BỘ CÔNG THƯƠNG

TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

ĐỒ ÁN TỐT NGHIỆP

TRƯỜNG CÔNG NGHỆ THÔNG TIN

MÔ PHỎNG CHUYỂN ĐỘNG BẦY ĐÀN BẰNG THUẬT TOÁN BOIDS VÀ TỐI ƯU HÓA THUẬT TOÁN

CBHD: TS. Lê Thị Anh

Sinh viên: Trương Đức Huy

Mã số sinh viên: 2021608438

Hà Nội – 2025

LỜI CẢM ƠN

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

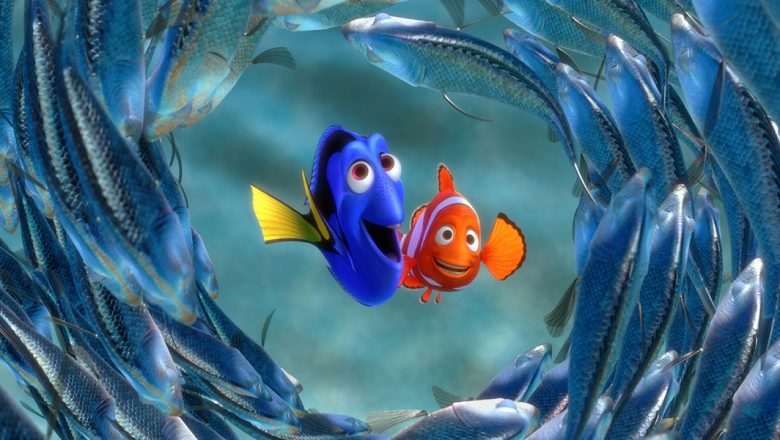
MỤC LỤC

LỜI MỞ ĐẦU

**Lý do chọn đề tài**

Thế giới tự nhiên luôn là nguồn cảm hứng bất tận cho khoa học và nghệ thuật. Một trong những hiện tượng ấn tượng và phức tạp nhất là chuyển động tập thể của các loài động vật, điển hình như bầy chim, đàn cá, hay đàn gia súc. Khả năng mô phỏng những chuyển động này một cách chân thực trong đồ họa máy tính không chỉ tạo ra hiệu ứng hình ảnh sống động mà còn mở ra nhiều ứng dụng tiềm năng trong các lĩnh vực giải trí, giáo dục và mô phỏng khoa học.

Thuật toán Boids, được giới thiệu lần đầu tiên bởi Craig W. Reynolds vào năm 1987, đã trở thành nền tảng kinh điển để mô phỏng hành vi bầy đàn. Điểm cốt lõi của thuật toán nằm ở tính chất phân tán: mỗi cá thể (boid) chỉ tuân theo ba quy tắc tương tác đơn giản với các cá thể lân cận, nhưng khi tập hợp lại, chúng tạo ra một chuyển động tổng thể có tổ chức và phức tạp đến bất ngờ. Phương pháp này đã được ứng dụng rộng rãi và tạo ra những hiệu ứng đột phá trong nhiều sản phẩm nổi tiếng. Ví dụ, cảnh cá bơi trong bộ phim Finding Nemo (2003) hay những đám chim di chuyển có tổ chức trong The Lion King (1994) đều là những ví dụ điển hình về sự hiệu quả của Boids trong lĩnh vực hoạt hình điện ảnh. Trong ngành công nghiệp game, Boids cũng được áp dụng rộng rãi để tạo ra các hiệu ứng đám đông, ví dụ như trong loạt game Total War để mô phỏng các đơn vị quân đội khổng lồ, hay trong Assassin's Creed để làm cho các thành phố trở nên sống động hơn với các đám đông người di chuyển tự nhiên.



Hình … Khung cảnh đàn cá bơi trong phim Đi Tìm Nemo

Trong quá trình triển khai thực tế trên Unity, tôi nhận thấy một rào cản lớn về hiệu năng. Vấn đề này xuất phát từ bản chất của thuật toán Boids gốc, nơi mỗi boid phải tính toán tương tác với mọi boid khác, dẫn đến độ phức tạp tính toán tăng theo cấp số nhân khi số lượng boids tăng lên. Điều này gây ra sự suy giảm đáng kể về tốc độ khung hình (frame rate). Các nhà phát triển game và các nhà làm kĩ xảo thường phải đối mặt với thách thức này khi muốn tạo ra các cảnh mô phỏng đám đông quy mô lớn. Việc tối ưu hóa thuật toán Boids trở nên cấp bách để có thể tạo ra những cảnh quay chân thực và sống động mà không bị hạn chế bởi giới hạn kỹ thuật.

Với những lý do trên, tôi đã quyết định chọn đề tài này. Mục tiêu của đồ án không chỉ dừng lại ở việc triển khai lại thuật toán Boids mà còn tập trung vào việc tối ưu hóa hiệu năng bằng các kỹ thuật lập trình tiên tiến trên Unity. Bằng cách so sánh hiệu quả của các phương pháp tối ưu hóa khác nhau—từ việc sử dụng Jobs System và Burst Compiler để tận dụng sức mạnh đa luồng của CPU, cho đến việc sử dụng các cấu trúc dữ liệu không gian như Quadtree và Spatial Hashing để tăng tốc độ truy vấn các boid lân cận và áp dụng GPU Instancing để tối ưu hóa việc vẽ. Tôi hy vọng sẽ cung cấp một giải pháp toàn diện và thực tế. Đây là một nỗ lực để vượt qua rào cản kỹ thuật của thuật toán kinh điển, từ đó mở rộng khả năng ứng dụng của nó trong các dự án thực tế, đặc biệt là trong môi trường phát triển game và phim hiện đại, nơi hiệu năng luôn là yếu tố then chốt.

**Mục tiêu của đề tài**

Mục tiêu của đồ án này là mô phỏng bầy đàn dựa trên thuật toán Boids, đồng thời tối ưu hóa hiệu năng để có thể xử lý một số lượng lớn các cá thể một cách hiệu quả trên nền tảng Unity. Để đạt được mục tiêu này, tôi đã chia thành các mục tiêu cụ thể như sau:

Mục tiêu 1: Nghiên cứu và hiểu sâu về thuật toán Boids gốc của Craig W. Reynolds, bao gồm ba quy tắc cơ bản (tránh va chạm, khớp vận tốc, tập trung đàn) và các cơ chế mô phỏng vật lý liên quan.

Mục tiêu 2: Triển khai một phiên bản cơ bản của thuật toán Boids trên Unity để làm nền tảng so sánh hiệu năng.

Mục tiêu 3: Nghiên cứu các công nghệ và kỹ thuật tối ưu hóa hiệu năng trên Unity, bao gồm:

Sử dụng các cấu trúc dữ liệu không gian như Quadtree và Spatial Hashing để tăng tốc độ truy vấn các boids lân cận, giảm độ phức tạp tính toán từ O(N²) xuống O(N log N) hoặc O(N).

Áp dụng mô hình lập trình đa luồng bằng Jobs System và Burst Compiler để phân phối các tác vụ tính toán tương tác của các boids trên nhiều lõi CPU.

Sử dụng GPU Instancing để tối ưu hóa việc vẽ hàng nghìn đối tượng cùng một lúc, giảm thiểu số lượng lệnh gọi vẽ (draw call) đến GPU.

Mục tiêu 4: Thực hiện các thử nghiệm hiệu năng trên từng phiên bản được tối ưu hóa. Thu thập và phân tích dữ liệu về tốc độ khung hình (FPS) và thời gian xử lý (computation time) khi số lượng boids tăng dần.

Mục tiêu 5: So sánh hiệu năng giữa phiên bản gốc và các phiên bản đã được tối ưu hóa để làm rõ sự cải thiện về hiệu suất, từ đó đưa ra kết luận và đề xuất các phương án triển khai tối ưu nhất cho các ứng dụng thực tế.

**Nội dung nghiên cứu**

Nội dung nghiên cứu của đề tài tập trung vào việc tìm hiểu và triển khai các giải pháp để cải thiện hiệu năng của thuật toán boids.

Tìm hiểu lý thuyết: Nghiên cứu sâu về nguyên lý hoạt động của thuật toán boids và các quy tắc cơ bản (separation, alignment, cohesion). Đồng thời, tìm hiểu về các cấu trúc dữ liệu không gian như Spatial Hash và Quadtree để tăng tốc quá trình tìm kiếm các boids lân cận.

Nghiên cứu công nghệ Unity: Tìm hiểu các công nghệ của Unity hỗ trợ tối ưu hóa hiệu năng, bao gồm:

* GPU Instancing: Kỹ thuật render hàng loạt các đối tượng giống nhau bằng một lệnh vẽ duy nhất, giảm thiểu lệnh gọi (draw call) đến CPU.
* Unity Jobs System: Cho phép thực hiện các tác vụ tính toán nặng trên các luồng xử lý riêng biệt (multi-threading), tránh gây tắc nghẽn luồng chính.
* Burst Compiler: Một trình biên dịch mạnh mẽ giúp chuyển đổi mã C# sang mã máy hiệu quả, tăng tốc đáng kể các tính toán phức tạp.

Triển khai thực nghiệm: Xây dựng các mô hình mô phỏng boids riêng biệt cho từng kỹ thuật tối ưu (tức mỗi scene là một phiên bản tối ưu khác nhau) để tiến hành đo lường hiệu năng.

Phân tích và đánh giá: Thu thập dữ liệu từ các thử nghiệm và sử dụng các công cụ profiling của Unity để phân tích sự cải thiện về tốc độ khung hình và thời gian xử lý của mỗi phương pháp.

**Bố cục đề tài**

Ngoại trừ các phần lời mở đầu, kết luận và tài liệu tham khảo báo cáo được chia thành bố cục như sau:

Chương 1: Cơ sở lý thuyết

Chương 2: Triển khai thuật toán và các phương pháp tối ưu

Chương 3: Đánh giá và kết luận

**CHƯƠNG 1 : CƠ SỞ LÝ THUYẾT**

**Thuật toán boids**

Thuật toán Boids, được giới thiệu bởi Craig W. Reynolds trong bài báo "Flocks, Herds, and Schools: A Distributed Behavioral Model" (1987), là một mô hình mô phỏng hành vi bầy đàn được tạo ra bằng cách mô phỏng hành vi của các cá thể riêng lẻ. Mô hình này đi ngược lại các phương pháp điều khiển tập trung, thay vào đó, mỗi thực thể (gọi là boid) hành động độc lập theo một bộ quy tắc đơn giản. Chuyển động tổng hợp của cả đàn là kết quả của sự tương tác phức tạp giữa các hành vi đơn giản này, mô phỏng một cách chân thực cách các loài động vật trong tự nhiên di chuyển theo đàn.

**Mô hình hành vi phân tán**

Mô hình Boids dựa trên khái niệm hành vi phân tán, nơi mà không có bất kỳ một thực thể "lãnh đạo" nào điều khiển toàn bộ bầy đàn. Mỗi boid được xem như một tác nhân độc lập có khả năng tự đưa ra quyết định di chuyển dựa trên nhận thức về môi trường xung quanh, bao gồm các boids lân cận và các chướng ngại vật (nếu có). Thay vì lập trình từng đường đi cụ thể, mô hình này cho phép các boids tự điều hướng, dẫn đến một hệ thống tự tổ chức, nơi các mô hình chuyển động phức tạp xuất hiện một cách tự nhiên từ các tương tác đơn giản.

**Ba quy tắc điều hướng cơ bản**

Để xác định hướng đi của mình, mỗi boid sẽ liên tục tính toán và kết hợp ba quy tắc điều hướng cơ bản. Mỗi quy tắc này sẽ tạo ra một vector lực riêng, và tổng hợp của ba vector này sẽ xác định hướng di chuyển tiếp theo của boid.

Quy tắc tách rời (Separation)

Ảnh có chứa vòng tròn, biểu đồ, đồng hồ, hàng

Nội dung do AI tạo ra có thể không chính xác.

Quy tắc tách rời yêu cầu mỗi boid phải duy trì một khoảng cách tối thiểu với các boids lân cận. Mục tiêu của quy tắc này là ngăn chặn sự va chạm giữa các boid và giữ cho chúng không bị dồn vào một điểm. Boid sẽ tính toán vị trí của các boid lân cận và tạo ra một lực đẩy hướng ra xa chúng. Lực này càng mạnh khi khoảng cách giữa boid và các boid lân cận càng gần.

Quy tắc căn chỉnh (Alignment)

Ảnh có chứa vòng tròn, nghệ thuật gấp giấy origami, thiết kế

Nội dung do AI tạo ra có thể không chính xác.

Quy tắc căn chỉnh yêu cầu mỗi boid điều chỉnh hướng và tốc độ của mình sao cho phù hợp với hướng và tốc độ trung bình của các boids lân cận. Điều này giúp cả đàn duy trì cùng một hướng di chuyển và tốc độ ổn định, tạo ra cảm giác cả đàn đang bay hoặc bơi cùng nhau.

Quy tắc kết nối (Cohesion)

Ảnh có chứa vòng tròn, biểu đồ, nghệ thuật gấp giấy origami

Nội dung do AI tạo ra có thể không chính xác.

Quy tắc kết nối yêu cầu mỗi boid di chuyển về phía trọng tâm của các boids lân cận. Mục tiêu là giữ cho bầy đàn không bị tan rã và tạo ra sự gắn kết về mặt hình học. Mỗi boid sẽ tính toán vị trí trung bình của các boids trong phạm vi quan sát của nó và tạo ra một lực kéo hướng về vị trí đó.

**Tùy chỉnh hành vi đàn Boids**

* Trọng số cho các quy tắc: Thay vì áp dụng các quy tắc Tách rời, Căn chỉnh và Kết nối với sức ảnh hưởng như nhau, người lập trình có thể gán các trọng số riêng biệt cho từng quy tắc để điều chỉnh đặc tính của bầy đàn.
* Ưu tiên hành vi: Một số tình huống đặc biệt, boid có thể ưu tiên tránh va chạm và bỏ qua tính toán các quy tắc.
* Giới hạn tốc độ tối thiểu và tối đa: Giúp duy trì tốc độ của đàn boids ổn định
* Các lực tác động bên ngoài: Có thể tổng hợp lực này cùng 3 quy tắc của boid để hướng boid di chuyển theo hướng mong muốn của người lập trình.

**Vì sao lựa chọn thuật toán boids**

Một số thuật toán khác:

**Particle Swarm Optimization (PSO)**

Cơ chế chính: Mỗi cá thể (particle) là một điểm trong không gian tìm kiếm, cá thể cập nhật vị trí dựa trên:

* Vị trí tốt nhất cá nhân (personal best)
* Vị trí tốt nhất của bầy (global best)

Ưu điểm:

* Tối ưu hóa hiệu quả trong không gian nhiều chiều.
* Không yêu cầu gradient, dễ áp dụng cho các bài toán tối ưu phi tuyến.

Nhược điểm:

* Không trực tiếp mô phỏng chuyển động vật lý tự nhiên.
* Dễ bị mắc kẹt ở cực trị địa phương nếu bầy thiếu đa dạng.

Ứng dụng: Tối ưu hóa, học máy, lập lịch.

**Flocking with Potential Fields**

Cơ chế chính: Mỗi cá thể di chuyển theo trường lực:

* Thu hút về trung tâm đàn (cohesion)
* Đẩy nhau ra (separation)
* Thu hút hoặc tránh các mục tiêu/vật cản

Ưu điểm:

* Linh hoạt, có thể thêm nhiều lực tương tác phức tạp.
* Tối ưu cho môi trường nhiều vật cản.

Nhược điểm:

* Tính toán lực phức tạp, khó kiểm soát chuyển động tự nhiên.
* Dễ gây hiện tượng “rung lắc” nếu các lực không cân bằng.

Ứng dụng: Robot swarm, mô phỏng vật lý, game.

**Leader-Follower (Thuật toán dẫn dắt)**

Cơ chế chính: Một hoặc nhiều cá thể dẫn hướng, các cá thể khác theo sau:

* Leader di chuyển theo đường định sẵn hoặc tối ưu
* Followers giữ khoảng cách và hướng đi với leader

Ưu điểm:

* Kiểm soát dễ dàng hướng đi chung.
* Tốt cho robot swarm hoặc mô phỏng đàn cần đi theo mục tiêu.

Nhược điểm:

* Ít tự nhiên hơn Boids vì phụ thuộc leader.
* Nếu leader gặp vấn đề, cả đàn ảnh hưởng.

Ứng dụng: Robot swarm, điều khiển đàn, mô phỏng quân sự.

**Boids**

Cơ chế chính: Mỗi cá thể (boid) tuân theo 3 quy tắc Separation, Alignment, Cohesion.

Ưu điểm:

* Mô phỏng chuyển động bầy đàn rất tự nhiên
* Dễ hiểu, dễ triển khai.
* Tính toán hoàn toàn phân tán (không cần quản lý trung tâm).

Nhược điểm:

* Khi số lượng boid lớn (hàng nghìn) dễ chậm, do cần tính toán khoảng cách giữa các boid.

Ứng dụng: Mô phỏng đàn chim, cá, AI cho game, phim.

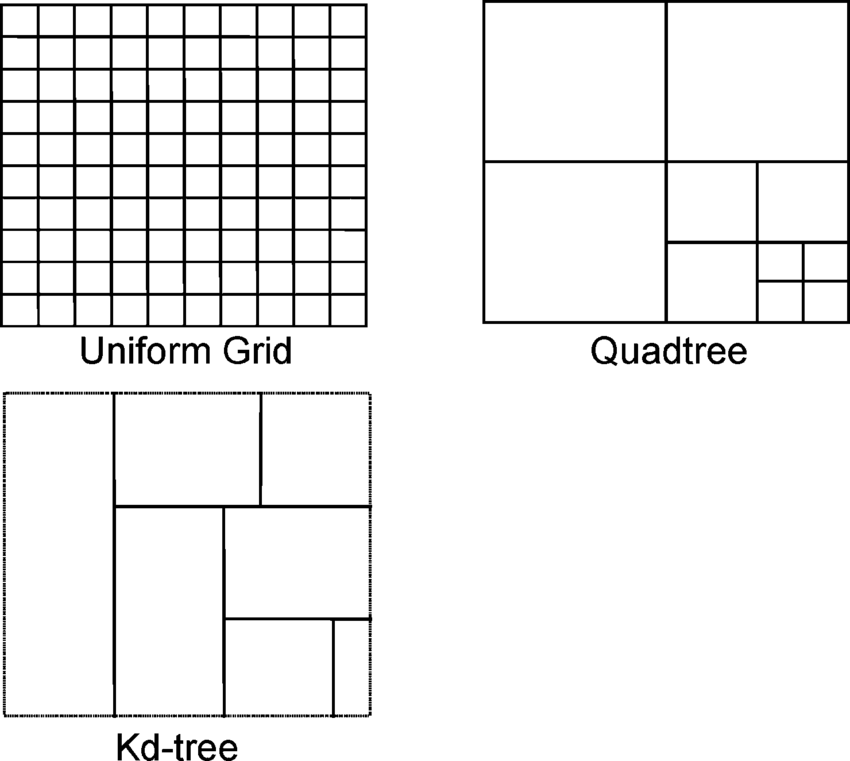
**Kết luận**Trong bối cảnh hiện tại, không có một thuật toán nào là "tốt nhất" cho mọi trường hợp. Nhưng với bài toán mô phỏng chuyển động bầy đàn và phân tán thì có thể nói Boids là một sự lựa chọn tốt, nhưng chỉ thực sự hiệu quả khi kết hợp với các kỹ thuật tối ưu hóa hiện đại. Trong đồ án lần này tôi sẽ sử dụng các cấu trúc dữ liệu chia không gian như Spatial Hashing, Quadtree, các công nghệ tính toán song song như Unity Jobs, Burst Compiler và GPU instancing để giải quyết vấn đề hiệu năng của thuật toán Boids

Chương 2: Các phương pháp tối ưu thuật toán

Spatial Partitioning

Spatial Partitioning là gì

Spatial Partitioning (phân vùng không gian) là kỹ thuật chia nhỏ một không gian lớn thành các vùng nhỏ hơn nhằm quản lý, tổ chức và xử lý dữ liệu trong không gian một cách hiệu quả. Thay vì coi toàn bộ không gian như một khối thống nhất, việc phân vùng giúp xác định nhanh những đối tượng nào nằm gần nhau, từ đó giảm số lượng phép tính cần thực hiện trong các bài toán liên quan đến tìm kiếm, va chạm hay hiển thị.



Vì sao cần phân vùng không gian

Trong nhiều ứng dụng, số lượng đối tượng trong không gian có thể rất lớn (ví dụ hàng nghìn đến hàng triệu đối tượng). Nếu áp dụng cách tiếp cận “brute force” – tức kiểm tra tất cả cặp đối tượng với nhau – thì độ phức tạp sẽ là O(n²), dẫn đến hiệu năng giảm mạnh khi n tăng. Spatial Partitioning giúp:

Giảm số phép tính không cần thiết: chỉ so sánh các đối tượng trong cùng vùng hoặc vùng lân cận thay vì toàn bộ.

Tăng tốc độ tìm kiếm đối tượng lân cận: hữu ích trong bài toán nearest neighbor.

Tối ưu tài nguyên tính toán: tiết kiệm CPU/GPU và bộ nhớ.

Cải thiện trải nghiệm người dùng: trong game hoặc mô phỏng, việc duy trì tốc độ khung hình cao là rất quan trọng.

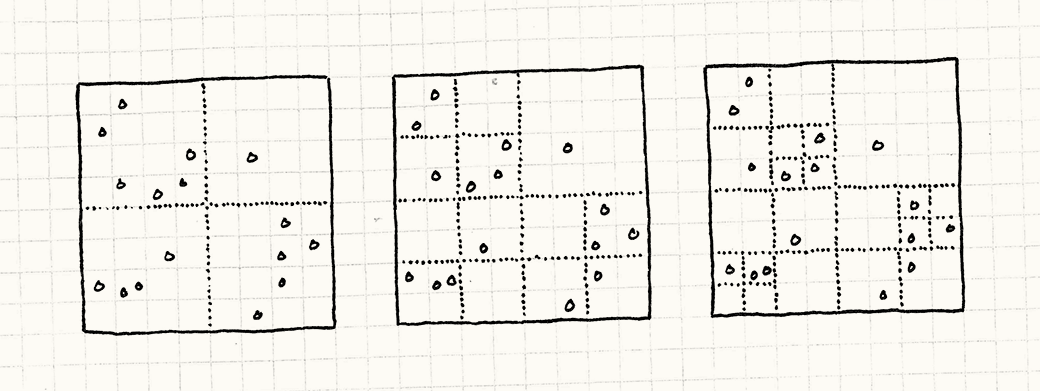
Từ những lợi ích nêu trên có thể thấy rằng Spatial Partitioning là một giải pháp quan trọng để tối ưu hiệu năng. Tuy nhiên, có nhiều cách khác nhau để thực hiện phân vùng không gian, tùy vào đặc thù phân bố của đối tượng. Trong đồ án lần này tôi sẽ sử dụng 2 cấu trúc dữ liệu phân vùng không gian phổ biến và hiệu quả nhất trong không gian 2 chiều là Quadtree và Spatial Hash.

Quadtree

Giới thiệu

Quadtree là một cấu trúc dữ liệu dạng cây được sử dụng để phân vùng không gian hai chiều. Ý tưởng cơ bản của quadtree là chia không gian ban đầu thành 4 phần nhỏ hơn (gọi là các “quadrant” hay ô con). Mỗi nút bên trong cây có tối đa bốn nút con, tương ứng với 4 vùng trong không gian. Quá trình này được lặp lại đệ quy cho đến khi mỗi ô chỉ còn chứa một số lượng đối tượng nhỏ hơn ngưỡng cho phép, hoặc đạt đến độ sâu nhất định.

Cấu trúc này được Raphael Finkel và J.L. Bentley giới thiệu năm 1974 và từ đó đến nay được áp dụng rộng rãi trong nhiều lĩnh vực như xử lý ảnh, nén dữ liệu, đồ họa máy tính, GIS, cũng như các bài toán tìm kiếm không gian và mô phỏng vật lý



Đặc điểm của Quadtree

Phân rã không gian thích nghi (adaptive subdivision): Không gian được chia nhỏ một cách linh hoạt, tập trung chi tiết ở các khu vực có nhiều đối tượng, trong khi các vùng trống hoặc ít dữ liệu được giữ ở mức phân giải thấp.

Quản lý đối tượng theo vùng: Mỗi nút lá (leaf node) quản lý một tập hợp đối tượng trong vùng tương ứng. Khi số lượng vượt quá ngưỡng, nút sẽ tiếp tục phân tách thành 4 nút con.

Hỗ trợ tìm kiếm nhanh: Nhờ tổ chức theo cấu trúc cây, quadtree giúp việc xác định vùng chứa một đối tượng hoặc tìm kiếm các đối tượng lân cận được thực hiện nhanh hơn so với quét toàn bộ dữ liệu.

Ứng dụng trong thuật toán Boids

Trong bài toán mô phỏng boids, một vấn đề lớn là chi phí tính toán va chạm và xác định láng giềng. Nếu không có phân vùng, mỗi cá thể (boid) phải kiểm tra khoảng cách đến tất cả cá thể khác → độ phức tạp O(n²). Điều này trở nên không khả thi khi số lượng boid tăng lên hàng chục nghìn.

Khi áp dụng quadtree, mỗi boid chỉ cần tìm kiếm láng giềng trong cùng ô hoặc các ô lân cận, thay vì trên toàn bộ không gian. Cách tiếp cận này giúp giảm đáng kể số lượng phép so sánh cần thực hiện.

Nói cách khác, quadtree đóng vai trò như một bộ lọc không gian (spatial filter), loại bỏ các đối tượng “xa không cần thiết” và chỉ giữ lại các đối tượng “có khả năng tương tác”. Đây là một trong những kỹ thuật then chốt để tối ưu hóa hiệu năng trong mô phỏng swarm AI nói chung và boids nói riêng.

Cách Quadtree hoạt động trong thuật toán Boids

Khởi tạo cây Quadtree

Ban đầu, toàn bộ không gian mô phỏng được coi là một hình chữ nhật gốc (root node).

Nút gốc có thông tin về kích thước (bounds), sức chứa tối đa (capacity), độ sâu hiện tại (depth) và độ sâu cho phép lớn nhất (max depth).

Ở giai đoạn này, cây chưa có phân nhánh, tất cả boid sẽ được thử chèn vào nút gốc

Chèn tất cả boid vào Quadtree

Với mỗi boid, Quadtree lấy tọa độ vị trí của boid để xác định boid thuộc về vùng nào.

Nếu số lượng boid trong một nút chưa vượt quá sức chứa (capacity), boid sẽ được lưu giữ trực tiếp trong nút đó.

Nếu số lượng vượt ngưỡng, nút sẽ được phân tách (Subdivide) thành 4 nút con (tương ứng với 4 vùng không gian nhỏ hơn). Toàn bộ boid đang lưu trong nút hiện tại sẽ được phân phối lại cho các nút con thích hợp.

Quá trình này tiếp tục đệ quy cho đến khi vùng không gian đủ nhỏ (≤ minSize), hoặc đạt đến độ sâu tối đa (maxDepth).

Truy vấn láng giềng

Khi một boid cần xác định các cá thể xung quanh để áp dụng ba quy tắc (Separation, Alignment, Cohesion), nó không cần duyệt toàn bộ danh sách.

Thay vào đó, Quadtree thực hiện truy vấn trong một vùng hình chữ nhật (range query) bao quanh boid đó với bán kính tương ứng với tầm ảnh hưởng (perception radius).

Quadtree sẽ chỉ duyệt qua các nút có giao với vùng tìm kiếm này. Nhờ vậy, số lượng phép kiểm tra thực tế giảm đáng kể, vì các vùng xa hoàn toàn được bỏ qua.

Cập nhật và tái sử dụng Quadtree mỗi khung hình

Vì boid luôn di chuyển, cây Quadtree cần được làm mới sau mỗi khung hình (frame).

Để tối ưu, tôi sử dụng Object Pooling: sau mỗi frame, các nút Quadtree không còn dùng sẽ được trả về bộ nhớ đệm để tái sử dụng, thay vì tạo và xóa liên tục. Cách này giúp giảm chi phí cấp phát bộ nhớ và tăng hiệu năng.