ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

ZERO-TO-HERO PROJECT GUIDE
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

ĐỀ TÀI:
MÔ PHỎNG VŨ TRỤ 3D (N-BODY SIMULATION)
THUẬT TOÁN BARNES-HUT VÀ CẤU TRÚC OCTREE

Nhóm thực hiện: [Tên Nhóm Của Bạn]
Thành viên:
1. [Tên thành viên 1] - Trưởng nhóm (Algorithm Lead)
2. [Tên thành viên 2] - Physics & Math Lead
3. [Tên thành viên 3] - Graphics & Optimization Lead

Giảng viên hướng dẫn: [Tên Giảng Viên]

Học kỳ 2 - Năm học 2025-2026

Mục lục

Phần I

Tổng quan và Mục tiêu Dự án

Chương 1

Giới thiệu Bài toán N-Body

1.1 Bài toán N-Body là gì?

Bài toán N-Body là bài toán **kinh điển** trong vật lý tính toán và thiên văn học. Mục tiêu là mô phỏng sự tương tác hấp dẫn giữa N vật thể (thiên thể, hạt, hoặc ngôi sao) trong không gian 3D.

[title=🚀 Ứng dụng thực tế]

- **Thiên văn học:** Mô phỏng sự hình thành thiên hà, va chạm giữa các thiên hà
- **Vật lý hạt:** Mô phỏng plasma, động lực học phân tử
- **Game Development:** Hệ thống particle nâng cao, hiệu ứng vũ trụ
- **Machine Learning:** Graph Neural Networks cho physical simulation

1.2 Vấn đề với Brute-Force

Với phương pháp tính toán trực tiếp (Brute-force), mỗi vật thể phải tính lực tác dụng từ **tất cả** $(N-1)$ vật thể còn lại:

$$\text{Số phép tính} = N \times (N-1) = N^2 - N \approx O(N^2) \tag{1.1}$$

Bảng 1.1: So sánh hiệu năng theo số lượng hạt

Số hạt N	**Brute-force** $O(N^2)$	**Barnes-Hut** $O(N \log N)$	**Tăng tốc**
1,000	1,000,000	10,000	100x
10,000	100,000,000	133,000	750x
100,000	10,000,000,000	1,660,000	6,000x

⚠ Quan trọng

Với $N = 10,000$ và FPS mục tiêu $= 60$, Brute-force cần tính 6×10^9 phép tính/giây. Đây là **không khả thi** với CPU thông thường!

1.3 Giải pháp: Thuật toán Barnes-Hut

Năm 1986, Josh Barnes và Piet Hut công bố thuật toán giảm độ phức tạp xuống $O(N \log N)$ bằng cách sử dụng **cấu trúc dữ liệu cây** (Octree cho 3D).

Ý tưởng cốt lõi: Nếu một cụm sao *ở đủ xa*, ta có thể coi cả cụm đó như **một điểm khối lượng duy nhất** đặt tại trọng tâm.

1.4 Mục tiêu của Đồ án

1. **Mục tiêu kỹ thuật:**
   - Cài đặt Octree và thuật toán Barnes-Hut từ đầu bằng C++
   - Đạt hiệu năng real-time với $N \geq 10,000$ hạt
   - Tích hợp đồ họa 3D với Raylib
2. **Mục tiêu học tập:**
   - Nắm vững Con trỏ (Pointers) và Quản lý bộ nhớ động
   - Hiểu sâu Đệ quy (Recursion) qua Tree traversal
   - Phân tích độ phức tạp thuật toán (Big-O)
3. **Mục tiêu mở rộng (nếu còn thời gian):**
   - Tối ưu hóa với OpenMP (Parallel Computing)
   - Hiệu ứng đồ họa nâng cao (Bloom, Trails)

Phần II

Lộ trình Học tập Cấp tốc (Just-in-Time Learning)

Chương 2

Pointers & Memory Management

2.1 Tại sao Con trỏ quan trọng cho Octree?

Octree là cấu trúc dữ liệu **đệ quy** - mỗi node có thể chứa tối đa 8 node con. Để biểu diễn mối quan hệ này, ta **bắt buộc** phải dùng con trỏ.

⚠ Quan trọng

Nếu không hiểu con trỏ, bạn sẽ gặp:

- **Segmentation Fault:** Truy cập vùng nhớ không hợp lệ
- **Memory Leak:** Quên giải phóng bộ nhớ, chương trình ngốn RAM
- **Dangling Pointer:** Trỏ đến vùng nhớ đã bị xóa

2.2 Kiến thức cần nắm

2.2.1 Con trỏ cơ bản

```
// 1. Khai bao con tro
int x = 10;
int* ptr = &x;  // ptr LUU DIA CHI cua x

// 2. Truy cap gia tri qua con tro (Dereference)
int value = *ptr;  // value = 10

// 3. nullptr - con tro khong tro den dau ca
int* safePtr = nullptr;  // LUON khoi tao con tro = nullptr

// 4. Kiem tra truoc khi dung
if (ptr != nullptr) {
    *ptr = 20;  // An toan
}
```

Listing 2.1: Pointer Basics

2.2.2 Dynamic Memory (Heap vs Stack)

```cpp
// STACK: Tu dong giai phong khi ra khoi scope
void stackExample() {
    int arr[100];  // 100 int tren Stack
}  // arr tu dong mat khi ham ket thuc

// HEAP: Phai TU TAY giai phong
void heapExample() {
    // Cap phat 1 object
    int* single = new int(42);

    // Cap phat mang
    int* arr = new int[1000];

    // !!! PHAI GIAI PHONG !!!
    delete single;      // Xoa 1 object
    delete[] arr;       // Xoa mang (chu y [])
}
```

Listing 2.2: Dynamic Memory Allocation

Mẹo

Quy tắc vàng: Mỗi `new` phải có một `delete` tương ứng!
Trong Modern C++, ưu tiên dùng `std::unique_ptr` hoặc `std::shared_ptr` để tự động quản lý bộ nhớ.

2.2.3 Con trỏ trong Struct/Class

```cpp
// Day la ly do Octree can con tro
struct OctreeNode {
    // Moi node co the co 8 con
    // Neu dung object truc tiep -> VO HAN (infinite recursion)
    // OctreeNode children[8];  // LOI! Khong the dinh nghia

    // Phai dung CON TRO
    OctreeNode* children[8];  // OK! Chi luu DIA CHI

    OctreeNode() {
        // Khoi tao tat ca con tro = nullptr
        for (int i = 0; i < 8; i++) {
            children[i] = nullptr;
        }
    }

    ~OctreeNode() {
        // Destructor: Giai phong tat ca node con (DE QUY!)
        for (int i = 0; i < 8; i++) {
            if (children[i] != nullptr) {
                delete children[i];  // Goi destructor cua con
                children[i] = nullptr;
            }
        }
    }
};
```

Listing 2.3: Pointers in OctreeNode

2.3 Bài tập thực hành

1. Viết một Linked List đơn giản với các thao tác: `insert`, `remove`, `print`
2. Viết Binary Search Tree với: `insert`, `search`, `inorderTraversal`
3. Đảm bảo không có memory leak bằng cách chạy Valgrind (Linux) hoặc Visual Studio Memory Profiler

Chương 3

Recursion (Đệ quy)

3.1 Đệ quy là gì và tại sao cần cho Octree?

[title=</> Định nghĩa Đệ quy] Đệ quy là kỹ thuật trong đó một hàm **tự gọi chính nó** với input nhỏ hơn cho đến khi đạt **điều kiện dừng** (base case).

Octree dùng đệ quy ở đâu?

1. **Insert:** Khi chèn body vào cây, nếu node đã có body, ta chia node thành 8 con và chèn đệ quy.
2. **ComputeMass:** Tính tổng khối lượng từ lá lên gốc (bottom-up).
3. **CalculateForce:** Duyệt cây từ gốc, quyết định đi sâu hay xấp xỉ.
4. **Destructor:** Xóa cây từ gốc, đệ quy xóa tất cả con.

3.2 Cấu trúc của hàm đệ quy

```
void recursiveFunction(Input input) {
    // 1. BASE CASE - Dieu kien dung
    //    Khong co base case = STACK OVERFLOW!
    if (/* dieu kien dung */) {
        return;  // hoac return gia tri
    }

    // 2. RECURSIVE CASE - Xu ly va goi lai chinh minh
    //    Input PHAI nho hon de tien ve base case
    recursiveFunction(smallerInput);
}

// Vi du: Tinh giai thua
int factorial(int n) {
    // Base case
    if (n <= 1) return 1;

    // Recursive case: n! = n * (n-1)!
    return n * factorial(n - 1);
```

```
}
```

Listing 3.1: Anatomy of Recursion

3.3 Đệ quy duyệt cây (Tree Traversal)

```
struct TreeNode {
    int value;
    TreeNode* left;
    TreeNode* right;
};

// Pre-order: Xu ly node TRUOC khi xuong con
// Dung cho: In cay, copy cay
void preorder(TreeNode* node) {
    if (node == nullptr) return;  // Base case

    process(node);            // Xu ly truoc
    preorder(node->left);     // Xuong trai
    preorder(node->right);    // Xuong phai
}

// Post-order: Xu ly node SAU khi xuong con
// Dung cho: Tinh toan tu duoi len (Center of Mass!)
void postorder(TreeNode* node) {
    if (node == nullptr) return;  // Base case

    postorder(node->left);    // Xuong trai truoc
    postorder(node->right);   // Xuong phai
    process(node);            // Xu ly SAU CUNG
}

// In-order: Trai -> Node -> Phai (cho BST)
void inorder(TreeNode* node) {
    if (node == nullptr) return;
    inorder(node->left);
    process(node);
    inorder(node->right);
}
```

Listing 3.2: Tree Traversal Methods

⚠ Quan trọng

Octree dùng Post-order để tính Center of Mass!
Lý do: Phải biết khối lượng của tất cả node con **trước** khi tính khối lượng của node cha.

3.4 Ví dụ: Đệ quy trong Octree

```
void OctreeNode::computeMassDistribution() {
    // BASE CASE 1: Node rong
    if (isExternal && body == nullptr) {
```

```
        totalMass = 0.0;
        return;
    }

    // BASE CASE 2: La (External node) co body
    if (isExternal && body != nullptr) {
        totalMass = body->mass;
        centerOfMass = body->pos;
        return;
    }

    // RECURSIVE CASE: Node trong (Internal node)
    // Buoc 1: Tinh khoi luong cua tat ca con truoc
    for (int i = 0; i < 8; i++) {
        if (children[i] != nullptr) {
            children[i]->computeMassDistribution();  // DE QUY!
        }
    }

    // Buoc 2: Tong hop tu cac con
    totalMass = 0.0;
    centerOfMass = Vec3(0, 0, 0);

    for (int i = 0; i < 8; i++) {
        if (children[i] != nullptr && children[i]->totalMass > 0) {
            double childMass = children[i]->totalMass;
            Vec3 childCOM = children[i]->centerOfMass;

            // Cong thuc trong tam: COM = sum(m_i * r_i) / sum(m_i)
            centerOfMass = centerOfMass + childCOM * childMass;
            totalMass += childMass;
        }
    }

    if (totalMass > 0) {
        centerOfMass = centerOfMass / totalMass;
    }
}
```

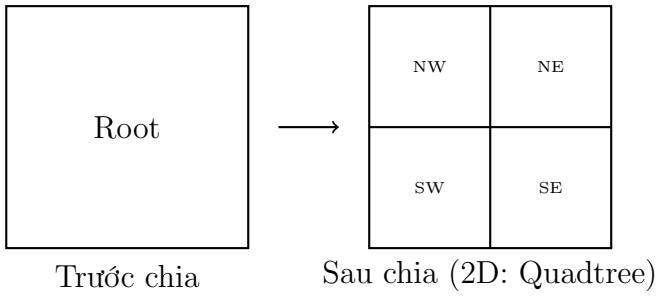
Listing 3.3: Recursive Mass Computation

Chương 4

Octree Data Structure

4.1 Khái niệm Octree

[title=🧊 Octree là gì?] Octree (Cây Bát phân) là cấu trúc dữ liệu cây dùng để **phân hoạch không gian 3D**.
Mỗi node đại diện cho một **hình lập phương** (cube) và có thể chia thành **8 hình lập phương con** (octants).



Hình 4.1: Minh họa phân chia không gian (2D cho đơn giản)

4.2 8 Octants trong 3D

Mỗi octant được xác định bởi vị trí tương đối với tâm của node cha:

Bảng 4.1: 8 Octants và điều kiện xác định

Index	Tên	X	Y	Z
0	Bottom-South-West	$<$ center	$<$ center	$<$ center
1	Bottom-South-East	$\geq$ center	$<$ center	$<$ center
2	Bottom-North-West	$<$ center	$\geq$ center	$<$ center
3	Bottom-North-East	$\geq$ center	$\geq$ center	$<$ center
4	Top-South-West	$<$ center	$<$ center	$\geq$ center
5	Top-South-East	$\geq$ center	$<$ center	$\geq$ center
6	Top-North-West	$<$ center	$\geq$ center	$\geq$ center
7	Top-North-East	$\geq$ center	$\geq$ center	$\geq$ center

```cpp
// Tinh chi so octant (0-7) cho mot diem
int getOctantIndex(const Vec3& point, const Vec3& nodeCenter) {
    int index = 0;

    // Bit 0: X >= center.x
    if (point.x >= nodeCenter.x) index |= 1;

    // Bit 1: Y >= center.y
    if (point.y >= nodeCenter.y) index |= 2;

    // Bit 2: Z >= center.z
    if (point.z >= nodeCenter.z) index |= 4;

    return index;  // Ket qua: 0 den 7
}
```

Listing 4.1: Calculate Octant Index

4.3 Hai loại Node trong Octree

1. **External Node (Lá):**
    - Không có node con
    - Chứa **tối đa 1 body** (hoặc rỗng)
    - Đây là điểm dừng của đệ quy
2. **Internal Node (Trong):**
    - Có ít nhất 1 node con
    - **Không chứa body trực tiếp**
    - Lưu trữ `totalMass` và `centerOfMass` (tổng hợp từ con)

Chương 5

Big-O Notation và Phân tích Độ phức tạp

5.1 Big-O là gì?

Big-O notation mô tả **tốc độ tăng trưởng** của thời gian chạy hoặc bộ nhớ khi input tăng.

Bảng 5.1: Các độ phức tạp phổ biến

Big-O	Tên	Ví dụ
$O(1)$	Constant	Truy cập array bằng index
$O(\log N)$	Logarithmic	Binary Search
$O(N)$	Linear	Duyệt mảng 1 lần
$O(N \log N)$	Linearithmic	**Barnes-Hut**, Merge Sort
$O(N^2)$	Quadratic	**Brute-force N-Body**, Bubble Sort
$O(2^N)$	Exponential	Brute-force Traveling Salesman

5.2 Tại sao Barnes-Hut là $O(N \log N)$?

Phân tích:

1. **Xây dựng Octree:** Chèn N body, mỗi lần chèn đi qua $O(\log N)$ mức $\Rightarrow O(N \log N)$
2. **Tính lực cho 1 body:** Duyệt cây, trung bình thăm $O(\log N)$ node (vì nhiều cụm được xấp xỉ)
3. **Tính lực cho N body:** $N \times O(\log N) = O(N \log N)$

Mẹo

Với $N = 10,000$:

- $O(N^2) = 100,000,000$ operations
- $O(N \log N) \approx 10,000 \times 13.3 = 133,000$ operations
- **Tăng tốc: 750 lần!**

5.3 So sánh trực quan

```
// BRUTE-FORCE: O(N^2)
// Moi body phai xet TAT CA body khac
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {  // N iterations
        if (i != j) {
            bodies[i].force += calculatePairForce(bodies[i], bodies[j]);
        }
    }
}
// Tong: N * N = N^2 lan tinh

// BARNES-HUT: O(N log N)
// Moi body chi xet ~log(N) "cum"
for (int i = 0; i < N; i++) {
    // calculateForce duyet cay, nhieu node duoc SKIP
    bodies[i].force = root->calculateForce(&bodies[i], theta);
}
// Trung binh: N * log(N) lan tinh
```

Listing 5.1: Brute-force vs Barnes-Hut

Phần III

Lộ trình 8 Tuần (Agile Sprints)

Chương 6

Sprint Planning Overview

Bảng 6.1: Tổng quan 8 Tuần

Tuần	**Focus**	**Sunday Milestone**
1	Setup & C++ Basics	Cửa sổ Raylib với hình lập phương quay
2	Brute-Force Physics	500 sao di chuyển theo hấp dẫn
3	Octree Structure	Khung dây Octree hiển thị trên màn hình
4	Octree Insert & Mass	Debug: In totalMass == sum of all bodies
5	Barnes-Hut Force	5,000 sao chạy real-time với BH
6	Optimization & Polish	10,000 sao + FPS stable $>$ 30
7	Visual Effects	Màu sao theo khối lượng + Camera smooth
8	Report & Demo	Video demo + Báo cáo hoàn chỉnh

Chương 7

Week 1: Environment Setup & Raylib Basics

7.1 Focus: Môi trường phát triển và Đồ họa 3D cơ bản

7.1.1 Tasks

1. **Cài đặt môi trường:**
   - Visual Studio 2022 hoặc VS Code + MinGW
   - Raylib 4.5+ (download từ raylib.com)
   - Git để quản lý version
2. **Học Raylib basics:**
   - `InitWindow()`, `CloseWindow()`
   - `BeginDrawing()`, `EndDrawing()`
   - `Camera3D` và điều khiển camera
3. **Code `Vec3` struct** với operator overloading
4. **Vẽ hình 3D cơ bản:** Cube, Sphere

7.1.2 Milestone: Spinning Cube

```
#include "raylib.h"

int main() {
    // Khoi tao cua so 1280x720
    InitWindow(1280, 720, "N-Body Simulation - Week 1");
    SetTargetFPS(60);

    // Thiet lap Camera 3D
    Camera3D camera = { 0 };
    camera.position = (Vector3){ 10.0f, 10.0f, 10.0f };
    camera.target = (Vector3){ 0.0f, 0.0f, 0.0f };
    camera.up = (Vector3){ 0.0f, 1.0f, 0.0f };
    camera.fovy = 45.0f;
    camera.projection = CAMERA_PERSPECTIVE;
```

```
15
16      float rotation = 0.0f;
17
18      while (!WindowShouldClose()) {
19          // Update
20          rotation += 1.0f;
21          UpdateCamera(&camera, CAMERA_ORBITAL);  // Cho phep xoay camera
    bang chuot
22
23          // Draw
24          BeginDrawing();
25              ClearBackground(BLACK);
26
27              BeginMode3D(camera);
28                  // Ve hinh lap phuong quay
29                  DrawCubeWires((Vector3){0, 0, 0}, 2.0f, 2.0f, 2.0f, GREEN)
    ;
30                  DrawGrid(10, 1.0f);
31              EndMode3D();
32
33              DrawText("Week 1: 3D Cube Spinning!", 10, 10, 20, WHITE);
34              DrawFPS(10, 40);
35          EndDrawing();
36      }
37
38      CloseWindow();
39      return 0;
40  }
```

Listing 7.1: Week 1 Milestone - main.cpp

Kết quả mong đợi: Một cửa sổ đen với lưới và hình lập phương wireframe. Có thể xoay camera bằng chuột.

Chương 8

Week 2: Brute-Force N-Body Implementation

8.1 Focus: Vật lý cơ bản và Brute-Force Algorithm

8.1.1 Tasks

1. **Implement `Body` struct:**
   - Position, Velocity, Acceleration, Mass
   - Hàm `update(dt)` để cập nhật vị trí
2. **Implement Newton's Gravity:**
   - Công thức lực hấp dẫn với Softening
   - Tính tổng lực từ tất cả body khác
3. **Galaxy Initialization:**
   - Tạo N body ngẫu nhiên theo hình cầu hoặc đĩa
   - Vận tốc ban đầu để tạo chuyển động xoay
4. **Render bodies** như các điểm sáng

8.1.2 Code: Body Struct

```
#pragma once
#include "Vec3.hpp"
#include "raylib.h"

struct Body {
    Vec3 pos;       // Vi tri
    Vec3 vel;       // Van toc
    Vec3 acc;       // Gia toc (tinh moi frame)
    double mass;    // Khoi luong
    Color color;    // Mau hien thi

    // Constructor
    Body(Vec3 p = Vec3(), Vec3 v = Vec3(), double m = 1.0)
```

```
        : pos(p), vel(v), acc(Vec3()), mass(m), color(WHITE) {}

    // Cap nhat vi tri bang Euler Integration
    void update(double dt) {
        vel = vel + acc * dt;   // v = v + a*dt
        pos = pos + vel * dt;   // x = x + v*dt
        acc = Vec3();           // Reset gia toc cho frame sau
    }

    // Ve body len man hinh
    void draw() const {
        // Chuyen Vec3 sang Vector3 cua Raylib
        Vector3 rlPos = { (float)pos.x, (float)pos.y, (float)pos.z };
        DrawSphere(rlPos, 0.05f, color);  // Ve hinh cau nho
    }
};
```

Listing 8.1: Body.hpp

8.1.3 Code: Brute-Force Force Calculation

```
#include <vector>
#include "Body.hpp"

// Hang so vat ly
const double G = 6.67430e-11;  // Hang so hap dan (SI units)
const double SOFTENING = 0.1;  // Tranh chia cho 0

// Tinh luc hap dan giua 2 body
Vec3 calculatePairForce(const Body& a, const Body& b) {
    Vec3 r = b.pos - a.pos;  // Vector tu a den b
    double distSq = r.x*r.x + r.y*r.y + r.z*r.z;

    // Them softening de tranh vo cuc khi dist -> 0
    double dist = sqrt(distSq + SOFTENING * SOFTENING);

    // F = G * m1 * m2 / r^2
    // a = F / m1 = G * m2 / r^2
    // Huong: r / |r| (normalized)
    double forceMagnitude = G * b.mass / (dist * dist * dist);

    return r * forceMagnitude;  // Vec3 acceleration
}

// BRUTE-FORCE: O(N^2)
void updateAllBodies_BruteForce(std::vector<Body>& bodies, double dt) {
    int N = bodies.size();

    // Tinh gia toc cho moi body
    for (int i = 0; i < N; i++) {
        Vec3 totalAcc;
        for (int j = 0; j < N; j++) {
            if (i != j) {
                totalAcc = totalAcc + calculatePairForce(bodies[i], bodies
    [j]);
            }
        }
```

```
36          bodies[i].acc = totalAcc;
37      }
38
39      // Cap nhat vi tri
40      for (int i = 0; i < N; i++) {
41          bodies[i].update(dt);
42      }
43  }
```

Listing 8.2: Physics.cpp

8.1.4 Milestone: 500 Stars Moving

Kết quả mong đợi: 500 điểm sáng di chuyển và bắt đầu tụ lại với nhau do lực hấp dẫn. FPS có thể thấp (20-30) - đây là bình thường cho Brute-force!

Chương 9

Week 3: Octree Structure & Visualization

9.1 Focus: Xây dựng cấu trúc Octree cơ bản

9.1.1 Tasks

1. **Implement `OctreeNode` class:**
   - Constructor với center và size
   - Mảng 8 con trỏ children
   - Destructor đệ quy
2. **Implement `getOctantIndex()`:** Xác định body thuộc octant nào
3. **Implement `subdivide()`:** Chia node thành 8 con
4. **Visual Debug:** Vẽ wireframe của tất cả node

9.1.2 Code: OctreeNode Class

```cpp
#pragma once
#include "Vec3.hpp"
#include "Body.hpp"
#include <array>

class OctreeNode {
public:
    // ==================== MEMBER VARIABLES ====================

    Vec3 center;           // Tam cua hinh lap phuong
    double halfSize;       // Nua canh (de tinh bounds)

    // === Du lieu vat ly (dung cho Barnes-Hut) ===
    double totalMass;      // Tong khoi luong cua node va cac con
    Vec3 centerOfMass;     // Trong tam khoi luong

    // === Cau truc cay ===
    Body* body;            // Chi dung khi la External Node (la)
    std::array<OctreeNode*, 8> children;  // 8 con tro den node con
```

```
20
21      bool isExternal;        // true = la (khong co con)
22
23      // ==================== CONSTRUCTOR/DESTRUCTOR ====================
24
25      OctreeNode(Vec3 c, double size)
26          : center(c), halfSize(size / 2.0),
27            totalMass(0.0), centerOfMass(Vec3()),
28            body(nullptr), isExternal(true)
29      {
30          // Khoi tao tat ca con tro children = nullptr
31          for (int i = 0; i < 8; i++) {
32              children[i] = nullptr;
33          }
34      }
35
36      // Destructor DE QUY - xoa tat ca node con
37      ~OctreeNode() {
38          for (int i = 0; i < 8; i++) {
39              if (children[i] != nullptr) {
40                  delete children[i];  // Goi destructor cua con
41                  children[i] = nullptr;
42              }
43          }
44      }
45
46      // ==================== METHODS ====================
47
48      // Xac dinh octant index (0-7) cho mot diem
49      int getOctantIndex(const Vec3& point) const {
50          int index = 0;
51          if (point.x >= center.x) index |= 1;  // Bit 0
52          if (point.y >= center.y) index |= 2;  // Bit 1
53          if (point.z >= center.z) index |= 4;  // Bit 2
54          return index;
55      }
56
57      // Tinh tam cua octant thu i
58      Vec3 getOctantCenter(int i) const {
59          double offset = halfSize / 2.0;
60          return Vec3(
61              center.x + ((i & 1) ? offset : -offset),
62              center.y + ((i & 2) ? offset : -offset),
63              center.z + ((i & 4) ? offset : -offset)
64          );
65      }
66
67      // Chia node thanh 8 con
68      void subdivide() {
69          double childSize = halfSize;  // Moi con co kich thuoc = nua cha
70          for (int i = 0; i < 8; i++) {
71              Vec3 childCenter = getOctantCenter(i);
72              children[i] = new OctreeNode(childCenter, childSize);
73          }
74          isExternal = false;  // Khong con la la nua
75      }
76
77      // INSERT - Se implement o Week 4
```

```
78      void insert(Body* newBody);
79
80      // COMPUTE MASS - Se implement o Week 4
81      void computeMassDistribution();
82
83      // CALCULATE FORCE - Se implement o Week 5
84      Vec3 calculateForce(const Body* target, double theta, double G, double
        softening);
85
86      // Ve wireframe de debug
87      void drawWireframe(Color color) const {
88          float s = (float)halfSize;
89          Vector3 c = { (float)center.x, (float)center.y, (float)center.z };
90          DrawCubeWires(c, s*2, s*2, s*2, color);
91
92          // Ve de quy cac con
93          if (!isExternal) {
94              for (int i = 0; i < 8; i++) {
95                  if (children[i] != nullptr) {
96                      children[i]->drawWireframe(color);
97                  }
98              }
99          }
100     }
101 };
```

Listing 9.1: OctreeNode.hpp

9.1.3 Milestone: Octree Wireframe Visible

Kết quả mong đợi: Các hình lập phương wireframe hiển thị, thể hiện cấu trúc phân cấp của Octree. Node có body sẽ được chia nhỏ hơn.

Chương 10

Week 4: Octree Insert & Mass Computation

10.1 Focus: Hoàn thiện Octree với Insert và ComputeMass

10.1.1 Tasks

1. **Implement `insert(Body*)`:**
   - Xử lý 3 trường hợp: Node rỗng, Node có body, Node có con
   - Logic đệ quy chia nhỏ khi cần
2. **Implement `computeMassDistribution()`:**
   - Duyệt Post-order (con trước, cha sau)
   - Tính totalMass và centerOfMass
3. **Debug & Verify:**
   - In ra `root->totalMass`
   - Phải bằng tổng khối lượng tất cả body!

10.1.2 Code: Insert Algorithm (Core of Octree)

```
void OctreeNode::insert(Body* newBody) {
    // ========== CASE 1: External node va rong ==========
    // Chi can gan body vao
    if (isExternal && body == nullptr) {
        body = newBody;
        return;
    }

    // ========== CASE 2: External node DA CO body ==========
    // Phai chia thanh 8 con, roi chen ca 2 body vao
    if (isExternal && body != nullptr) {
        // Buoc 1: Chia node hien tai
        subdivide();

        // Buoc 2: Chen body CU vao octant tuong ung
```

```
        Body* oldBody = body;
        body = nullptr;  // Node nay khong con la la nua

        int oldIndex = getOctantIndex(oldBody->pos);
        children[oldIndex]->insert(oldBody);  // DE QUY!

        // Buoc 3: Chen body MOI vao octant tuong ung
        int newIndex = getOctantIndex(newBody->pos);
        children[newIndex]->insert(newBody);  // DE QUY!

        return;
    }

    // ========== CASE 3: Internal node (da co con) ==========
    // Chi can tim octant dung va chen de quy
    int index = getOctantIndex(newBody->pos);

    // Tao node con neu chua ton tai
    if (children[index] == nullptr) {
        Vec3 childCenter = getOctantCenter(index);
        children[index] = new OctreeNode(childCenter, halfSize);
    }

    children[index]->insert(newBody);  // DE QUY!
}
```

Listing 10.1: OctreeNode::insert()

10.1.3 Code: Compute Mass Distribution

```
void OctreeNode::computeMassDistribution() {
    // ========== BASE CASE 1: Node rong ==========
    if (isExternal && body == nullptr) {
        totalMass = 0.0;
        centerOfMass = Vec3();
        return;
    }

    // ========== BASE CASE 2: La co body ==========
    if (isExternal && body != nullptr) {
        totalMass = body->mass;
        centerOfMass = body->pos;
        return;
    }

    // ========== RECURSIVE CASE: Node trong ==========
    // POST-ORDER: Tinh con truoc, cha sau

    // Buoc 1: Goi de quy cho tat ca con
    for (int i = 0; i < 8; i++) {
        if (children[i] != nullptr) {
            children[i]->computeMassDistribution();
        }
    }

    // Buoc 2: Tong hop tu cac con
    totalMass = 0.0;
```

```
    Vec3 weightedSum = Vec3();

    for (int i = 0; i < 8; i++) {
        if (children[i] != nullptr && children[i]->totalMass > 0) {
            double m = children[i]->totalMass;
            Vec3 com = children[i]->centerOfMass;

            weightedSum = weightedSum + com * m;
            totalMass += m;
        }
    }

    // Buoc 3: Tinh trong tam
    if (totalMass > 0) {
        centerOfMass = weightedSum / totalMass;
    }
}
```

Listing 10.2: OctreeNode::computeMassDistribution()

10.1.4 Milestone: Debug Verification

```
// Trong main loop, sau khi build tree:
double totalBodyMass = 0.0;
for (const auto& b : bodies) {
    totalBodyMass += b.mass;
}

root->computeMassDistribution();

std::cout << "Sum of body masses: " << totalBodyMass << std::endl;
std::cout << "Root totalMass: " << root->totalMass << std::endl;
std::cout << "MATCH: " << (abs(totalBodyMass - root->totalMass) < 0.001 ?
   "YES" : "NO") << std::endl;
```

Listing 10.3: Verification Code

Kết quả mong đợi: Console in ra "MATCH: YES". Nếu không, có bug trong insert hoặc computeMass!

Chương 11

Week 5: Barnes-Hut Force Calculation

11.1 Focus: Thuật toán Barnes-Hut hoàn chỉnh

11.1.1 Tasks

1. **Implement `calculateForce()`:**
   - Kiểm tra Multipole Acceptance Criterion ($s/d < \theta$)
   - Nếu đủ xa: dùng centerOfMass
   - Nếu quá gần: đệ quy vào con
2. **Thay thế Brute-force bằng Barnes-Hut** trong main loop
3. **Benchmark:** So sánh FPS giữa 2 phương pháp

11.1.2 Code: Barnes-Hut Force Calculation

```
Vec3 OctreeNode::calculateForce(const Body* target, double theta,
                                double G, double softening) {
    // ========== CASE 1: Node rong ==========
    if (totalMass == 0.0) {
        return Vec3();  // Khong co luc
    }

    // ========== CASE 2: La voi 1 body ==========
    if (isExternal && body != nullptr) {
        // Khong tinh luc len chinh no
        if (body == target) {
            return Vec3();
        }

        // Tinh luc truc tiep
        Vec3 r = body->pos - target->pos;
        double distSq = r.x*r.x + r.y*r.y + r.z*r.z;
        double dist = sqrt(distSq + softening * softening);

        // a = G * m / r^2 * direction
        double magnitude = G * body->mass / (dist * dist * dist);
        return r * magnitude;
    }

```

```
    // ========== CASE 3: Node trong - BARNES-HUT LOGIC ==========
    Vec3 r = centerOfMass - target->pos;
    double distSq = r.x*r.x + r.y*r.y + r.z*r.z;
    double dist = sqrt(distSq);

    // Kich thuoc node
    double s = halfSize * 2.0;

    // *** MULTIPOLE ACCEPTANCE CRITERION ***
    // Neu s/d < theta, cum nay "du xa" de xap xi
    if (dist > 0 && (s / dist) < theta) {
        // Dung centerOfMass va totalMass de tinh
        double distSoftened = sqrt(distSq + softening * softening);
        double magnitude = G * totalMass / (distSoftened * distSoftened *
   distSoftened);
        return r * magnitude;
    }

    // Neu khong du xa, phai di sau vao cac con
    Vec3 totalForce;
    for (int i = 0; i < 8; i++) {
        if (children[i] != nullptr) {
            totalForce = totalForce + children[i]->calculateForce(
                target, theta, G, softening
            );  // DE QUY!
        }
    }

    return totalForce;
}
```

Listing 11.1: OctreeNode::calculateForce()

11.1.3 Code: Main Loop with Barnes-Hut

```
// Constants
const double THETA = 0.5;       // Accuracy parameter
const double G = 1.0;           // Simplified gravity constant
const double SOFTENING = 0.1;
const double DT = 0.01;         // Time step
const double UNIVERSE_SIZE = 100.0;

void simulationStep(std::vector<Body>& bodies) {
    // ====== STEP 1: Rebuild Octree ======
    // (Tree phai rebuild moi frame vi bodies di chuyen)

    Vec3 center(0, 0, 0);
    OctreeNode* root = new OctreeNode(center, UNIVERSE_SIZE);

    for (auto& body : bodies) {
        root->insert(&body);
    }

    // ====== STEP 2: Compute Mass Distribution ======
    root->computeMassDistribution();

    // ====== STEP 3: Calculate Forces using Barnes-Hut ======
```

```
23      for (auto& body : bodies) {
24          body.acc = root->calculateForce(&body, THETA, G, SOFTENING);
25      }
26
27      // ====== STEP 4: Update Positions ======
28      for (auto& body : bodies) {
29          body.update(DT);
30      }
31
32      // ====== STEP 5: Clean up ======
33      delete root;  // Destructor se xoa toan bo cay
34  }
```

Listing 11.2: Main Loop

11.1.4 Milestone: 5,000 Stars Real-time

Kết quả mong đợi: FPS tăng đáng kể! Với $N = 5,000$:

- Brute-force: ~5-10 FPS
- Barnes-Hut: ~30-60 FPS

Chương 12

Week 6: Optimization & Boundary Handling

12.1 Focus: Tối ưu hóa và xử lý biên

12.1.1 Tasks

1. **OpenMP Parallelization:**
   - Thêm `#pragma omp parallel for` vào vòng tính lực
   - Compile với flag `-fopenmp`
2. **Boundary Handling:**
   - Xử lý body bay ra ngoài universe bounds
   - Options: Bỏ qua, teleport, hoặc expand bounds
3. **Memory Optimization:**
   - Reuse memory thay vì new/delete mỗi frame
   - Object pooling cho OctreeNode

12.1.2 Code: OpenMP Parallelization

```
#include <omp.h>

void simulationStep(std::vector<Body>& bodies) {
    // ... Build tree ...

    // *** PARALLEL FORCE CALCULATION ***
    // Chi can them 1 dong nay!
    #pragma omp parallel for
    for (size_t i = 0; i < bodies.size(); i++) {
        bodies[i].acc = root->calculateForce(&bodies[i], THETA, G,
    SOFTENING);
    }

    // Update cung co the parallel
    #pragma omp parallel for
```

```
15      for (size_t i = 0; i < bodies.size(); i++) {
16          bodies[i].update(DT);
17      }
18
19      delete root;
20  }
```

Listing 12.1: Parallel Force Calculation

12.1.3 Milestone: 10,000 Stars Stable $>$ 30 FPS

Compile command:

```
g++ -O3 -fopenmp main.cpp -o simulation -lraylib
```

Chương 13

Week 7: Visual Effects & Polish

13.1 Focus: Đồ họa đẹp và UX

13.1.1 Tasks

1. **Mass-based Coloring:** Sao nhỏ = đỏ, sao lớn = xanh
2. **Smooth Camera:**
   - Orbital camera với zoom
   - Follow camera theo center of mass
3. **UI Overlay:**
   - Hiển thị FPS, số hạt, theta
   - Controls help text
4. **Optional - Bloom Effect:** Post-processing cho sao phát sáng

13.1.2 Code: Mass to Color Mapping

```
// Anh xa khoi luong sang mau sac (Quang pho sao)
// Do lun (Red Dwarf) -> Vang (Sun) -> Xanh khong lo (Blue Giant)
Color massToColor(double mass, double minMass, double maxMass) {
    // Normalize mass to [0, 1]
    double t = (mass - minMass) / (maxMass - minMass);
    t = fmax(0.0, fmin(1.0, t));  // Clamp

    unsigned char r, g, b;

    if (t < 0.25) {
        // Do dam -> Do nhat (Red Dwarf)
        double local_t = t / 0.25;
        r = 139 + (unsigned char)(116 * local_t);  // 139 -> 255
        g = 0 + (unsigned char)(69 * local_t);     // 0 -> 69
        b = 0;
    }
    else if (t < 0.5) {
        // Do nhat -> Vang (Orange to Yellow)
        double local_t = (t - 0.25) / 0.25;
        r = 255;
```

```
        g = 69 + (unsigned char)(186 * local_t);   // 69 -> 255
        b = 0 + (unsigned char)(100 * local_t);    // 0 -> 100
    }
    else if (t < 0.75) {
        // Vang -> Trang (Yellow to White)
        double local_t = (t - 0.5) / 0.25;
        r = 255;
        g = 255;
        b = 100 + (unsigned char)(155 * local_t);  // 100 -> 255
    }
    else {
        // Trang -> Xanh (White to Blue Giant)
        double local_t = (t - 0.75) / 0.25;
        r = 255 - (unsigned char)(100 * local_t);  // 255 -> 155
        g = 255 - (unsigned char)(100 * local_t);  // 255 -> 155
        b = 255;
    }

    return (Color){ r, g, b, 255 };
}
```

Listing 13.1: Star Color based on Mass

13.1.3 Milestone: Beautiful Galaxy Visualization

Kết quả: Một thiên hà với sao nhiều màu sắc, camera mượt mà, và UI thông tin.

Chương 14

Week 8: Documentation & Demo

14.1 Focus: Hoàn thiện báo cáo và Demo

14.1.1 Tasks

1. **Báo cáo kỹ thuật:**
   - Giải thích thuật toán với hình minh họa
   - Biểu đồ so sánh performance
   - Source code có comment đầy đủ
2. **Video Demo:**
   - 3-5 phút showcase
   - Thể hiện: Galaxy formation, real-time performance, code walkthrough
3. **Presentation Slides:**
   - 15 slides max
   - Focus: Problem → Solution → Demo → Lessons Learned
4. **Code Cleanup:**
   - Xóa debug code
   - Thêm README.md với build instructions

14.1.2 Deliverables

Item	**Format**
Source Code	GitHub Repository
Báo cáo	PDF (20-30 trang)
Video Demo	MP4 (3-5 phút)
Slides	PDF/PPTX (15 slides)

Phần IV

Kiến trúc Hệ thống & Code Hoàn chỉnh

Chương 15

Project Structure

15.1 Directory Layout

```
n-body-simulation/
|-- include/
|   |-- Vec3.hpp          # 3D Vector math
|   |-- Body.hpp          # Celestial body struct
|   |-- OctreeNode.hpp    # Octree data structure
|   |-- Physics.hpp       # Force calculations
|   |-- Galaxy.hpp        # Galaxy initialization
|   |-- Renderer.hpp      # Raylib rendering helpers
|
|-- src/
|   |-- main.cpp          # Entry point & main loop
|   |-- Vec3.cpp
|   |-- Body.cpp
|   |-- OctreeNode.cpp
|   |-- Physics.cpp
|   |-- Galaxy.cpp
|   |-- Renderer.cpp
|
|-- assets/               # (Optional) textures, fonts
|-- build/                # Compiled output
|-- CMakeLists.txt        # Build configuration
|-- README.md
```

Chương 16

Complete Code Implementation

16.1 Vec3.hpp - 3D Vector Mathematics

```cpp
#pragma once
#include <cmath>

/**
 * @brief 3D Vector class for physics calculations
 *
 * Tai sao dung struct thay vi class?
 * -> Vec3 la "Plain Old Data" (POD), khong can encapsulation
 * -> Performance tot hon khi copy
 */
struct Vec3 {
    double x, y, z;

    // ==================== CONSTRUCTORS ====================

    // Default constructor - khoi tao (0, 0, 0)
    Vec3() : x(0.0), y(0.0), z(0.0) {}

    // Parameterized constructor
    Vec3(double x_, double y_, double z_) : x(x_), y(y_), z(z_) {}

    // ==================== OPERATOR OVERLOADING ====================
    // Cho phep viet code toan hoc tu nhien: v1 + v2, v * 2.0, etc.

    // Vector + Vector
    Vec3 operator+(const Vec3& other) const {
        return Vec3(x + other.x, y + other.y, z + other.z);
    }

    // Vector - Vector
    Vec3 operator-(const Vec3& other) const {
        return Vec3(x - other.x, y - other.y, z - other.z);
    }

    // Vector * Scalar
    Vec3 operator*(double scalar) const {
        return Vec3(x * scalar, y * scalar, z * scalar);
    }

    // Vector / Scalar
```

```
41      Vec3 operator/(double scalar) const {
42          // Chu y: Khong check divide by zero o day de giu performance
43          return Vec3(x / scalar, y / scalar, z / scalar);
44      }
45
46      // Compound assignment operators
47      Vec3& operator+=(const Vec3& other) {
48          x += other.x; y += other.y; z += other.z;
49          return *this;
50      }
51
52      Vec3& operator-=(const Vec3& other) {
53          x -= other.x; y -= other.y; z -= other.z;
54          return *this;
55      }
56
57      Vec3& operator*=(double scalar) {
58          x *= scalar; y *= scalar; z *= scalar;
59          return *this;
60      }
61
62      // ==================== UTILITY METHODS ====================
63
64      // Do dai binh phuong (tranh sqrt khi khong can)
65      double magnitudeSq() const {
66          return x*x + y*y + z*z;
67      }
68
69      // Do dai vector
70      double magnitude() const {
71          return std::sqrt(magnitudeSq());
72      }
73
74      // Khoang cach binh phuong den vector khac
75      double distanceSq(const Vec3& other) const {
76          double dx = x - other.x;
77          double dy = y - other.y;
78          double dz = z - other.z;
79          return dx*dx + dy*dy + dz*dz;
80      }
81
82      // Khoang cach den vector khac
83      double distance(const Vec3& other) const {
84          return std::sqrt(distanceSq(other));
85      }
86
87      // Normalize (do dai = 1)
88      Vec3 normalized() const {
89          double mag = magnitude();
90          if (mag > 0) {
91              return *this / mag;
92          }
93          return Vec3();  // Tra ve (0,0,0) neu vector = 0
94      }
95
96      // Dot product (tich vo huong)
97      double dot(const Vec3& other) const {
98          return x*other.x + y*other.y + z*other.z;
```

```
    }

    // Cross product (tich co huong)
    Vec3 cross(const Vec3& other) const {
        return Vec3(
            y*other.z - z*other.y,
            z*other.x - x*other.z,
            x*other.y - y*other.x
        );
    }
};

// Scalar * Vector (cho phep viet: 2.0 * vec)
inline Vec3 operator*(double scalar, const Vec3& vec) {
    return vec * scalar;
}
```

Listing 16.1: include/Vec3.hpp

16.2 Body.hpp - Celestial Body Structure

```
#pragma once
#include "Vec3.hpp"
#include "raylib.h"

/**
 * @brief Represents a celestial body (star, planet, particle)
 *
 * Moi Body co:
 * - Position (vi tri trong khong gian 3D)
 * - Velocity (van toc - huong va toc do di chuyen)
 * - Acceleration (gia toc - duoc tinh moi frame tu luc hap dan)
 * - Mass (khoi luong - quyet dinh luc hap dan)
 * - Color (mau sac hien thi)
 */
struct Body {
    // ==================== PHYSICS PROPERTIES ====================
    Vec3 pos;       // Position (meters in simulation space)
    Vec3 vel;       // Velocity (meters/second)
    Vec3 acc;       // Acceleration (meters/second^2) - reset moi frame
    double mass;    // Mass (kilograms)

    // ==================== VISUAL PROPERTIES ====================
    Color color;    // Raylib Color for rendering
    float radius;   // Visual radius for drawing

    // ==================== CONSTRUCTORS ====================

    Body()
        : pos(Vec3()), vel(Vec3()), acc(Vec3()),
          mass(1.0), color(WHITE), radius(0.1f) {}

    Body(Vec3 position, Vec3 velocity, double m)
        : pos(position), vel(velocity), acc(Vec3()),
          mass(m), color(WHITE), radius(0.1f) {}

```

```
36      // ==================== PHYSICS METHODS ====================
37
38      /**
39       * @brief Update position using Semi-implicit Euler integration
40       * @param dt Time step (seconds)
41       *
42       * Semi-implicit Euler (aka Symplectic Euler):
43       * 1. v_new = v + a * dt
44       * 2. x_new = x + v_new * dt  (dung v MOI, khong phai v cu)
45       *
46       * Tai sao khong dung Euler thuong?
47       * -> Euler thuong: x_new = x + v * dt (dung v cu)
48       * -> Gay ra "energy drift" - nang luong tang theo thoi gian
49       * -> Semi-implicit Euler bao toan nang luong tot hon
50       */
51      void update(double dt) {
52          // Step 1: Update velocity with current acceleration
53          vel = vel + acc * dt;
54
55          // Step 2: Update position with NEW velocity
56          pos = pos + vel * dt;
57
58          // Step 3: Reset acceleration for next frame
59          // (Se duoc tinh lai trong calculateForce)
60          acc = Vec3();
61      }
62
63      /**
64       * @brief Verlet Integration (more stable for long simulations)
65       * @param dt Time step
66       *
67       * Velocity Verlet algorithm:
68       * x_new = x + v*dt + 0.5*a*dt^2
69       * v_new = v + 0.5*(a_old + a_new)*dt
70       *
71       * Note: Can tinh a_new truoc khi update velocity
72       * Phuc tap hon nhung chinh xac hon Euler
73       */
74      void updateVerlet(double dt, const Vec3& newAcc) {
75          // Update position
76          pos = pos + vel * dt + acc * (0.5 * dt * dt);
77
78          // Update velocity using average of old and new acceleration
79          vel = vel + (acc + newAcc) * (0.5 * dt);
80
81          // Store new acceleration
82          acc = newAcc;
83      }
84
85      // ==================== RENDERING METHODS ====================
86
87      /**
88       * @brief Draw the body as a sphere in 3D space
89       */
90      void draw() const {
91          // Convert Vec3 to Raylib's Vector3
92          Vector3 rlPos = {
93              static_cast<float>(pos.x),
```

```
            static_cast<float>(pos.y),
            static_cast<float>(pos.z)
        };

        // Draw solid sphere
        DrawSphere(rlPos, radius, color);
    }

    /**
     * @brief Draw as a point (faster for many bodies)
     */
    void drawPoint() const {
        Vector3 rlPos = {
            static_cast<float>(pos.x),
            static_cast<float>(pos.y),
            static_cast<float>(pos.z)
        };
        DrawPoint3D(rlPos, color);
    }
};
```

Listing 16.2: include/Body.hpp

16.3 OctreeNode.hpp - Complete Octree Implementation

```
#pragma once
#include "Vec3.hpp"
#include "Body.hpp"
#include <array>
#include <cmath>

/**
 * @brief Octree Node for Barnes-Hut algorithm
 *
 * Octree la cau truc cay phan hoach khong gian 3D.
 * Moi node dai dien cho mot hinh lap phuong (cube) trong khong gian.
 *
 * Co 2 loai node:
 * 1. External (Leaf): Khong co con, chua toi da 1 body
 * 2. Internal: Co it nhat 1 con, luu totalMass va centerOfMass
 */
class OctreeNode {
public:
    // ==================== SPATIAL PROPERTIES ====================
    Vec3 center;       // Tam cua hinh lap phuong
    double halfSize;   // Nua canh (bounds: center +/- halfSize)

    // ==================== PHYSICS PROPERTIES (for Barnes-Hut) ====================
    double totalMass;     // Tong khoi luong cua node va cac con
    Vec3 centerOfMass;    // Trong tam khoi luong (weighted average)

    // ==================== TREE STRUCTURE ====================
    Body* body;                           // Con tro den body (chi khi la External)
    std::array<OctreeNode*, 8> children;  // 8 con tro den node con
```

```
30      bool isExternal;                    // true = leaf node
31
32      // ==================== CONSTRUCTOR ====================
33
34      /**
35       * @brief Construct a new Octree Node
36       * @param c Center position of the cube
37       * @param size Full size of the cube edge
38       */
39      OctreeNode(Vec3 c, double size)
40          : center(c),
41            halfSize(size / 2.0),
42            totalMass(0.0),
43            centerOfMass(Vec3()),
44            body(nullptr),
45            isExternal(true)
46      {
47          // Khoi tao tat ca children = nullptr
48          // Su dung std::array::fill() cho clean code
49          children.fill(nullptr);
50      }
51
52      // ==================== DESTRUCTOR ====================
53
54      /**
55       * @brief Recursive destructor - delete entire subtree
56       *
57       * QUAN TRONG: Day la vi du cua DE QUY trong destructor.
58       * Khi delete root, no se tu dong delete tat ca node con.
59       */
60      ~OctreeNode() {
61          for (int i = 0; i < 8; i++) {
62              if (children[i] != nullptr) {
63                  delete children[i];  // Goi destructor cua con (DE QUY!)
64                  children[i] = nullptr;
65              }
66          }
67      }
68
69      // ==================== HELPER METHODS ====================
70
71      /**
72       * @brief Determine which octant a point belongs to
73       * @param point The 3D position to check
74       * @return Index 0-7 representing the octant
75       *
76       * Octant indexing using bit flags:
77       * Bit 0 (value 1): X >= center.x
78       * Bit 1 (value 2): Y >= center.y
79       * Bit 2 (value 4): Z >= center.z
80       *
81       * Example: point at (5, -2, 3) with center (0, 0, 0)
82       * - X >= 0: true  -> bit 0 = 1
83       * - Y >= 0: false -> bit 1 = 0
84       * - Z >= 0: true  -> bit 2 = 1
85       * - Result: 0b101 = 5
86       */
87      int getOctantIndex(const Vec3& point) const {
```

```
        int index = 0;
        if (point.x >= center.x) index |= 1;
        if (point.y >= center.y) index |= 2;
        if (point.z >= center.z) index |= 4;
        return index;
    }

    /**
     * @brief Calculate the center position of a child octant
     * @param i Octant index (0-7)
     * @return Center position of the child cube
     */
    Vec3 getOctantCenter(int i) const {
        double offset = halfSize / 2.0;
        return Vec3(
            center.x + ((i & 1) ? offset : -offset),
            center.y + ((i & 2) ? offset : -offset),
            center.z + ((i & 4) ? offset : -offset)
        );
    }

    /**
     * @brief Check if a point is within this node's bounds
     */
    bool contains(const Vec3& point) const {
        return (point.x >= center.x - halfSize && point.x < center.x +
    halfSize) &&
               (point.y >= center.y - halfSize && point.y < center.y +
    halfSize) &&
               (point.z >= center.z - halfSize && point.z < center.z +
    halfSize);
    }

    // ==================== CORE ALGORITHMS ====================

    /**
     * @brief Subdivide this node into 8 children
     *
     * Duoc goi khi can chen body thu 2 vao External node.
     */
    void subdivide() {
        double childSize = halfSize;  // Moi con co kich thuoc = nua cha

        for (int i = 0; i < 8; i++) {
            Vec3 childCenter = getOctantCenter(i);
            children[i] = new OctreeNode(childCenter, childSize);
        }

        isExternal = false;  // Node nay tro thanh Internal
    }

    /**
     * @brief Insert a body into the octree
     * @param newBody Pointer to the body to insert
     *
     * THUAT TOAN INSERT (De quy):
     *
     * Case 1: External node, rong
```

```
     *   -> Gan body vao, xong
     *
     * Case 2: External node, da co body
     *   -> Chia thanh 8 con (subdivide)
     *   -> Chen body cu vao octant tuong ung
     *   -> Chen body moi vao octant tuong ung
     *
     * Case 3: Internal node (da co con)
     *   -> Tim octant chua body moi
     *   -> Tao node con neu chua ton tai
     *   -> Goi insert de quy vao node con
     */
    void insert(Body* newBody) {
        // Kiem tra body co nam trong bounds khong
        if (!contains(newBody->pos)) {
            // Body ngoai bounds - co the skip hoac expand tree
            return;
        }

        // CASE 1: External node va rong
        if (isExternal && body == nullptr) {
            body = newBody;
            return;
        }

        // CASE 2: External node DA CO body
        if (isExternal && body != nullptr) {
            // Luu body cu
            Body* existingBody = body;
            body = nullptr;

            // Chia node
            subdivide();

            // Chen lai body cu
            int existingIndex = getOctantIndex(existingBody->pos);
            children[existingIndex]->insert(existingBody);  // DE QUY

            // Chen body moi
            int newIndex = getOctantIndex(newBody->pos);
            children[newIndex]->insert(newBody);  // DE QUY

            return;
        }

        // CASE 3: Internal node
        int index = getOctantIndex(newBody->pos);

        // Tao node con neu chua co
        if (children[index] == nullptr) {
            Vec3 childCenter = getOctantCenter(index);
            children[index] = new OctreeNode(childCenter, halfSize);
        }

        children[index]->insert(newBody);  // DE QUY
    }

    /**
```

```
201      * @brief Compute total mass and center of mass for Barnes-Hut
202      *
203      * THUAT TOAN COMPUTE MASS (Post-order traversal):
204      *
205      * 1. Neu la External rong: mass = 0
206      * 2. Neu la External co body: mass = body.mass, COM = body.pos
207      * 3. Neu la Internal:
208      *    a. Goi de quy cho tat ca con (POST-ORDER!)
209      *    b. Tong hop: totalMass = sum(child.totalMass)
210      *    c. COM = sum(child.COM * child.mass) / totalMass
211      */
212     void computeMassDistribution() {
213         // BASE CASE 1: External rong
214         if (isExternal && body == nullptr) {
215             totalMass = 0.0;
216             centerOfMass = Vec3();
217             return;
218         }
219 
220         // BASE CASE 2: External co body
221         if (isExternal && body != nullptr) {
222             totalMass = body->mass;
223             centerOfMass = body->pos;
224             return;
225         }
226 
227         // RECURSIVE CASE: Internal node
228         // Buoc 1: Tinh cho tat ca con truoc (POST-ORDER!)
229         for (int i = 0; i < 8; i++) {
230             if (children[i] != nullptr) {
231                 children[i]->computeMassDistribution();
232             }
233         }
234 
235         // Buoc 2: Tong hop tu cac con
236         totalMass = 0.0;
237         Vec3 weightedSum;
238 
239         for (int i = 0; i < 8; i++) {
240             if (children[i] != nullptr && children[i]->totalMass > 0.0) {
241                 totalMass += children[i]->totalMass;
242                 weightedSum += children[i]->centerOfMass * children[i]->
    totalMass;
243             }
244         }
245 
246         // Buoc 3: Tinh Center of Mass
247         if (totalMass > 0.0) {
248             centerOfMass = weightedSum / totalMass;
249         }
250     }
251 
252     /**
253      * @brief Calculate gravitational force on a target body using Barnes-
    Hut
254      * @param target Body to calculate force on
255      * @param theta Accuracy parameter (typically 0.5)
256      * @param G Gravitational constant
```

```
257      * @param softening Softening parameter to avoid singularity
258      * @return Acceleration vector
259      *
260      * THUAT TOAN BARNES-HUT:
261      *
262      * 1. Neu node rong: return 0
263      * 2. Neu la External co body (va khong phai chinh no):
264      *    -> Tinh luc truc tiep
265      * 3. Neu la Internal:
266      *    a. Tinh ratio = s / d (kich thuoc / khoang cach)
267      *    b. Neu ratio < theta: "Du xa", dung centerOfMass de tinh
268      *    c. Neu ratio >= theta: "Qua gan", di sau vao cac con
269      */
270     Vec3 calculateForce(const Body* target, double theta,
271                         double G, double softening) const {
272         // CASE 1: Node rong
273         if (totalMass == 0.0) {
274             return Vec3();
275         }
276 
277         // CASE 2: External co body
278         if (isExternal && body != nullptr) {
279             // Khong tinh luc len chinh no
280             if (body == target) {
281                 return Vec3();
282             }
283 
284             // Tinh luc truc tiep
285             Vec3 r = body->pos - target->pos;
286             double distSq = r.magnitudeSq();
287             double dist = std::sqrt(distSq + softening * softening);
288 
289             // F = G * m1 * m2 / r^2
290             // a = F / m1 = G * m2 / r^2
291             // Huong: r / |r| (normalized)
292             // Ket hop: a = G * m * r / |r|^3
293             double magnitude = G * body->mass / (dist * dist * dist);
294             return r * magnitude;
295         }
296 
297         // CASE 3: Internal node - BARNES-HUT DECISION
298         Vec3 r = centerOfMass - target->pos;
299         double distSq = r.magnitudeSq();
300         double dist = std::sqrt(distSq);
301 
302         // Kich thuoc node (canh day du)
303         double s = halfSize * 2.0;
304 
305         // *** MULTIPOLE ACCEPTANCE CRITERION ***
306         // s/d < theta nghia la: "cum nay du xa de xap xi"
307         if (dist > 0 && (s / dist) < theta) {
308             // Du xa -> Coi nhu 1 khoi luong duy nhat tai centerOfMass
309             double distSoftened = std::sqrt(distSq + softening * softening
    );
310             double magnitude = G * totalMass / (distSoftened *
    distSoftened * distSoftened);
311             return r * magnitude;
312         }
```

```
313
314         // Qua gan -> Phai di sau vao tung con
315         Vec3 totalForce;
316         for (int i = 0; i < 8; i++) {
317             if (children[i] != nullptr) {
318                 totalForce += children[i]->calculateForce(target, theta, G
    , softening);
319             }
320         }
321
322         return totalForce;
323     }
324
325     // ==================== DEBUG & VISUALIZATION ====================
326
327     /**
328      * @brief Draw wireframe representation of the octree
329      */
330     void drawWireframe(Color color = GREEN) const {
331         // Convert to Raylib types
332         Vector3 c = {
333             static_cast<float>(center.x),
334             static_cast<float>(center.y),
335             static_cast<float>(center.z)
336         };
337         float size = static_cast<float>(halfSize * 2.0);
338
339         // Draw this node's cube
340         DrawCubeWires(c, size, size, size, color);
341
342         // Recursively draw children
343         if (!isExternal) {
344             for (int i = 0; i < 8; i++) {
345                 if (children[i] != nullptr) {
346                     children[i]->drawWireframe(color);
347                 }
348             }
349         }
350     }
351
352     /**
353      * @brief Count total nodes in the tree (for debugging)
354      */
355     int countNodes() const {
356         int count = 1;  // This node
357         if (!isExternal) {
358             for (int i = 0; i < 8; i++) {
359                 if (children[i] != nullptr) {
360                     count += children[i]->countNodes();
361                 }
362             }
363         }
364         return count;
365     }
366
367     /**
368      * @brief Get maximum depth of the tree
369      */
```

```
    int maxDepth() const {
        if (isExternal) return 1;

        int maxChildDepth = 0;
        for (int i = 0; i < 8; i++) {
            if (children[i] != nullptr) {
                maxChildDepth = std::max(maxChildDepth, children[i]->
    maxDepth());
            }
        }
        return 1 + maxChildDepth;
    }
};
```

Listing 16.3: include/OctreeNode.hpp

16.4 Galaxy.hpp - Galaxy Initialization

```
#pragma once
#include "Body.hpp"
#include <vector>
#include <random>
#include <cmath>

/**
 * @brief Galaxy initialization utilities
 *
 * Cung cap cac ham de tao galaxy ban dau voi:
 * - Phan bo khoi luong thuc te
 * - Van toc xoay de galaxy on dinh
 */

namespace Galaxy {

/**
 * @brief Create a disk-shaped galaxy
 * @param numBodies Number of stars
 * @param diskRadius Radius of the disk
 * @param diskHeight Thickness of the disk
 * @param centralMass Mass of central black hole (optional)
 * @return Vector of initialized bodies
 */
std::vector<Body> createDiskGalaxy(int numBodies, double diskRadius,
                                   double diskHeight, double centralMass
    = 0.0) {
    std::vector<Body> bodies;
    bodies.reserve(numBodies);

    // Random number generator
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<double> angleDist(0.0, 2.0 * M_PI);
    std::uniform_real_distribution<double> heightDist(-diskHeight/2,
    diskHeight/2);
    std::uniform_real_distribution<double> massDist(0.5, 2.0);  // Solar
    masses
```

```
36
37      // Exponential radius distribution (more stars near center)
38      std::exponential_distribution<double> radiusDist(3.0 / diskRadius);
39
40      // Optional: Central massive object (black hole)
41      if (centralMass > 0.0) {
42          Body centralBody(Vec3(0, 0, 0), Vec3(0, 0, 0), centralMass);
43          centralBody.color = YELLOW;
44          centralBody.radius = 0.5f;
45          bodies.push_back(centralBody);
46      }
47
48      // Create stars
49      for (int i = 0; i < numBodies; i++) {
50          // Position in disk
51          double r = std::min(radiusDist(gen), diskRadius);
52          double theta = angleDist(gen);
53          double h = heightDist(gen);
54
55          Vec3 pos(r * cos(theta), h, r * sin(theta));
56
57          // Velocity: Orbital velocity for circular orbit
58          // v = sqrt(G * M_enclosed / r)
59          // Simplified: Assume enclosed mass proportional to r
60          double orbitalSpeed = 0.0;
61          if (r > 0.1) {
62              // Empirical formula for realistic rotation curve
63              double enclosedMass = centralMass + (double)i * 1.0 * (r /
    diskRadius);
64              orbitalSpeed = sqrt(0.1 * enclosedMass / r);  // Simplified
65          }
66
67          // Perpendicular to radius (tangent direction for orbit)
68          Vec3 vel(-orbitalSpeed * sin(theta), 0.0, orbitalSpeed * cos(theta
    ));
69
70          // Random mass
71          double mass = massDist(gen);
72
73          Body star(pos, vel, mass);
74          bodies.push_back(star);
75      }
76
77      return bodies;
78  }
79
80  /**
81   * @brief Create a spherical cluster
82   * @param numBodies Number of stars
83   * @param clusterRadius Radius of sphere
84   * @return Vector of initialized bodies
85   */
86  std::vector<Body> createSphereCluster(int numBodies, double clusterRadius)
        {
87      std::vector<Body> bodies;
88      bodies.reserve(numBodies);
89
90      std::random_device rd;
```

```
    std::mt19937 gen(rd());
    std::uniform_real_distribution<double> dist(-1.0, 1.0);
    std::uniform_real_distribution<double> massDist(0.5, 2.0);

    for (int i = 0; i < numBodies; i++) {
        // Random point in sphere using rejection sampling
        Vec3 pos;
        do {
            pos = Vec3(dist(gen), dist(gen), dist(gen));
        } while (pos.magnitudeSq() > 1.0);

        pos = pos * clusterRadius;

        // Small random velocity
        Vec3 vel(dist(gen) * 0.1, dist(gen) * 0.1, dist(gen) * 0.1);

        double mass = massDist(gen);

        bodies.push_back(Body(pos, vel, mass));
    }

    return bodies;
}

/**
 * @brief Create two colliding galaxies
 */
std::vector<Body> createCollidingGalaxies(int bodiesPerGalaxy, double
    separation) {
    // Galaxy 1 at origin
    auto galaxy1 = createDiskGalaxy(bodiesPerGalaxy, 20.0, 2.0, 100.0);

    // Galaxy 2 offset and moving toward galaxy 1
    auto galaxy2 = createDiskGalaxy(bodiesPerGalaxy, 20.0, 2.0, 100.0);

    // Offset galaxy 2
    for (auto& body : galaxy2) {
        body.pos.x += separation;
        body.vel.x -= 0.5;  // Moving toward galaxy 1
    }

    // Combine
    galaxy1.insert(galaxy1.end(), galaxy2.begin(), galaxy2.end());
    return galaxy1;
}

} // namespace Galaxy
```

Listing 16.4: include/Galaxy.hpp

16.5 main.cpp - Complete Simulation Loop

```
/**
 * @file main.cpp
 * @brief 3D N-Body Simulation using Barnes-Hut Algorithm
 *
```

```
 * Main entry point for the simulation.
 * Handles:
 * - Raylib window initialization
 * - Main loop (Input -> Physics -> Render)
 * - Camera controls
 * - Performance monitoring
 */

#include "raylib.h"
#include "Vec3.hpp"
#include "Body.hpp"
#include "OctreeNode.hpp"
#include "Galaxy.hpp"

#include <vector>
#include <iostream>
#include <chrono>

// ==================== SIMULATION CONSTANTS ====================

namespace Config {
    // Window
    constexpr int SCREEN_WIDTH = 1280;
    constexpr int SCREEN_HEIGHT = 720;
    constexpr int TARGET_FPS = 60;

    // Simulation
    constexpr int NUM_BODIES = 10000;
    constexpr double UNIVERSE_SIZE = 200.0;  // Half-size of simulation space
    constexpr double G = 1.0;                 // Gravitational constant (scaled)
    constexpr double SOFTENING = 0.5;         // Softening parameter
    constexpr double DT = 0.016;              // Time step (~60 FPS)
    constexpr double THETA = 0.5;             // Barnes-Hut accuracy (0.5 is typical)

    // Visual
    constexpr bool DRAW_OCTREE = false;       // Toggle octree visualization
}

// ==================== COLOR MAPPING ====================

/**
 * @brief Map star mass to stellar color (Red Dwarf -> Blue Giant)
 */
Color massToColor(double mass, double minMass, double maxMass) {
    double t = (mass - minMass) / (maxMass - minMass);
    t = fmax(0.0, fmin(1.0, t));  // Clamp to [0, 1]

    unsigned char r, g, b;

    if (t < 0.33) {
        // Red -> Orange
        double lt = t / 0.33;
        r = 200;
        g = static_cast<unsigned char>(50 + 150 * lt);
```

```
59          b = 50;
60      } else if (t < 0.66) {
61          // Orange -> White
62          double lt = (t - 0.33) / 0.33;
63          r = 255;
64          g = static_cast<unsigned char>(200 + 55 * lt);
65          b = static_cast<unsigned char>(50 + 205 * lt);
66      } else {
67          // White -> Blue
68          double lt = (t - 0.66) / 0.34;
69          r = static_cast<unsigned char>(255 - 100 * lt);
70          g = static_cast<unsigned char>(255 - 100 * lt);
71          b = 255;
72      }
73
74      return Color{r, g, b, 255};
75  }
76
77  /**
78   * @brief Apply colors to all bodies based on mass
79   */
80  void applyMassColors(std::vector<Body>& bodies) {
81      if (bodies.empty()) return;
82
83      // Find mass range
84      double minMass = bodies[0].mass;
85      double maxMass = bodies[0].mass;
86      for (const auto& b : bodies) {
87          minMass = fmin(minMass, b.mass);
88          maxMass = fmax(maxMass, b.mass);
89      }
90
91      // Apply colors
92      for (auto& b : bodies) {
93          b.color = massToColor(b.mass, minMass, maxMass);
94          b.radius = 0.05f + 0.1f * static_cast<float>((b.mass - minMass) /
    (maxMass - minMass));
95      }
96  }
97
98  // ==================== MAIN FUNCTION ====================
99
100 int main() {
101     // ========== INITIALIZATION ==========
102
103     // Initialize Raylib window
104     InitWindow(Config::SCREEN_WIDTH, Config::SCREEN_HEIGHT,
105                "N-Body Simulation - Barnes-Hut Algorithm");
106     SetTargetFPS(Config::TARGET_FPS);
107
108     // Initialize 3D camera
109     Camera3D camera = { 0 };
110     camera.position = Vector3{ 100.0f, 80.0f, 100.0f };
111     camera.target = Vector3{ 0.0f, 0.0f, 0.0f };
112     camera.up = Vector3{ 0.0f, 1.0f, 0.0f };
113     camera.fovy = 60.0f;
114     camera.projection = CAMERA_PERSPECTIVE;
115
```

```
116     // Create galaxy
117     std::cout << "Initializing " << Config::NUM_BODIES << " bodies..." <<
    std::endl;
118     std::vector<Body> bodies = Galaxy::createDiskGalaxy(
119         Config::NUM_BODIES,
120         50.0,    // Disk radius
121         5.0,     // Disk height
122         1000.0   // Central mass
123     );
124 
125     // Apply colors based on mass
126     applyMassColors(bodies);
127 
128     std::cout << "Simulation ready! Bodies: " << bodies.size() << std::
    endl;
129 
130     // Performance metrics
131     double physicsTime = 0.0;
132     int frameCount = 0;
133 
134     // ========== MAIN LOOP ==========
135 
136     while (!WindowShouldClose()) {
137         // ====== INPUT ======
138         UpdateCamera(&camera, CAMERA_ORBITAL);
139 
140         // Keyboard controls
141         if (IsKeyPressed(KEY_R)) {
142             // Reset simulation
143             bodies = Galaxy::createDiskGalaxy(Config::NUM_BODIES, 50.0,
    5.0, 1000.0);
144             applyMassColors(bodies);
145         }
146 
147         if (IsKeyPressed(KEY_SPACE)) {
148             // Toggle pause (not implemented - exercise for reader)
149         }
150 
151         // ====== PHYSICS (Barnes-Hut) ======
152         auto physicsStart = std::chrono::high_resolution_clock::now();
153 
154         // Step 1: Build Octree
155         Vec3 center(0, 0, 0);
156         OctreeNode* root = new OctreeNode(center, Config::UNIVERSE_SIZE *
    2);
157 
158         for (auto& body : bodies) {
159             root->insert(&body);
160         }
161 
162         // Step 2: Compute mass distribution
163         root->computeMassDistribution();
164 
165         // Step 3: Calculate forces using Barnes-Hut
166         #ifdef _OPENMP
167         #pragma omp parallel for
168         #endif
169         for (size_t i = 0; i < bodies.size(); i++) {
```

```
            bodies[i].acc = root->calculateForce(
                &bodies[i],
                Config::THETA,
                Config::G,
                Config::SOFTENING
            );
        }

        // Step 4: Update positions
        for (auto& body : bodies) {
            body.update(Config::DT);
        }

        // Cleanup
        int treeNodes = root->countNodes();
        int treeDepth = root->maxDepth();
        delete root;
        root = nullptr;

        auto physicsEnd = std::chrono::high_resolution_clock::now();
        physicsTime = std::chrono::duration<double, std::milli>(physicsEnd - physicsStart).count();

        // ====== RENDERING ======
        BeginDrawing();
            ClearBackground(Color{5, 5, 15, 255});  // Deep space blue

            BeginMode3D(camera);
                // Draw all bodies
                for (const auto& body : bodies) {
                    body.draw();
                }

                // Optional: Draw octree wireframe
                if (Config::DRAW_OCTREE && root != nullptr) {
                    root->drawWireframe(Color{0, 100, 0, 100});
                }

                // Reference grid
                DrawGrid(20, 10.0f);
            EndMode3D();

            // UI Overlay
            DrawRectangle(10, 10, 300, 150, Color{0, 0, 0, 150});
            DrawText("N-Body Simulation (Barnes-Hut)", 20, 20, 20, WHITE);
            DrawText(TextFormat("Bodies: %d", (int)bodies.size()), 20, 50, 16, GREEN);
            DrawText(TextFormat("FPS: %d", GetFPS()), 20, 70, 16, GREEN);
            DrawText(TextFormat("Physics: %.2f ms", physicsTime), 20, 90, 16, YELLOW);
            DrawText(TextFormat("Tree Nodes: %d", treeNodes), 20, 110, 16, GRAY);
            DrawText(TextFormat("Tree Depth: %d", treeDepth), 20, 130, 16, GRAY);

            DrawText("Controls: Mouse to orbit, R to reset", 20, SCREEN_HEIGHT - 30, 16, GRAY);
        EndDrawing();
```

```
222
223         frameCount++;
224     }
225
226     // ========== CLEANUP ==========
227     CloseWindow();
228
229     return 0;
230 }
```

Listing 16.5: src/main.cpp

Phần V

Vật lý và Toán học Đồ họa

Chương 17

Newton's Law of Gravitation

17.1 Công thức Lực Hấp dẫn

Lực hút giữa hai vật thể i và j được mô tả bởi định luật Newton:

$$\boxed{\vec{F}_{ij} = G\frac{m_i m_j}{r_{ij}^2}\hat{r}_{ij}} \tag{17.1}$$

Trong đó:

- $G = 6.674 \times 10^{-11}\ \text{N} \cdot \text{m}^2/\text{kg}^2$ — Hằng số hấp dẫn
- m_i, m_j — Khối lượng của hai vật thể
- $r_{ij} = |\vec{r}_j - \vec{r}_i|$ — Khoảng cách giữa hai vật thể
- $\hat{r}_{ij}$ — Vector đơn vị hướng từ i đến j

17.2 Từ Lực đến Gia tốc

Theo định luật Newton thứ 2: $\vec{F} = m\vec{a}$

Do đó, gia tốc của vật thể i do vật thể j gây ra:

$$\vec{a}_{ij} = \frac{\vec{F}_{ij}}{m_i} = G\frac{m_j}{r_{ij}^2}\hat{r}_{ij} \tag{17.2}$$

Quan trọng

Trong code, ta tính gia tốc (không phải lực) vì:

1. Không cần nhân với khối lượng của target body
2. Trực tiếp cộng vào velocity khi update

17.3 Softening Parameter (ϵ)

17.3.1 Vấn đề: Division by Zero

Khi hai vật thể quá gần nhau ($r \to 0$):

$$\vec{a} = G\frac{m}{r^2} \to \infty \tag{17.3}$$

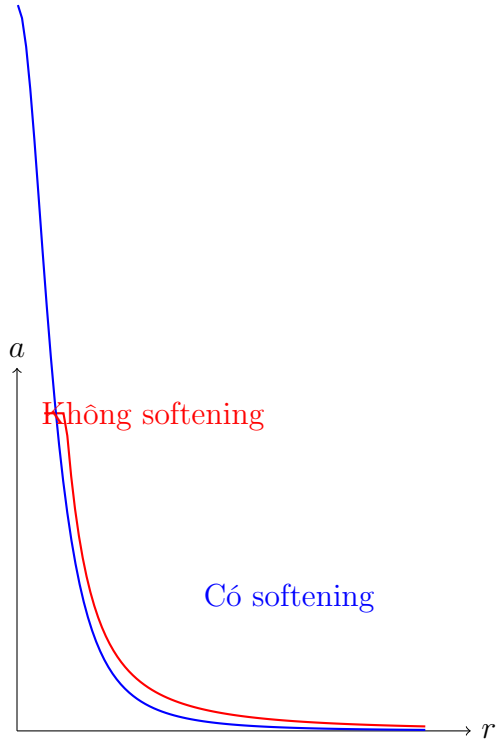
Điều này gây ra:

- Numerical overflow trong code
- Vận tốc tăng vọt phi thực tế
- Vật thể "bắn"ra khỏi simulation

17.3.2 Giải pháp: Plummer Softening

Thay r bằng $\sqrt{r^2 + \epsilon^2}$:

$$\boxed{\vec{a}_{ij} = G\frac{m_j}{(r_{ij}^2 + \epsilon^2)^{3/2}}\vec{r}_{ij}} \tag{17.4}$$



Hình 17.1: So sánh gia tốc có và không có softening

Mẹo

Chọn giá trị ϵ:

- Quá nhỏ: Vẫn có numerical instability
- Quá lớn: Mất độ chính xác, các sao không tương tác đủ mạnh
- **Khuyến nghị:** $\epsilon \approx$ kích thước trung bình của vật thể, hoặc $\sim$1% khoảng cách trung bình

17.4 Implementation trong Code

```
/**
 * @brief Calculate gravitational acceleration with softening
 * @param target The body receiving the force
 * @param source The body exerting the force
 * @param G Gravitational constant
 * @param epsilon Softening parameter
 * @return Acceleration vector
 */
Vec3 calculateGravitationalAcceleration(
    const Body& target,
    const Body& source,
    double G,
    double epsilon
) {
    // Step 1: Compute displacement vector (from target to source)
    Vec3 r = source.pos - target.pos;

    // Step 2: Compute squared distance
    double distSq = r.x*r.x + r.y*r.y + r.z*r.z;

    // Step 3: Apply softening
    // d = sqrt(r^2 + eps^2)
    double softDistSq = distSq + epsilon * epsilon;
    double softDist = std::sqrt(softDistSq);

    // Step 4: Compute acceleration magnitude
    // |a| = G * M / d^2
    // But we need direction, so: a = G * M * r / d^3
    // (r/d gives unit vector, then multiply by G*M/d^2)
    double magnitude = G * source.mass / (softDist * softDist * softDist);

    // Step 5: Return acceleration vector
    return r * magnitude;
}
```

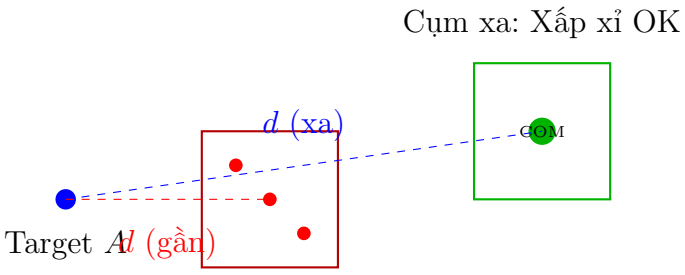
Listing 17.1: Complete Force Calculation

Chương 18

Barnes-Hut Multipole Acceptance Criterion

18.1 Ý tưởng Cốt lõi

Barnes-Hut dựa trên quan sát: **Nếu một cụm sao ở đủ xa, ta có thể coi cả cụm như một điểm khối lượng duy nhất.**



Hình 18.1: Minh họa tiêu chuẩn Barnes-Hut

18.2 Tiêu chuẩn Multipole

Gọi:

- s = kích thước cạnh của node Octree
- d = khoảng cách từ target đến center of mass của node
- θ = tham số độ chính xác (thường chọn 0.5)

$$\boxed{\text{Điều kiện xấp xỉ: } \frac{s}{d} < \theta} \tag{18.1}$$

- $\frac{s}{d} < \theta$: Node này "đủ xa"⇒ Dùng center of mass
- $\frac{s}{d} \geq \theta$: Node này "quá gần"⇒ Đi sâu vào con

18.3 Ảnh hưởng của θ

Bảng 18.1: Trade-off giữa độ chính xác và tốc độ

θ	**Độ chính xác**	**Tốc độ**	**Ghi chú**
0	Tuyệt đối (= Brute-force)	Chậm nhất	$O(N^2)$
0.3	Rất cao	Chậm	Scientific simulations
0.5	**Tốt**	**Nhanh**	**Khuyến nghị**
0.7	Trung bình	Rất nhanh	Game/Visualization
1.0+	Thấp	Nhanh nhất	Không khuyến nghị

18.4 Pseudo-code Barnes-Hut

Algorithm 1 Barnes-Hut calculateForce

function CalculateForce(node, targetBody, θ, G, ϵ)
  if node is empty **then**
    return $\vec{0}$
  end if
  if node is external AND contains a body **then**
    if body == targetBody **then**
      return $\vec{0}$ ▷ Không tự hấp dẫn
    end if
    return DirectForce(targetBody, node.body, G, ϵ)
  end if
  $\vec{r} \leftarrow$ node.centerOfMass − targetBody.pos
  $d \leftarrow |\vec{r}|$
  $s \leftarrow$ node.size
  if $s/d < \theta$ **then** ▷ Multipole criterion
    return ApproximateForce(targetBody, node.totalMass, node.centerOfMass, G, ϵ)
  else
    $\vec{F} \leftarrow \vec{0}$
    for each child in node.children **do**
      $\vec{F} \leftarrow \vec{F}+$ CalculateForce(child, targetBody, θ, G, ϵ)
    end for
    return $\vec{F}$
  end if
end function

Chương 19

Mass-based Star Coloring

19.1 Khoa học Màu sắc Sao

Trong thiên văn học, màu sắc của ngôi sao phụ thuộc vào **nhiệt độ bề mặt**, liên quan đến khối lượng:

Bảng 19.1: Phân loại Quang phổ Sao

Loại	Màu	Nhiệt độ (K)	Khối lượng ($M_\odot$)
M	Đỏ	2,500 - 3,500	0.08 - 0.5
K	Cam	3,500 - 5,000	0.5 - 0.8
G	Vàng (như Mặt Trời)	5,000 - 6,000	0.8 - 1.1
F	Trắng-Vàng	6,000 - 7,500	1.1 - 1.5
A	Trắng	7,500 - 10,000	1.5 - 2.5
B	Xanh-Trắng	10,000 - 30,000	2.5 - 16
O	Xanh	30,000+	16+

19.2 Thuật toán Color Mapping

```
/**
 * @brief Map star mass to realistic stellar color
 * @param mass The star's mass
 * @param minMass Minimum mass in simulation
 * @param maxMass Maximum mass in simulation
 * @return RGB Color
 *
 * THUAT TOAN:
 * 1. Normalize mass to range [0, 1]
 * 2. Map to color gradient:
 *    0.0 -> Red (M-class, Red Dwarf)
 *    0.25 -> Orange (K-class)
 *    0.5 -> Yellow (G-class, like Sun)
 *    0.75 -> White (F/A-class)
 *    1.0 -> Blue (O/B-class, Blue Giant)
 */
Color massToStellarColor(double mass, double minMass, double maxMass) {
    // Step 1: Normalize to [0, 1]
    double t = (mass - minMass) / (maxMass - minMass);
    t = std::clamp(t, 0.0, 1.0);  // C++17 clamp
```

```
21
22      // Step 2: Interpolate through color gradient
23      // Using piecewise linear interpolation
24
25      unsigned char r, g, b;
26
27      if (t < 0.2) {
28          // ===== RED DWARF ZONE (M-class) =====
29          // Deep Red (139, 0, 0) -> Bright Red (255, 69, 0)
30          double lt = t / 0.2;
31          r = static_cast<unsigned char>(139 + 116 * lt);  // 139 -> 255
32          g = static_cast<unsigned char>(0 + 69 * lt);      // 0 -> 69
33          b = 0;
34      }
35      else if (t < 0.4) {
36          // ===== ORANGE ZONE (K-class) =====
37          // Orange-Red (255, 69, 0) -> Orange (255, 140, 0)
38          double lt = (t - 0.2) / 0.2;
39          r = 255;
40          g = static_cast<unsigned char>(69 + 71 * lt);    // 69 -> 140
41          b = 0;
42      }
43      else if (t < 0.6) {
44          // ===== YELLOW ZONE (G-class, Sun-like) =====
45          // Orange (255, 140, 0) -> Yellow (255, 255, 100)
46          double lt = (t - 0.4) / 0.2;
47          r = 255;
48          g = static_cast<unsigned char>(140 + 115 * lt);  // 140 -> 255
49          b = static_cast<unsigned char>(0 + 100 * lt);    // 0 -> 100
50      }
51      else if (t < 0.8) {
52          // ===== WHITE ZONE (F/A-class) =====
53          // Yellow-White (255, 255, 100) -> Pure White (255, 255, 255)
54          double lt = (t - 0.6) / 0.2;
55          r = 255;
56          g = 255;
57          b = static_cast<unsigned char>(100 + 155 * lt);  // 100 -> 255
58      }
59      else {
60          // ===== BLUE ZONE (O/B-class, Blue Giant) =====
61          // White (255, 255, 255) -> Blue (155, 176, 255)
62          double lt = (t - 0.8) / 0.2;
63          r = static_cast<unsigned char>(255 - 100 * lt);  // 255 -> 155
64          g = static_cast<unsigned char>(255 - 79 * lt);   // 255 -> 176
65          b = 255;
66      }
67
68      return Color{r, g, b, 255};
69  }
```

Listing 19.1: Mass to Stellar Color

19.3 Visual Size based on Mass

Ngoài màu sắc, ta còn có thể điều chỉnh kích thước hiển thị theo khối lượng:

```
1  /**
```

```
 * @brief Map mass to visual radius for rendering
 *
 * Note: Trong thuc te, ban kinh sao ~ M^0.8 (Main Sequence)
 * Nhung de visual dep hon, ta dung logarithmic scaling
 */
float massToRadius(double mass, double minMass, double maxMass) {
    const float MIN_RADIUS = 0.03f;
    const float MAX_RADIUS = 0.3f;

    // Logarithmic scaling (looks better visually)
    double logMin = std::log(minMass);
    double logMax = std::log(maxMass);
    double logMass = std::log(mass);

    double t = (logMass - logMin) / (logMax - logMin);
    t = std::clamp(t, 0.0, 1.0);

    return MIN_RADIUS + static_cast<float>(t) * (MAX_RADIUS - MIN_RADIUS);
}

// Trong main loop:
void applyVisuals(std::vector<Body>& bodies) {
    // Tim min/max mass
    double minM = bodies[0].mass, maxM = bodies[0].mass;
    for (const auto& b : bodies) {
        minM = std::min(minM, b.mass);
        maxM = std::max(maxM, b.mass);
    }

    // Apply
    for (auto& b : bodies) {
        b.color = massToStellarColor(b.mass, minM, maxM);
        b.radius = massToRadius(b.mass, minM, maxM);
    }
}
```

Listing 19.2: Mass to Visual Radius

Phần VI

Tính năng Nâng cao (Tùy chọn)

Chương 20

Collision Detection & Merging

20.1 Phát hiện Va chạm

Hiện tại các sao đi xuyên qua nhau. Để thực tế hơn:

```
/**
 * @brief Check and handle collisions between bodies
 * @param bodies Vector of all bodies
 *
 * Note: O(N^2) - Trong production, dung spatial hashing
 */
void handleCollisions(std::vector<Body>& bodies) {
    std::vector<int> toRemove;

    for (size_t i = 0; i < bodies.size(); i++) {
        for (size_t j = i + 1; j < bodies.size(); j++) {
            double dist = bodies[i].pos.distance(bodies[j].pos);
            double minDist = bodies[i].radius + bodies[j].radius;

            if (dist < minDist) {
                // INELASTIC COLLISION: Merge into body i
                // Conservation of momentum: m1*v1 + m2*v2 = (m1+m2)*
    v_final
                double totalMass = bodies[i].mass + bodies[j].mass;

                bodies[i].vel = (bodies[i].vel * bodies[i].mass +
                                 bodies[j].vel * bodies[j].mass) /
    totalMass;

                // Center of mass position
                bodies[i].pos = (bodies[i].pos * bodies[i].mass +
                                 bodies[j].pos * bodies[j].mass) /
    totalMass;

                bodies[i].mass = totalMass;

                toRemove.push_back(j);
            }
        }
    }

    // Remove merged bodies (reverse order to preserve indices)
    std::sort(toRemove.begin(), toRemove.end(), std::greater<int>());
    for (int idx : toRemove) {
```

```
37          bodies.erase(bodies.begin() + idx);
38      }
39  }
```

Listing 20.1: Simple Collision Detection

Chương 21

OpenMP Parallelization

21.1 Tối ưu hóa với Đa luồng

```cpp
#include <omp.h>

// Chi can them 1 dong de song song hoa tinh luc!
void updateWithOpenMP(std::vector<Body>& bodies, OctreeNode* root) {
    // Parallel force calculation
    // Moi thread tinh luc cho mot nhom bodies
    #pragma omp parallel for schedule(dynamic)
    for (size_t i = 0; i < bodies.size(); i++) {
        bodies[i].acc = root->calculateForce(
            &bodies[i], THETA, G, SOFTENING
        );
    }

    // Parallel position update
    #pragma omp parallel for
    for (size_t i = 0; i < bodies.size(); i++) {
        bodies[i].update(DT);
    }
}

// Compile: g++ -O3 -fopenmp main.cpp -o simulation
```

Listing 21.1: OpenMP Integration

Mẹo

Hiệu quả OpenMP:

- 4-core CPU: Tăng tốc ∼3.5x
- 8-core CPU: Tăng tốc ∼6-7x
- Schedule `dynamic` tốt hơn `static` vì workload không đều (phụ thuộc vị trí body trong tree)

Chương 22

Visual Effects: Trails & Bloom

22.1 Star Trails (Vệt đuôi sao)

```cpp
// Luu lich su vi tri
struct Body {
    // ... existing members ...
    std::deque<Vec3> trail;  // Vi tri truoc do
    static const int TRAIL_LENGTH = 50;

    void updateTrail() {
        trail.push_front(pos);
        if (trail.size() > TRAIL_LENGTH) {
            trail.pop_back();
        }
    }

    void drawTrail() const {
        for (size_t i = 1; i < trail.size(); i++) {
            // Fade out theo vi tri
            unsigned char alpha = 255 - (255 * i / TRAIL_LENGTH);
            Color trailColor = {color.r, color.g, color.b, alpha};

            Vector3 start = toRaylib(trail[i-1]);
            Vector3 end = toRaylib(trail[i]);
            DrawLine3D(start, end, trailColor);
        }
    }
};
```

Listing 22.1: Simple Trail Effect

Chương 23

Kết luận

23.1 Những gì nhóm sẽ đạt được

Sau 8 tuần thực hiện đồ án này, nhóm sẽ:

1. **Kiến thức DSA:**
   - Hiểu sâu về Con trỏ và Quản lý bộ nhớ động
   - Thành thạo Đệ quy qua Tree traversal
   - Nắm vững phân tích Big-O và tối ưu thuật toán
2. **Kỹ năng lập trình:**
   - Viết C++ sạch theo Modern conventions
   - Sử dụng thư viện đồ họa (Raylib)
   - Debug và optimize code hiệu quả
3. **Kiến thức khoa học:**
   - Hiểu vật lý Newton và mô phỏng số
   - Biết về thiên văn học cơ bản (màu sao, cấu trúc thiên hà)
4. **Sản phẩm:**
   - Một mô phỏng vũ trụ 3D đẹp mắt
   - Portfolio project ấn tượng cho CV

23.2 Lời khuyên cuối cùng

⚠ Quan trọng

1. **Đừng sợ sai:** Bugs là thầy giáo tốt nhất. Mỗi Segmentation Fault dạy bạn về pointers.
2. **Commit thường xuyên:** Dùng Git, commit mỗi khi hoàn thành một feature nhỏ.

3. **Visual debugging:** Vẽ Octree wireframe giúp debug cực kỳ hiệu quả.
4. **Incremental development:** Brute-force trước, Barnes-Hut sau. Đừng cố làm tất cả cùng lúc.
5. **Hỏi khi cần:** Stack Overflow, GitHub Issues, và thầy/cô luôn sẵn sàng giúp đỡ.

"The cosmos is within us. We are made of star-stuff."
— Carl Sagan

LET'S BUILD THIS UNIVERSE!