Agent-Based Evolutionary Game Dynamics*.

Un modelo basado en agentes consta de tres elementos: los **agentes**, el **entorno** en que estos interactúan y las **reglas** que determinan tanto el comportamiento individual como las interacciones (Fig. 2, de [3]).

Un agente se define como una entidad computacional con cuatro características esenciales: *autonomía*, *heterogeneidad*, *actividad limitada* y *percepción restringida*:

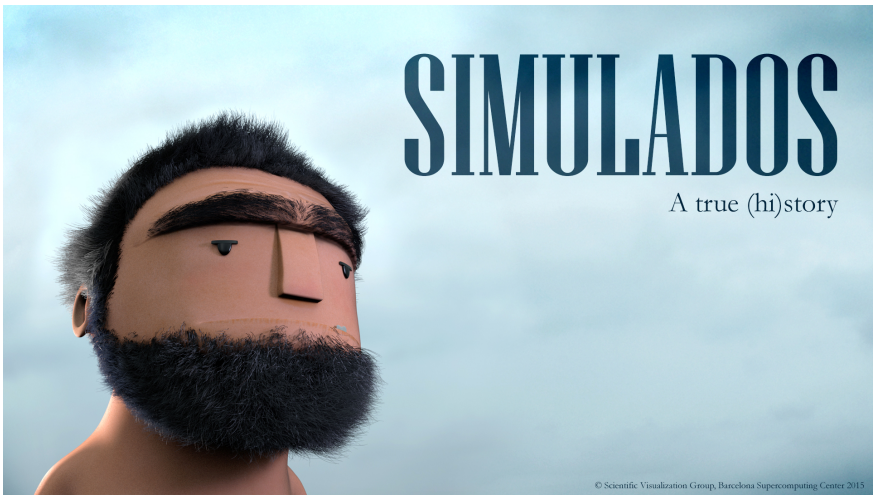
- **Autonomía**: cada agente toma decisiones independientes basadas en su estado interno y su percepción del entorno.
- **Heterogeneidad**: diferentes agentes poseen distintos atributos o siguan variantes de las reglas base.
- **Actividad**: proactividad (iniciativa propia) o reactividad (respuesta a estímulos externos).
- **Percepción limitada**: restringe el conocimiento del agente a su entorno inmediato.

Los fenómenos emergentes constituyen el aspecto más intrigante de los sistemas basados en agentes. “Emergencia” se refiere a aquellos comportamientos colectivos que no pueden predecirse a partir de las reglas individuales y que surgen de forma espontánea cuando múltiples agentes interactúan. Ejemplos cotidianos incluyen la formación espontánea de carriles en multitudes que caminan en direcciones opuestas, o los patrones en espiral que forman los cardúmenes de peces para protegerse de depredadores.

Para ilustrar la ubicuidad histórica de los modelos basados en agentes, consideremos un ejemplo de la cultura *pop*: el videojuego *Pac-Man* (Fig. 3). Los fantasmas que persiguen al jugador

son agentes. Cada fantasma sigue reglas simples e individualizadas: Blinky (rojo) persigue al protagonista, Pinky (rosa) intenta emboscarlo, Inky (azul) combina estrategias y Clyde (naranja) alterna entre persecución y comportamiento aleatorio[2]. Estos fantasmas generan situaciones complejas que desafían a quien está controlando a Pac-Man.

La simulación basada en agentes complementa los sistemas dinámicos que estudiamos en unidades anteriores. Las EDO modelan comportamiento generalizado ("top-down"), mientras que ABM explora la heterogeneidad individual y los mecanismos que generan esos comportamientos al ser agregados, así como las diferencias que causan diferentes configuraciones de un mundo. El cortometraje *Simulados* (Fig. 4) muestra este mecanismo mediante un proyecto que mide la evolución de las sociedades humanas del pasado: no tenemos registros de lo que sucedió en ese entonces, pero sí hay vestigios de lo que hicieron las personas, y mediante simulación podemos aproximar una reconstrucción de la historia..



Entre las ventajas de ABM destacan su capacidad para modelar comportamientos intuitivos sin necesidad de formalizar ecuaciones complejas, su flexibilidad para incorporar heterogeneidad y adaptabilidad, y su potencial para capturar fenómenos emergentes que serían difíciles de predecir con otros enfoques. ¿Cómo evacuar un estadio ante una emergencia (Fig. 5)? ¿Cómo evacuar una ciudad (Fig. 6)? No existen datos suficientes para predecir esos fenómenos considerando la complejidad que conllevan, pero sí para simularlos.

[2] Puedes ver una explicación aquí.

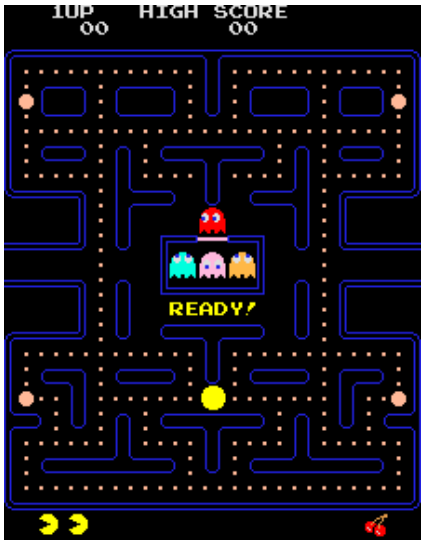
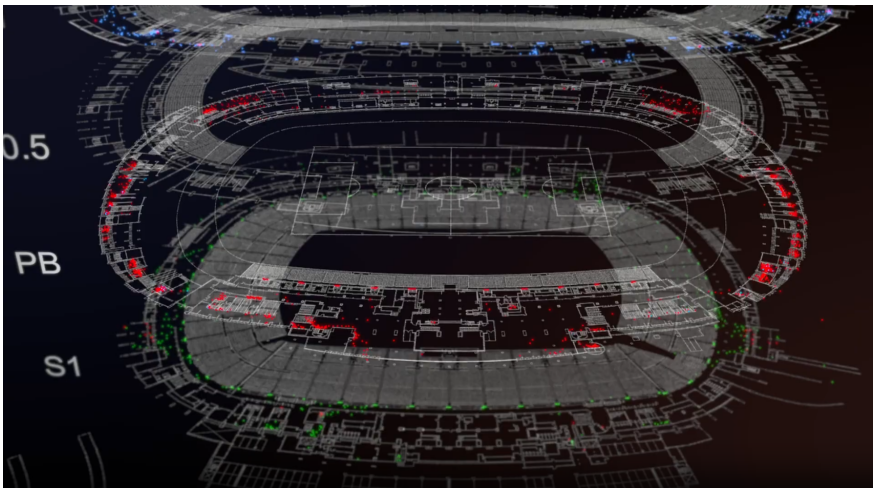


Figura 3. *Pac-man*. ¿Sabías que cada fantasma tiene *personalidad*? Fuente: Wikipedia.

Figura 4. *Simulados* es un cortometraje que comunica el funcionamiento y los resultados de modelar el comportamiento humano siglos atrás. Fuente: BSC Visualization Group.

Figura 5. Simulación basada en agentes para evacuación del Camp Nou, el estadio del FC Barcelona. Fuente: BSC Visualization Group (nota: el profe participó en este proyecto :D).



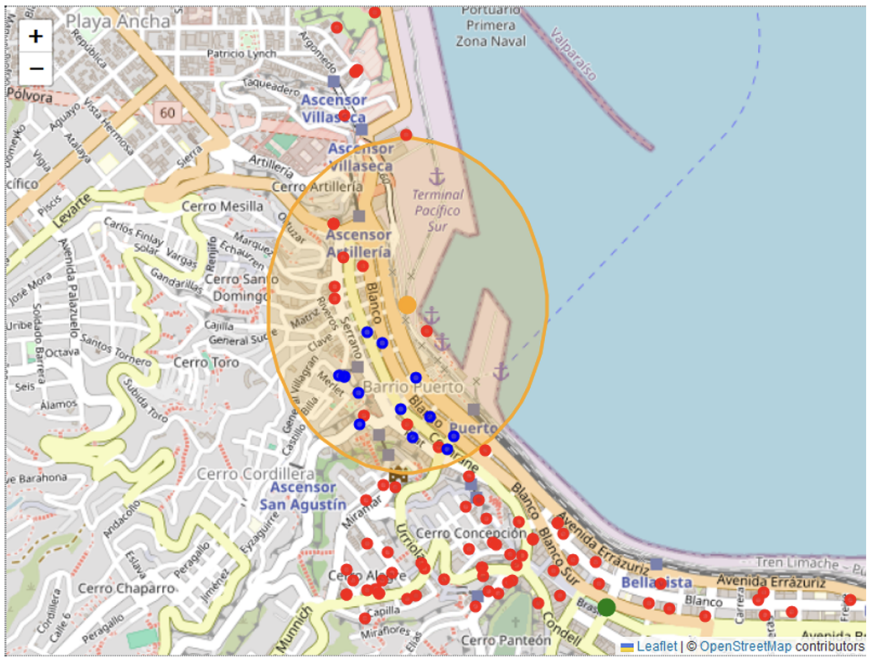


Figura 6. Simulación para evacuación urbana con el propósito de mejorar los planes de evacuación ante incendios forestales en Valparaíso. Este fue un proyecto de título de Paula Cabrera, titulada DCC en marzo de 2025 y profesora auxiliar del curso en su versión de otoño de ese mismo año.

Las aplicaciones de ABM abarcan múltiples disciplinas. En ciencias sociales, facilitan el análisis de dinámicas de multitudes, patrones de evacuación y propagación de información. En la industria del entretenimiento, estos modelos generan comportamientos realistas en personajes no jugadores (NPC, *Non Playable Character*) y multitudes virtuales en películas y videojuegos. En ingeniería, contribuyen al diseño de sistemas robóticos colaborativos, desde enjambres de drones hasta vehículos autónomos. En biología, permiten estudiar comportamientos animales desde cardúmenes y bandadas hasta ecosistemas completos. Un ejemplo concreto es la tesis doctoral de Sarah Ward [5], que comprendió los factores que explicaban el decline pobla-

cional de flamencos en un lago importante para esta especie en África (Fig. 7).

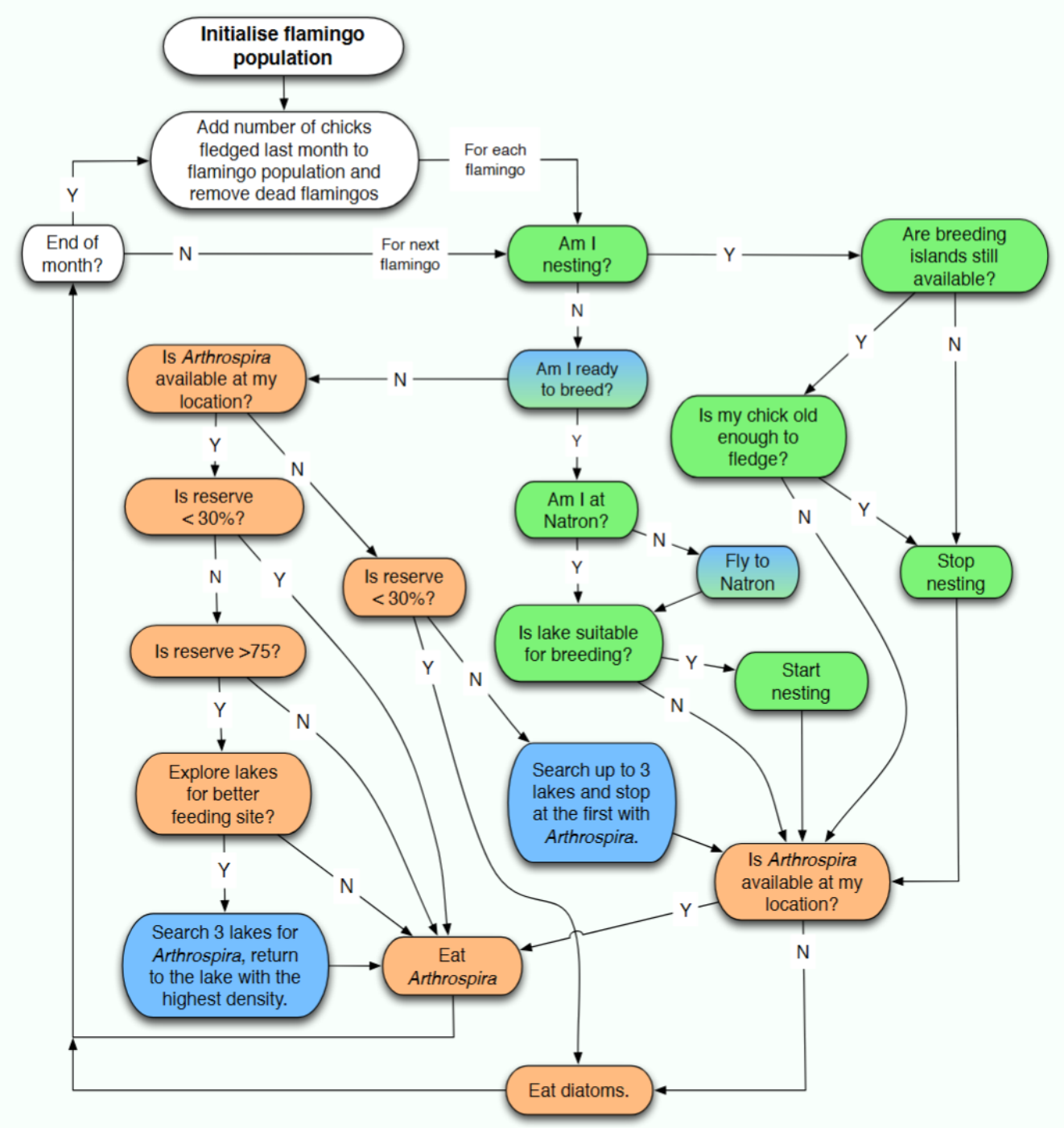


Figura 7. Modelo de simulación de flamencos en los lagos de África. Las cajas verdes se relacionan con el comportamiento reproductivo; las azules, con el desplazamiento entre un lago y otro; y las verdes, con la alimentación. Fuente: Sarah Ward.

BOIDS: Un modelo para comportamiento colectivo

La Fig. 8 muestra una bandada de estorninos en una danza aérea conocida como "murmuración". ¿Cómo logran miles de aves volar en formaciones tan complejas sin colisionar? ¿Existe un líder que coordina estos movimientos?

Figura 8. Una bandada de estorninos. ¿Por qué no chocan entre sí? Fuente: WikiCommons.

Una de las implementaciones más conocidas de simulación basada en agentes es el modelo BOIDS [4], desarrollado por Craig Reynolds. Su nombre deriva de una contracción de *bird-oid objects* (objetos similares a aves), lo que refleja su inspiración original: simular el comportamiento de bandadas de pájaros. Según propuso Reynolds, en una bandada no hay líder ni plan central, sino que cada ave sigue reglas simples que, en conjunto, generan este comportamiento colectivo **emergente**.

En cine, BOIDS es base de la animación de grupos: las escenas de estampidas en *El Rey León*, los cardúmenes en *Buscando a Nemo* y las batallas épicas en *El Señor de los Anillos* utilizan variantes del algoritmo BOIDS. En videojuegos, controla el comportamiento de enemigos y personajes no-controlables (NPC) que actúan en grupo de manera automatizada (es posible que los *minions* de *League of Legends* utilicen un algoritmo similar). En simulaciones científicas, ayuda a entender desde el comportamiento animal hasta dinámicas de evacuación en emergencias.

Un agente (un *boid*) sigue tres reglas básicas, cada una asociada a una fuerza que influye en su movimiento: *separación*, *alineación* y *cohesión* (Fig. 9). La primera regla, **separación**, actúa como un mecanismo para evitar colisiones. Cada boid mantiene una distancia mínima respecto a sus vecinos cercanos, lo que genera una fuerza repulsiva que aumenta con la cercanía. La regla de separación para un boid i se expresa como:

$$\vec{v}_{sep} = \sum_{j \in N_i} \frac{\vec{x}_i - \vec{x}_j}{|\vec{x}_i - \vec{x}_j|^2},$$

donde N_i representa el conjunto de vecinos dentro del radio de percepción del boid i, y $\vec{x}_i$ y $\vec{x}_j$ son las posiciones del boid i y su vecino j. La división por el cuadrado de la distancia enfatiza la importancia de evitar colisiones inminentes: cuanto más cerca está un vecino, más fuerte es la repulsión.

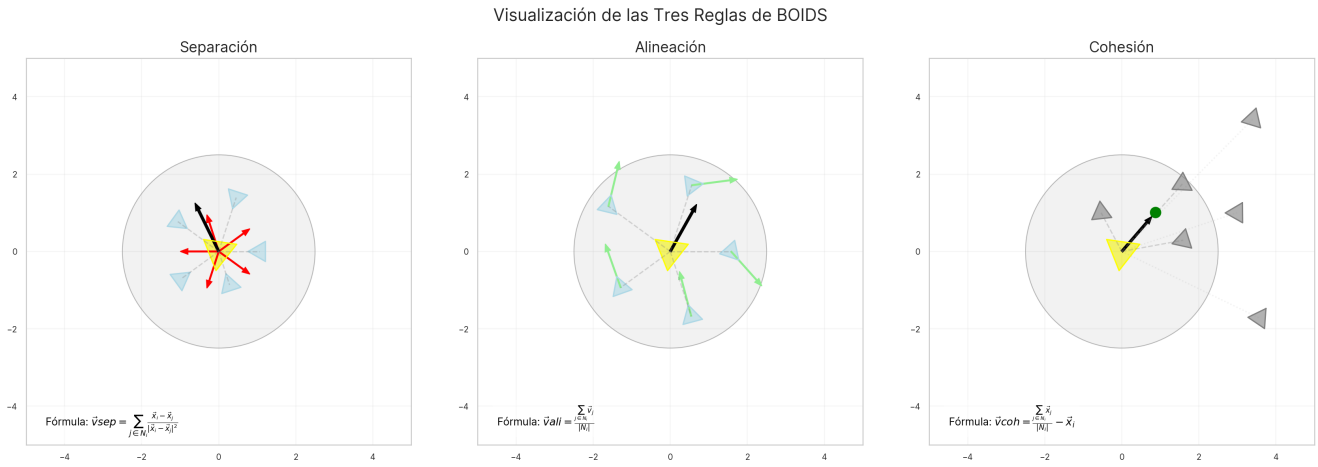


Figura 9. Diagrama de reglas de comportamiento en una simulación BOIDS.

La segunda regla, **alineación**, induce a cada boid a orientarse en la dirección promedio de sus vecinos. Esta tendencia a "ir con la corriente" crea flujo coherente dentro del grupo y aumenta la eficiencia del movimiento colectivo. La regla de alineación se calcula como:

$$\vec{v}_{ali} = \frac{\sum_{j \in N_i} \vec{v}_j}{|N_i|},$$

donde $\vec{v}_j$ representa el vector velocidad del vecino j. Esta fórmula calcula el promedio de las velocidades de todos los vecinos, lo que proporciona una dirección "consensuada" hacia la cual el boid debería orientarse.

La tercera regla, **cohesión**, atrae cada boid hacia la posición promedio de sus vecinos, lo que mantiene la integridad del grupo. Sin esta fuerza, los boids tenderían a dispersarse debido a pequeñas perturbaciones o evasiones. La regla de cohesión se define como:

$$\vec{v}_{coh} = \frac{\sum_{j \in N_i} \vec{x}_j}{|N_i|} - \vec{x}_i.$$

La fórmula calcula el centroide (centro de masa) del grupo local y genera una fuerza proporcional a la distancia entre el boid y este centro.

Las reglas de comportamiento de BOIDS han seguido siendo objeto de investigación (Fig. 10), puesto que el modelo base no logra recrear del todo los comportamientos de las aves [2].

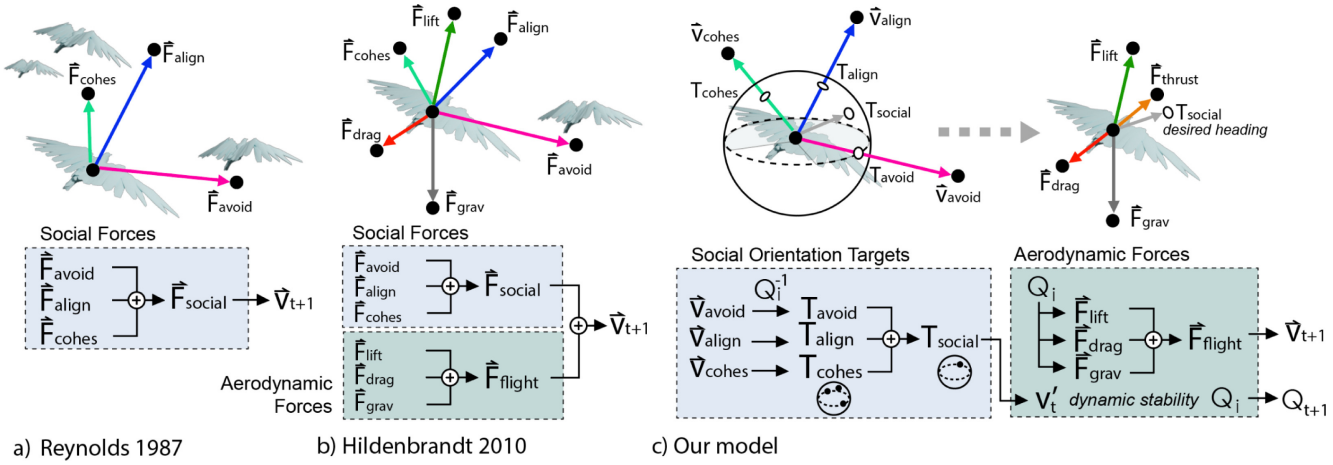


Figura 10. El modelo en (c) puede replicar las murmuraciones de los estorninos. Fuente: R. C. Hoetzlein.

Para actualizar el estado de cada boid en la simulación, combinamos las fuerzas con pesos específicos que determinan la importancia relativa de cada regla, e integramos (en este caso, utilizando el método de Euler):

$$\vec{v}_i(t + \Delta t) = \vec{v}_i(t) + \Delta t \cdot (w_{sep}\vec{v}_{sep} + w_{ali}\vec{v}_{ali} + w_{coh}\vec{v}_{coh}).$$

Los coeficientes w_{sep}, w_{ali} y w_{coh} controlan la influencia de cada regla. Sus valores determinarán si la bandada se mueve de forma cohesionada, se dispersa con facilidad o tiende a formar subgrupos.

Restricciones y parámetros

En una simulación debemos implementar restricciones que reflejen las limitaciones físicas del sistema modelado. En el caso de BOIDS, estas restricciones permiten mantener un comportamiento realista:

- **Velocidad máxima**, que limita qué tan rápido puede moverse un boid: $|\vec{v}_i| \leq v_{max}$. Esta restricción refleja las limitaciones energéticas y físicas de los organismos reales.
- **Aceleración máxima**: $|\vec{a}_i| \leq a_{max}$, que modela las restricciones biomecánicas que impiden cambios instantáneos de velocidad.
- **Radio de percepción**, que determina la distancia máxima a la que un boid puede detectar a sus vecinos, y modela las limitaciones sensoriales de los organismos reales.
- **Ángulo de visión**, que restringe la percepción a un cono frontal, calculado mediante el producto escalar: $\cos(\theta) = \frac{\vec{v}_i \cdot (\vec{x}_j - \vec{x}_i)}{|\vec{v}_i||\vec{x}_j - \vec{x}_i|} > \cos(\theta_{max})$. Esta fórmula determina si un vecino j se encuentra dentro del campo visual del boid i, al comparar el ángulo formado entre la dirección de movimiento

y la dirección hacia el vecino con un ángulo máximo predefinido.

Otros parámetros incluyen las limitaciones de cambio máximo de los parámetros de cada boid en un paso temporal. Estos valores deben calibrarse para obtener comportamientos realistas según el tipo de organismo simulado: los peces pueden girar con mayor brusquedad que las aves, y los insectos pueden acelerar con mayor rapidez que los mamíferos grandes.

Métricas para analizar comportamiento emergente

La simulación es una herramienta para comprender fenómenos complejos: definimos las reglas de comportamiento de los agentes, pero medimos métricas del sistema general. Por ejemplo, para analizar el comportamiento emergente, podemos utilizar diversas métricas cuantitativas.

La **polarización** mide el grado de alineamiento global del grupo, calculada como:

$$P = \frac{1}{N} \left| \sum_{i=1}^{N} \frac{\vec{v}_i}{|\vec{v}_i|} \right|.$$

Este valor oscila entre 0 (orientaciones aleatorias) y 1 (todos los boids alineados). La polarización alta indica formaciones direccionales como las observadas en migraciones (Fig. **??**).

La **dispersión** evalúa cuán extendido o compacto se encuentra el grupo, medida como la desviación estándar de las distancias entre agentes. Una dispersión baja indica grupos compactos, mientras que valores altos sugieren distribuciones espaciales más difusas.

El **número de *clusters*** permite identificar si la población se ha dividido en subgrupos distinguibles. Mediante análisis de conectividad en el grafo de vecindad, podemos determinar cuántos componentes conexos existen en un momento dado, lo que revela patrones de fragmentación y reagrupamiento.

La **estabilidad** examina la variación temporal de las métricas anteriores. Un sistema estable mostrará fluctuaciones pequeñas en sus métricas, mientras que cambios significativos pueden indicar transiciones de fase[3] en el comportamiento colectivo.

Estas métricas proporcionan herramientas analíticas que permiten calibrar modelos para obtener comportamientos específicos.

[3] Recuerden que el espacio de fase es el conjunto de estados posibles del sistema. Una transición de fase es, entonces, el paso de un conjunto de estados posibles a otro.

Implementación: BOIDS con partículas

En el repositorio del curso, el ejemplo `boids_particles` implementa BOIDS en 2D con visualización en tiempo real. Cada boid se representa como un triángulo orientado en la dirección de su movimiento. Se ejecuta con:

```
uv run python caja_de_juguetes.py boids_particles
```

El modelo está construido con la biblioteca Mesa, diseñada para simulación basada en agentes en Python. Mesa provee dos abstracciones que utilizamos: `mesa.Agent` (el agente individual) y `mesa.Model` (el mundo que contiene a los agentes).

Cada boid implementa las tres reglas como métodos separados. La regla de cohesión calcula el centroide de los vecinos y genera un vector que apunta hacia él:

```
def cohere(self, neighbors):
    cohere = np.zeros(2)
    if neighbors:
        for neighbor in neighbors:
            cohere += neighbor.pos
        cohere /= len(neighbors)
    return self.model.space.get_heading(self.pos, cohere)
```

La separación genera una fuerza que aleja al boid de vecinos demasiado cercanos:

```
def separate(self, neighbors):
    separation_vector = np.zeros(2)
    for other in (n.pos for n in neighbors):
        if self.model.space.get_distance(self.pos, other) < self.distance:
            separation_vector -= self.model.space.get_heading(self.pos, other)
    return separation_vector
```

La alineación promedia las velocidades de los vecinos:

```
def match_heading(self, neighbors):
    match_vector = np.zeros(2)
    if neighbors:
        for neighbor in neighbors:
            match_vector += neighbor.velocity
        match_vector /= len(neighbors)
    return match_vector
```

En cada paso de simulación, el boid combina las tres fuerzas con sus pesos respectivos y actualiza su velocidad. La velocidad se normaliza y se limita a una velocidad máxima (`self.speed`):

```
def step(self):
    neighbors = self.model.query_area(self.pos, self.vision)
    neighbors = list(filter(lambda x: x ≠ self, neighbors))

    self.velocity += (
        self.cohere(neighbors) * self.cohere_factor
        + self.separate(neighbors) * self.separation_factor
        + self.match_heading(neighbors) * self.match_factor
        + self.avoid_borders() * self.border_factor
    )

    velocity_norm = np.linalg.norm(self.velocity)
    if velocity_norm ⩽ 0.00001:
        self.velocity = np.random.random(2) * 2 - 1
        velocity_norm = np.linalg.norm(self.velocity)

    self.velocity ⊨ velocity_norm
    self.current_speed = min(velocity_norm, self.speed)
    new_pos = self.pos + self.velocity * self.current_speed
```

Noten que `self.speed` actúa como la velocidad máxima del boid. Además de las tres reglas de Reynolds, el agente también incorpora una regla de **evasión de bordes** que lo aleja de los límites del mundo cuando se acerca a ellos.

La clase `World` hereda de `mesa.Model` y los agentes y las consultas de vecindad, para lo cual utiliza una estructura de datos:

```
class World(mesa.Model):
    def __init__(self, population, width, height, speed, vision,
                 distance, spatial_method="quadtree", ... ):
        self.space = mesa.space.ContinuousSpace(width, height, True)
        # crear agentes con posiciones y velocidades aleatorias
        self.make_agents()

    def step(self):
        # reconstruir la estructura espacial cada paso
        # (los agentes se han movido desde el paso anterior)
        if self.spatial_method == "quadtree":
            self.qt = QuadTree(boundary, capacity=4, max_depth=8)
            for idx, boid in self.id_to_agent.items():
                self.qt.insert(boid.pos[0], boid.pos[1], idx)
        # ejecutar un paso para todos los agentes en orden aleatorio
        self.agents.shuffle_do("step")
```

En cada paso, la estructura espacial se reconstruye desde cero. Esto es necesario porque los boids se mueven entre pasos, por lo que la partición del espacio cambia. La alternativa sería actualizar la estructura de forma incremental (mover los puntos que cambiaron de cuadrante), pero la reconstrucción completa es más simple y, para las cantidades de agentes que manejamos, es suficiente.

Estructuras de datos espaciales

La implementación de BOIDS introduce un desafío computacional: para que cada agente determine su comportamiento, necesita identificar y evaluar sus vecinos. Con un enfoque ingenuo, cada boid examinaría a todos los demás agentes, lo que resulta en una complejidad computacional de $O(n^2)$ para cada paso de simulación. Esta limitación vuelve impracticable la simulación de poblaciones numerosas.

Para solucionar este problema se utilizan estructuras de datos espaciales como *quadtree* (árbol cuádruple), que organiza los datos espaciales de manera jerárquica. Un *quadtree* divide de forma recursiva el espacio bidimensional en cuatro cuadrantes de igual tamaño. Comienza con un rectángulo que abarca todo el área de simulación, y el algoritmo subdivide cada región cuando contiene más de un cierto número de elementos (capacidad máxima). Cada nodo interno del árbol tiene exactamente cuatro hijos, correspondientes a los cuadrantes noroeste (NO), noreste (NE), suroeste (SO) y sureste (SE).

La inserción de un punto en un *quadtree* sigue un algoritmo recursivo (Fig. 11). Primero se determina a qué cuadrante pertenece el punto mediante comparaciones simples. Si el cuadrante correspondiente ya contiene el máximo de elementos permitidos, se subdivide y los puntos existentes se redistribuyen entre los cuatro nuevos cuadrantes.

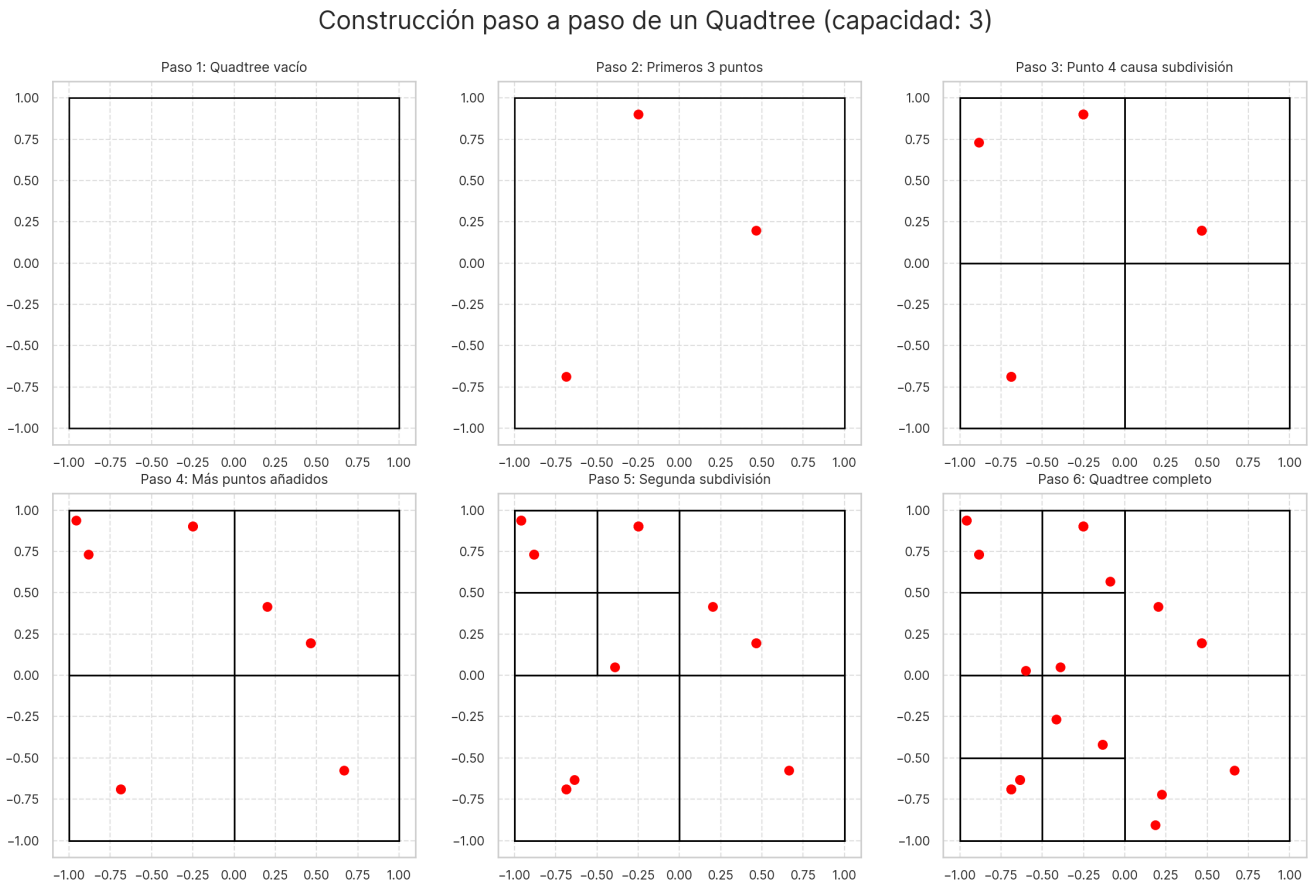
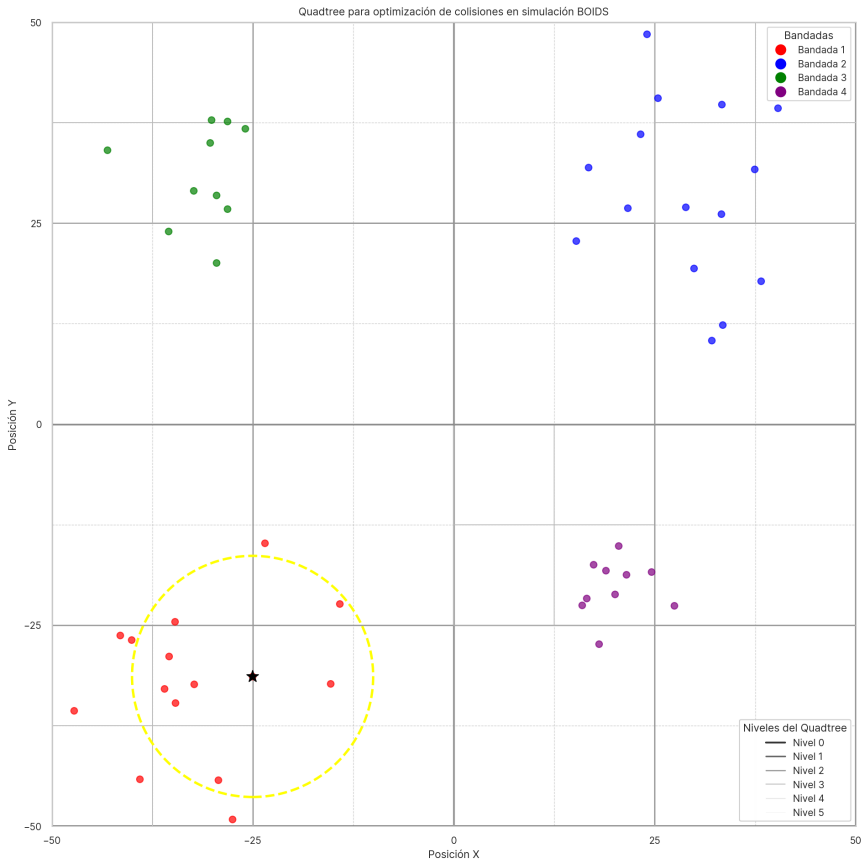


Figura 11. Ejemplo de inserción de puntos en *quadtree* de capacidad máxima de tres puntos por nodo. Cuando un nodo alcanza esta capacidad y se inserta un cuarto punto, el nodo se subdivide y los puntos se redistribuyen entre los cuatro nuevos cuadrantes.

Así, para encontrar todos los puntos dentro de cierto radio desde una posición, **solo necesitamos explorar los cuadrantes que intersectan con el círculo de búsqueda**. Esto reduce la cantidad de comparaciones necesarias, y lleva la complejidad de $O(n^2)$ a $O(\log n + k)$, donde k es el número de vecinos encontrados.

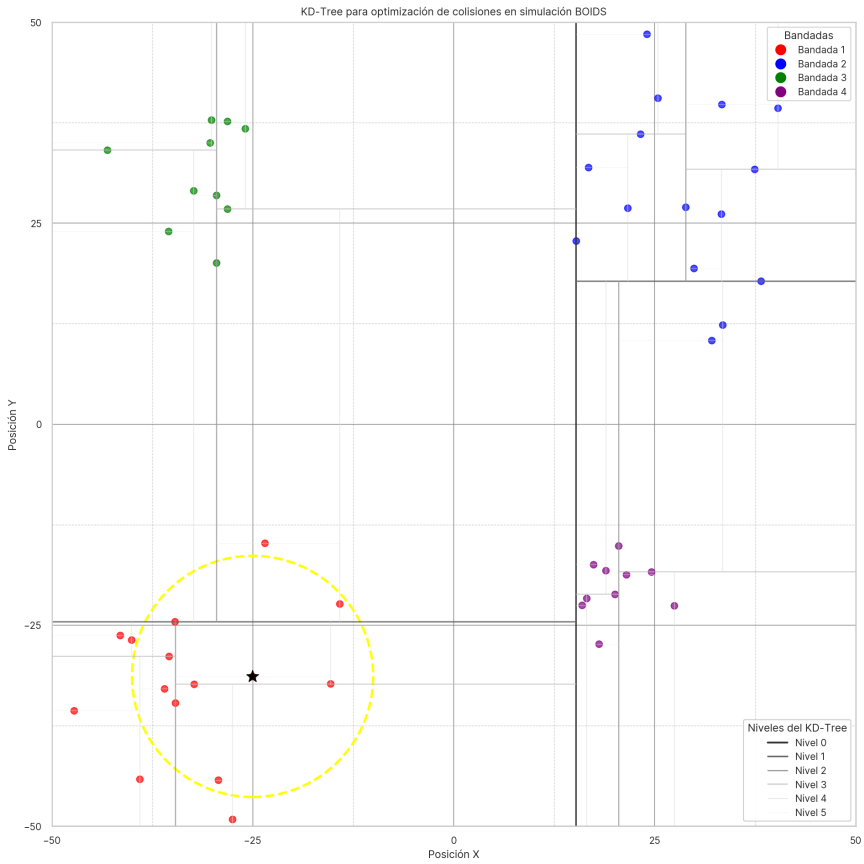
Ahora bien, debemos considerar que los criterios de subdivisión, como la capacidad máxima por cuadrante o la profundidad máxima del árbol, afectan el rendimiento. También debemos decidir entre reconstruir el *quadtree* en cada paso de simulación o actualizarlo de forma incremental (Fig. 12), lo que implica un balance entre simplicidad de implementación y eficiencia.

Figura 12. Ejemplo de aplicación de *quadtrees* para una simulación BOIDS.



El *quadtree* no es la única estructura espacial aplicable a simulaciones BOIDS. Estructuras como el *kd-tree* ofrecen ventajas similares con mecanismos diferentes de partición espacial. Mientras que el *quadtree* divide el espacio en regiones de igual tamaño (siempre en la mitad de cada coordenada), el *kd-tree* selecciona los puntos de división según la distribución de los datos (Fig. 13), de forma típica en la mediana de cada dimensión de forma alternada. Esta adaptabilidad puede resultar ventajosa en ciertos escenarios, aunque añade complejidad a la implementación.

Figura 13. Ejemplo de aplicación de *kd-trees* para una simulación BOIDS.



La elección entre estas estructuras depende de factores como la distribución espacial de los agentes, la frecuencia de operaciones de inserción y consulta, y las restricciones de memoria. En la práctica, ambas estructuras proporcionan mejoras de rendimiento que hacen viable la simulación de poblaciones numerosas.

El ejemplo `quadtree_demo` permite visualizar la estructura del *quadtree* y experimentar con consultas de rango:

```
uv run python caja_de_juguetes.py quadtree_demo
```

En esta demostración, el usuario agrega puntos con el clic izquierdo y observa cómo el *quadtree* se subdivide para acomodarlos. Al mover el puntero, un círculo de consulta muestra qué cuadrantes se visitan (resaltados en azul) y cuáles se descartan (en gris).

Visualización

La representación visual de una simulación BOIDS es necesaria para la validación, interpretación y análisis de los resultados. Dependiendo de los objetivos, podemos adoptar diferentes

estrategias de visualización y de programación (incluso se puede ejecutar en la GPU, como en la Fig. 14).

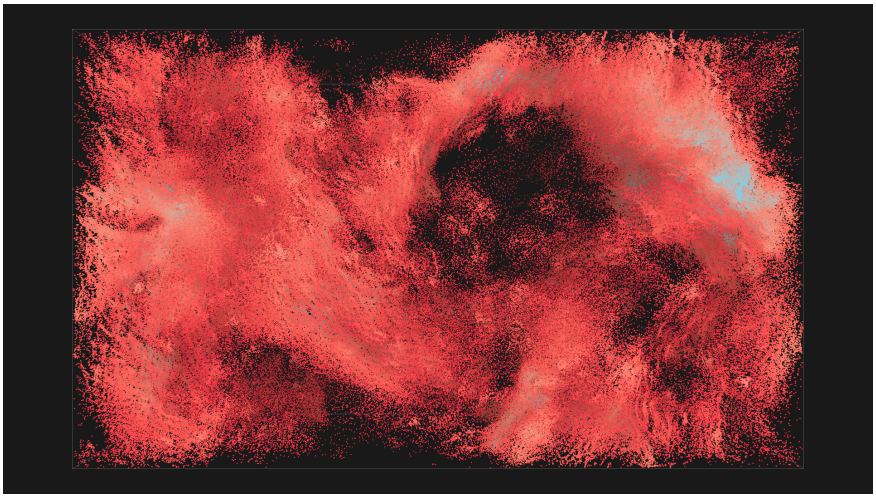


Figura 14. Una implementación de BOIDS en la GPU. Permite visualizar 2^{20} boids en tiempo real, utilizando la GPU y estructuras de datos especializadas como *octree* (una versión 3D del *quadtree*). Fuente: flocking.cpp.

Para la representación individual de agentes, se utilizan triángulos orientados en la dirección del movimiento. En simulaciones más elaboradas, pueden emplearse modelos tridimensionales simplificados que reflejen la naturaleza del organismo simulado: peces estilizados para cardúmenes o aves para bandadas. El color puede codificar propiedades como velocidad, energía o estado, lo que añade una dimensión adicional a la visualización. En nuestro ejemplo `boids_particles`, el color varía con la velocidad del boid: los más rápidos aparecen en tonos rojos y los más lentos en verdes.

La estructura espacial subyacente también puede visualizarse para comprender mejor el funcionamiento del algoritmo. Mostrar los cuadrantes del *quadtree* (activando la tecla T en `boids_particles`) ayuda a comprender cómo se organiza la simulación en el espacio. Visualizar los radios de percepción mediante círculos alrededor de cada agente (tecla V) ilustra el alcance de sus interacciones.

Para análisis más profundos, podemos implementar visualizaciones específicas: trazas de movimiento que muestran las trayectorias recientes (es decir, **ornitografías**), mapas de calor que revelan las regiones más transitadas, o representaciones visuales de *clusters* que destacan la formación de subgrupos dentro de la población. Estas técnicas analíticas transforman la simulación de una animación a una herramienta de investigación, una aplicación de visualización científica.

Extensiones avanzadas

El modelo BOIDS básico puede enriquecerse con extensiones que amplían su versatilidad y realismo. Incorporar comporta-

mientos adicionales como **evasión de obstáculos**, **seguimiento de objetivos** o **componentes aleatorios** permite modelar situaciones más complejas y realistas.

Los modelos de **liderazgo** introducen jerarquías en la estructura del grupo, donde ciertos agentes ejercen mayor influencia sobre el comportamiento colectivo. Esta extensión permite estudiar fenómenos como la propagación de información dentro del grupo o las dinámicas de toma de decisiones colectivas.

La interacción con el entorno añade otra capa de complejidad: **campos de fuerza externos** como viento o corrientes acuáticas, **respuesta a cambios ambientales** como variaciones de luz, o comportamientos específicos como **búsqueda de alimento** y **huida de depredadores**. Estas interacciones ambientales crean dinámicas más ricas y situadas en contexto.

Una extensión sofisticada es implementar **comportamientos adaptables**, donde las reglas que siguen los agentes no permanecen estáticas sino que evolucionan en respuesta a experiencias previas. Mediante algoritmos genéticos o técnicas de aprendizaje por refuerzo (*Reinforcement Learning*), los agentes pueden "descubrir" comportamientos óptimos para su entorno, lo que simula procesos evolutivos o de aprendizaje observados en organismos reales.

Estas extensiones transforman BOIDS de un modelo de simulación a una plataforma experimental para explorar la emergencia de comportamientos complejos a partir de interacciones simples, y conectan con campos tan diversos como la biología, la psicología, la robótica y la inteligencia artificial. Por eso, la próxima vez que observes una bandada de pájaros en el cielo, recuerda que además de su belleza natural hay principios computacionales igual de fascinantes.

2 Ejercicios

Formación en V para vuelos de larga distancia (Control 2, Otoño 2025, parte a de una pregunta)

Las aves migratorias (como los gansos) frecuentemente vuelan en formación V para conservar energía. Volar en la estela de otra implica menor resistencia del aire, permitiendo a cada ave ahorrar hasta un 30% de energía.

1. Defina una **regla de posicionamiento en V** donde:

    - Las aves intentan posicionarse en un ángulo aproximado de $45°$ detrás y a un lado de otra ave.
    - Hay una distancia lateral óptima respecto al ave delantera.

- El ave líder debe ser visible para la mayoría de la bandada.

Su respuesta debe incluir el diagrama de fuerzas que afecta a las aves y la definición de estas fuerzas.

2. El ave líder gasta más energía que las demás. Explique cómo definiría un método de **rotación de liderazgo**. Puede responder con texto libre o pseudo-código.

BOIDS discretos en grilla (Control 2, Primavera 2025)

En *New Rally-X*, los autos enemigos navegan por un laberinto siguiendo una grilla discreta. Cada celda puede contener máximo un auto, y el movimiento está restringido a las 4 direcciones cardinales (Norte, Sur, Este, Oeste).

La **distancia Manhattan** entre dos posiciones (x_1, y_1) y (x_2, y_2) en la grilla se define como:

$$d_M((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|$$

Esta métrica es natural para este problema porque representa exactamente el número mínimo de movimientos cardinales necesarios para ir de una celda a otra (sin considerar obstáculos).

Sea $\mathbf{p}_i = (x_i, y_i)$ la posición del auto i en la grilla y N_i el conjunto de vecinos dentro del radio r (medido en distancia Manhattan).

La regla de **separación** se define como:

$$\vec{F}_{sep} = \sum_{j \in N_i} \frac{\mathbf{p}_i - \mathbf{p}_j}{d_M(\mathbf{p}_i, \mathbf{p}_j)}$$

donde si $d_M(\mathbf{p}_i, \mathbf{p}_j) = 1$ (celda adyacente ocupada), la fuerza repulsiva es máxima.

a) Defina matemáticamente las otras dos reglas BOIDS adaptadas a la grilla:

Alineación: Adapte la ecuación continua $\vec{v}_{ali} = \frac{\sum_{j \in N_i} \vec{v}_j}{|N_i|}$ al caso discreto donde cada auto j tiene un vector de dirección $\vec{d}_j \in \{(0,1), (0,-1), (1,0), (-1,0)\}$ correspondiente a $\{N, S, E, O\}$.

Proponga:

$$\vec{F}_{ali} = \text{[su definición aquí]}$$

Cohesión: Adapte la ecuación continua $\vec{v}_{coh} = \frac{\sum_{j \in N_i} \mathbf{p}_j}{|N_i|} - \mathbf{p}_i$ al caso discreto. El centro de masa es:

$$\mathbf{c} = \frac{1}{|N_i|} \sum_{j \in N_i} \mathbf{p}_j$$

Proponga:

$$\vec{F}_{coh} = \text{[su definición aquí]}$$

Justifique por qué sus definiciones preservan el comportamiento esencial de las reglas originales.

b) Para convertir las fuerzas continuas a decisiones discretas:

Dado el vector de fuerza total:

$$\vec{F}_{total} = w_{sep}\vec{F}_{sep} + w_{ali}\vec{F}_{ali} + w_{coh}\vec{F}_{coh}$$

con pesos $w_{sep} = 0.5$, $w_{ali} = 0.3$, $w_{coh} = 0.2$.

1. Normalice $\vec{F}_{total}$ si $|\vec{F}_{total}| > 0$
2. Seleccione la dirección cardinal $d^* \in \{N, S, E, O\}$ que maximiza:

   $$d^* = \arg\max_{d \in \{N,S,E,O\}} \langle \vec{F}_{total}, \vec{d} \rangle$$

   donde $\langle \cdot, \cdot \rangle$ es el producto punto.

Escriba el algoritmo completo para calcular d^* dado un auto y sus vecinos.

c) Analice el comportamiento emergente:

1. **Radio de vecindad**: Si el laberinto tiene dimensiones $M \times N$, ¿qué valor de r recomendaría? Justifique considerando el trade-off entre información local y global.

2. **Métrica de cohesión discreta**: Defina una métrica C para medir la cohesión del grupo en espacio discreto:

   $$C = \frac{1}{|G|} \sum_{i \in G} \min_{j \in G, j \neq i} d_M(\mathbf{p}_i, \mathbf{p}_j)$$

   ¿Qué valores de C indican alta cohesión? ¿Cómo se compara con la dispersión en espacio continuo?

3. **Efecto de la discretización**: Explique cómo la restricción "una celda por auto" afecta la formación de patrones comparado con BOIDS continuos.

Referencias

[1] Bajec, I.L., Zimic, N. y Mraz, M. 2007. The computational beauty of flocking: boids revisited. *Mathematical and Computer Modelling of Dynamical Systems*. 13, 4 (2007), 331-347. https://doi.org/10.1080/13873950600883485.

[2] Hoetzlein, R.C. 2024. Flock2: A model for orientation-based social flocking. *Journal of Theoretical Biology*. 593, (2024), 111880. https://doi.org/10.1016/j.jtbi.2024.111880.

[3] Izquierdo, L.R., Izquierdo, S.S. y Sandholm, W.H. 2019. *Agent-Based Evolutionary Game Dynamics*. Independent.

[4] Reynolds, C.W. 1987. Flocks, herds and schools: A distributed behavioral model. *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987), 25-34.

[5] Ward, S. 2015. *Are changes in the lesser flamingo population a natural consequence of soda lake dynamics?* University of Southampton.