**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**

Nội dung

[LỜI MỞ ĐẦU 3](#_Toc208410215)

[1. Lý do chọn đề tài 3](#_Toc208410216)

[2. Mục tiêu của đề tài 3](#_Toc208410217)

[3. Nội dung nghiên cứu 3](#_Toc208410218)

[4. Bố cục đề tài 3](#_Toc208410219)

[CHƯƠNG 1 : TỔNG QUAN VỀ THUẬT TOÁN BOIDS 4](#_Toc208410220)

# LỜI MỞ ĐẦU

# Lý do chọn đề tài

# Mục tiêu của đề tài

# Nội dung nghiên cứu

# 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: Tổng quan về thuật toán boids
* Chương 2: Các phương pháp tối ưu thuật toán
* Chương 3: Cài đặt và đánh giá kết quả

# CHƯƠNG 1 : TỔNG QUAN VỀ THUẬT TOÁN BOIDS

## Giới thiệu chung

Thuật toán Boids được Craig Reynolds đề xuất năm 1986 nhằm mô phỏng hành vi bầy đàn của các loài chim và cá trong tự nhiên. “Boid” là viết tắt của từ “bird-oid object” (vật thể giống chim).

Thuật toán này không cố gắng lập trình trực tiếp toàn bộ quỹ đạo phức tạp của một đàn chim, mà thay vào đó, mô phỏng dựa trên các quy tắc đơn giản áp dụng cho từng cá thể. Từ những quy tắc cục bộ này, hành vi tập thể phức tạp của cả đàn sẽ tự nhiên xuất hiện.

## Cách thuật toán boids hoạt động

Mỗi cá thể trong đàn được gọi là boid, mỗi boid có 1 vector vận tốc và vị trí. Khi không có các boid khác xung quanh thì boid di chuyển theo hướng của vector vận tốc với tốc độ bằng độ lớn của vector vận tốc. Khi có các boid xung quanh thì boid được điều khiển bởi 3 quy tắc sau.

### Quy tắc Separation

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

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

* **Định nghĩa**

Separation là quy tắc giúp mỗi cá thể (boid) tránh va chạm với các cá thể khác trong vùng lân cận bằng cách tạo ra một lực đầy có hướng tránh xa các boid khác

* **Nguyên tắc hoạt động**
  + Xác định bán kính rS
  + Với mỗi boid, kiểm tra khoảng cách đến các boid xung quanh
  + Nếu boid nào nằm trong bán kính rs, boid đó sẽ tạo ra 1 vector lực đẩy đến boid đang xét, khoảng cách càng gần lực đẩy càng mạnh
  + Tổng hợp tất cả các vector lực đẩy

### Quy tắc Alignment

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

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

* **Định nghĩa**

Separation là quy tắc giúp mỗi cá thể (boid) tránh va chạm với các cá thể khác trong vùng lân cận bằng cách tạo ra một lực đầy có hướng tránh xa các boid khác

* **Nguyên tắc hoạt động**
  + Xác định bán kính ra
  + Với mỗi boid, tìm tất cả các boid nằm trong bán kính này.
  + Tổng hợp vector vận tốc của các boid lân cận rồi chia cho số lượng các boid lân cận để lấy vector vận tốc trung bình.

### Quy tắc Cohesion

* **Định nghĩa**

Cohesion là quy tắc giúp mỗi cá thể (boid) có xu hướng di chuyển về phía trung tâm đàn để giữ cho đàn không bị phân tán rời rạc.

* **Nguyên tắc hoạt động**
  + Xác định bán kính rc
  + Với mỗi boid, tìm tất cả các boid nằm trong bán kính này.
  + Tính vị trí trung bình của nhóm boid lân cận bằng cách tổng hợp vector vị trí của các boid lân cận rồi chia cho số lượng boid lân cận.
  + Boid đang xét tạo ra một vector lực hút hướng về vị trí trung bình đó.

### Tổng hợp các quy tắc

Sau khi áp dụng 3 quy tắc ta được 3 vector lực cho từng quy tắc, tổng hợp 3 lực này sẽ ra lực tác dụng lên boid. Có thể dùng thêm các biến trọng số để thay đổi độ lớn từng lực theo ý muốn

A screenshot of a computer screen

AI-generated content may be incorrect.

Theo định luật II Newton:

= m.

Thuật toán boid không quan tâm đến khối lượng của từng boid nên ta coi m = 1, khi đó vector lực tác dụng lên boid sẽ cung cấp cho boid một vector gia tốc có hướng và độ lớn bằng vector lực.

Vector vận tốc tại thời điểm t:

(t) = 0 + .t

Áp dụng trong mô phỏng máy tính:

new = odd + .

## Ứng dụng của thuật toán Boids

Thuật toán Boids được ứng dụng rộng rãi trong nhiều lĩnh vực:

* Đồ họa máy tính & phim ảnh: tạo hiệu ứng đàn chim, đàn cá, đám đông (ví dụ: trong các bộ phim hoạt hình của Disney/Pixar).
* Trí tuệ nhân tạo trong game: điều khiển NPC theo nhóm, mô phỏng đám đông trong game chiến thuật hay sinh tồn.
* Mô phỏng khoa học: nghiên cứu hành vi tập thể của động vật, xe cộ, hoặc con người trong các tình huống khẩn cấp.
* Robot swarm: áp dụng cho nhóm robot tự hành phối hợp với nhau.

## Vấn đề hiệu năng khi mô phỏng thuật toán và phương pháp tối ưu

Mặc dù các quy tắc của Boids đơn giản, nhưng khi số lượng cá thể lớn (hàng nghìn đến hàng chục nghìn), chi phí tính toán tăng rất nhanh. Mỗi boid cần tính toán khoảng cách và vận tốc tương đối với tất cả các boid khác → độ phức tạp O(n²). Điều này khiến mô phỏng bị chậm khi số lượng cá thể tăng.

Vì vậy, trong đồ án lần này tôi sẽ sử dụng các hướng tối ưu sau để cải thiện hiệu năng:

* **Spatial Partitioning**: chia không gian bằng **Spatial Hashing**, **Quadtree** để giảm số lượng so sánh.
* **Tận dụng GPU**: **GPU instancing** để render song song.
* **Unity DOTS**: là một framework của unity áp dụng kiến trúc **Entity Component System (ECS)** cùng với **Job System** và **Burst Compiler** để tối ưu bộ nhớ, xử lý đa luồng và tăng tốc độ mô phỏng.

# 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 (Vector2) 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
* Cập nhật và tái sử dụng Quadtree mỗi khung hình