



**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:

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

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  $6 \times 10^9$  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 // 1. Khai báo con trỏ
2 int x = 10;
3 int* ptr = &x; // ptr LUU DIA CHI cua x
4
5 // 2. Truy cập giá trị qua con trỏ (Dereference)
6 int value = *ptr; // value = 10
7
8 // 3. nullptr - con trỏ không trỏ đến đâu cả
9 int* safePtr = nullptr; // LUON khai tạo con trỏ = nullptr
10
11 // 4. Kiểm tra trước khi dùng
12 if (ptr != nullptr) {
13     *ptr = 20; // An toàn
14 }
```

Listing 2.1: Pointer Basics

#### 2.2.2 Dynamic Memory (Heap vs Stack)



```

1 // STACK: Tu dong giai phong khi ra khoi scope
2 void stackExample() {
3     int arr[100]; // 100 int tren Stack
4 } // arr tu dong mat khi ham ket thuc
5
6 // HEAP: Phai TU TAY giai phong
7 void heapExample() {
8     // Cap phat 1 object
9     int* single = new int(42);
10
11     // Cap phat mang
12     int* arr = new int[1000];
13
14     // !!! PHAI GIAI PHONG !!!
15     delete single; // Xoa 1 object
16     delete[] arr; // Xoa mang (chu y [])
17 }

```

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

```

1 // Day la ly do Octree can con tro
2 struct OctreeNode {
3     // Moi node co the co 8 con
4     // Neu dung object truc tiep -> VO HAN (infinite recursion)
5     // OctreeNode children[8]; // LOI! Khong the dinh nghia
6
7     // Phai dung CON TRO
8     OctreeNode* children[8]; // OK! Chi luu DIA CHI
9
10    OctreeNode() {
11        // Khoi tao tat ca con tro = nullptr
12        for (int i = 0; i < 8; i++) {
13            children[i] = nullptr;
14        }
15    }
16
17    ~OctreeNode() {
18        // Destructor: Giai phong tat ca node con (DE QUY!)
19        for (int i = 0; i < 8; i++) {
20            if (children[i] != nullptr) {
21                delete children[i]; // Goi destructor cua con
22                children[i] = nullptr;
23            }
24        }
25    }
26 };

```

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

```
1 void recursiveFunction(Input input) {
2     // 1. BASE CASE - Điều kiện dừng
3     //     Không có base case = STACK OVERFLOW!
4     if (/* điều kiện dừng */) {
5         return; // hoặc return giá trị
6     }
7
8     // 2. RECURSIVE CASE - Xử lý và gọi lại chính mình
9     //     Input PHẢI nhỏ hơn để tiến về base case
10    recursiveFunction(smallerInput);
11 }
12
13 // Ví dụ: Tính giai thừa
14 int factorial(int n) {
15     // Base case
16     if (n <= 1) return 1;
17
18     // Recursive case: n! = n * (n-1)!
19     return n * factorial(n - 1);
20 }
```

20 }

Listing 3.1: Anatomy of Recursion

### 3.3 Độ quy duyệt cây (Tree Traversal)

```

1 struct TreeNode {
2     int value;
3     TreeNode* left;
4     TreeNode* right;
5 };
6
7 // Pre-order: Xu ly node TRUOC khi xuong con
8 // Dung cho: In cay, copy cay
9 void preorder(TreeNode* node) {
10     if (node == nullptr) return; // Base case
11
12     process(node); // Xu ly truoc
13     preorder(node->left); // Xuong trai
14     preorder(node->right); // Xuong phai
15 }
16
17 // Post-order: Xu ly node SAU khi xuong con
18 // Dung cho: Tinh toan tu duoi len (Center of Mass!)
19 void postorder(TreeNode* node) {
20     if (node == nullptr) return; // Base case
21
22     postorder(node->left); // Xuong trai truoc
23     postorder(node->right); // Xuong phai
24     process(node); // Xu ly SAU CUNG
25 }
26
27 // In-order: Trai -> Node -> Phai (cho BST)
28 void inorder(TreeNode* node) {
29     if (node == nullptr) return;
30     inorder(node->left);
31     process(node);
32     inorder(node->right);
33 }

```

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: Độ quy trong Octree

```

1 void OctreeNode::computeMassDistribution() {
2     // BASE CASE 1: Node rỗng
3     if (isExternal && body == nullptr) {

```

```

4         totalMass = 0.0;
5         return;
6     }
7
8     // BASE CASE 2: La (External node) co body
9     if (isExternal && body != nullptr) {
10         totalMass = body->mass;
11         centerOfMass = body->pos;
12         return;
13     }
14
15     // RECURSIVE CASE: Node trong (Internal node)
16     // Buoc 1: Tinh khoi luong cua tat ca con truoc
17     for (int i = 0; i < 8; i++) {
18         if (children[i] != nullptr) {
19             children[i]->computeMassDistribution(); // DE QUY!
20         }
21     }
22
23     // Buoc 2: Tong hop tu cac con
24     totalMass = 0.0;
25     centerOfMass = Vec3(0, 0, 0);
26
27     for (int i = 0; i < 8; i++) {
28         if (children[i] != nullptr && children[i]->totalMass > 0) {
29             double childMass = children[i]->totalMass;
30             Vec3 childCOM = children[i]->centerOfMass;
31
32             // Cong thuc trong tam: COM = sum(m_i * r_i) / sum(m_i)
33             centerOfMass = centerOfMass + childCOM * childMass;
34             totalMass += childMass;
35         }
36     }
37
38     if (totalMass > 0) {
39         centerOfMass = centerOfMass / totalMass;
40     }
41 }

```

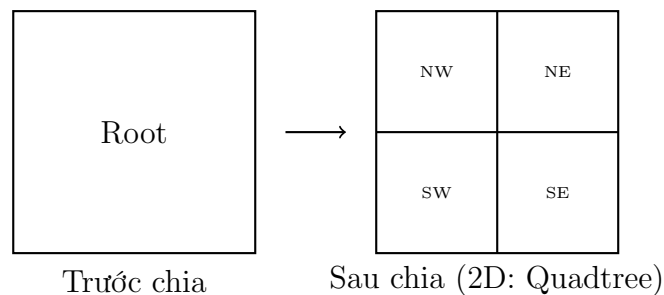
Listing 3.3: Recursive Mass Computation

# 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	≥ center	< center	< center
2	Bottom-North-West	< center	≥ center	< center
3	Bottom-North-East	≥ center	≥ center	< center
4	Top-South-West	< center	< center	≥ center
5	Top-South-East	≥ center	< center	≥ center
6	Top-North-West	< center	≥ center	≥ center
7	Top-North-East	≥ center	≥ center	≥ center

```

1 // Tính chỉ số octant (0-7) cho một điểm
2 int getOctantIndex(const Vec3& point, const Vec3& nodeCenter) {
3     int index = 0;
4
5     // Bit 0: X >= center.x
6     if (point.x >= nodeCenter.x) index |= 1;
7
8     // Bit 1: Y >= center.y
9     if (point.y >= nodeCenter.y) index |= 2;
10
11    // Bit 2: Z >= center.z
12    if (point.z >= nodeCenter.z) index |= 4;
13
14    return index; // Kết quả: 0 đến 7
15 }

```

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	<b>Barnes-Hut</b> , Merge Sort
$O(N^2)$	Quadratic	<b>Brute-force N-Body</b> , 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

```
1 // BRUTE-FORCE: O(N^2)
2 // Moi body phai xet TAT CA body khac
3 for (int i = 0; i < N; i++) {
4     for (int j = 0; j < N; j++) { // N iterations
5         if (i != j) {
6             bodies[i].force += calculatePairForce(bodies[i], bodies[j]);
7         }
8     }
9 }
10 // Tong: N * N = N^2 lan tinh
11
12 // BARNES-HUT: O(N log N)
13 // Moi body chi xet ~log(N) "cum"
14 for (int i = 0; i < N; i++) {
15     // calculateForce duyet cay, nhieu node duoc SKIP
16     bodies[i].force = root->calculateForce(&bodies[i], theta);
17 }
18 // 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

```
1 #include "raylib.h"
2
3 int main() {
4     // Khoi tao cua so 1280x720
5     InitWindow(1280, 720, "N-Body Simulation - Week 1");
6     SetTargetFPS(60);
7
8     // Thiet lap Camera 3D
9     Camera3D camera = { 0 };
10    camera.position = (Vector3){ 10.0f, 10.0f, 10.0f };
11    camera.target = (Vector3){ 0.0f, 0.0f, 0.0f };
12    camera.up = (Vector3){ 0.0f, 1.0f, 0.0f };
13    camera.fovy = 45.0f;
14    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 phép xoay camera
bang chuột
22
23         // Draw
24         BeginDrawing();
25         ClearBackground(BLACK);
26
27         BeginMode3D(camera);
28         // Ve hình lập phương 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

```
1 #pragma once
2 #include "Vec3.hpp"
3 #include "raylib.h"
4
5 struct Body {
6     Vec3 pos;        // Vị trí
7     Vec3 vel;        // Vận tốc
8     Vec3 acc;        // Gia tốc (tính mọi frame)
9     double mass;     // Khối lượng
10    Color color;      // Màu hiển thị
11
12    // Constructor
13    Body(Vec3 p = Vec3(), Vec3 v = Vec3(), double m = 1.0)
```

```

14         : pos(p), vel(v), acc(Vec3()), mass(m), color(WHITE) {}
15
16     // Cap nhat vi tri bang Euler Integration
17     void update(double dt) {
18         vel = vel + acc * dt;    // v = v + a*dt
19         pos = pos + vel * dt;    // x = x + v*dt
20         acc = Vec3();            // Reset gia toc cho frame sau
21     }
22
23     // Ve body len man hinh
24     void draw() const {
25         // Chuyen Vec3 sang Vector3 cua Raylib
26         Vector3 rlPos = { (float)pos.x, (float)pos.y, (float)pos.z };
27         DrawSphere(rlPos, 0.05f, color); // Ve hinh cau nho
28     }
29 };

```

Listing 8.1: Body.hpp

### 8.1.3 Code: Brute-Force Force Calculation

```

1 #include <vector>
2 #include "Body.hpp"
3
4 // Hang so vat ly
5 const double G = 6.67430e-11; // Hang so hap dan (SI units)
6 const double SOFTENING = 0.1; // Tranh chia cho 0
7
8 // Tinh luc hap dan giua 2 body
9 Vec3 calculatePairForce(const Body& a, const Body& b) {
10     Vec3 r = b.pos - a.pos; // Vector tu a den b
11     double distSq = r.x*r.x + r.y*r.y + r.z*r.z;
12
13     // Them softening de tranh vo cuc khi dist -> 0
14     double dist = sqrt(distSq + SOFTENING * SOFTENING);
15
16     // F = G * m1 * m2 / r^2
17     // a = F / m1 = G * m2 / r^2
18     // Huong: r / |r| (normalized)
19     double forceMagnitude = G * b.mass / (dist * dist * dist);
20
21     return r * forceMagnitude; // Vec3 acceleration
22 }
23
24 // BRUTE-FORCE: O(N^2)
25 void updateAllBodies_BruteForce(std::vector<Body>& bodies, double dt) {
26     int N = bodies.size();
27
28     // Tinh gia toc cho moi body
29     for (int i = 0; i < N; i++) {
30         Vec3 totalAcc;
31         for (int j = 0; j < N; j++) {
32             if (i != j) {
33                 totalAcc = totalAcc + calculatePairForce(bodies[i], bodies
34 [j]);
35             }
36         }
37     }
38 }

```

```

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

```
1 #pragma once
2 #include "Vec3.hpp"
3 #include "Body.hpp"
4 #include <array>
5
6 class OctreeNode {
7 public:
8     // ===== MEMBER VARIABLES =====
9
10    Vec3 center;           // Tam của hình lập phương
11    double halfSize;       // Nửa cạnh (để tính bounds)
12
13    // === Dữ liệu vật lý (dùng cho Barnes-Hut) ===
14    double totalMass;       // Tổng khối lượng của node và các con
15    Vec3 centerOfMass;      // Trọng tâm khối lượng
16
17    // === Cấu trúc cây ===
18    Body* body;             // Chỉ dùng khi là External Node (là)
19    std::array<OctreeNode*, 8> children; // 8 con trỏ đến 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)

```
1 void OctreeNode::insert(Body* newBody) {
2     // ===== CASE 1: External node va rong =====
3     // Chỉ cần gan body vào
4     if (isExternal && body == nullptr) {
5         body = newBody;
6         return;
7     }
8
9     // ===== CASE 2: External node DA CO body =====
10    // Phải chia thành 8 con, rồi chen cả 2 body vào
11    if (isExternal && body != nullptr) {
12        // Bước 1: Chia node hiện tại
13        subdivide();
14
15        // Bước 2: Chen body CU vào octant tương ứng
```

```

16     Body* oldBody = body;
17     body = nullptr; // Node nay khong con la la nua
18
19     int oldIndex = getOctantIndex(oldBody->pos);
20     children[oldIndex]->insert(oldBody); // DE QUY!
21
22     // Buoc 3: Chen body MOI vao octant tuong ung
23     int newIndex = getOctantIndex(newBody->pos);
24     children[newIndex]->insert(newBody); // DE QUY!
25
26     return;
27 }
28
29 // ===== CASE 3: Internal node (da co con) =====
30 // Chi can tim octant dung va chen de quy
31 int index = getOctantIndex(newBody->pos);
32
33 // Tao node con neu chua ton tai
34 if (children[index] == nullptr) {
35     Vec3 childCenter = getOctantCenter(index);
36     children[index] = new OctreeNode(childCenter, halfSize);
37 }
38
39 children[index]->insert(newBody); // DE QUY!
40 }

```

Listing 10.1: OctreeNode::insert()

### 10.1.3 Code: Compute Mass Distribution

```

1 void OctreeNode::computeMassDistribution() {
2     // ===== BASE CASE 1: Node rong =====
3     if (isExternal && body == nullptr) {
4         totalMass = 0.0;
5         centerOfMass = Vec3();
6         return;
7     }
8
9     // ===== BASE CASE 2: La co body =====
10    if (isExternal && body != nullptr) {
11        totalMass = body->mass;
12        centerOfMass = body->pos;
13        return;
14    }
15
16    // ===== RECURSIVE CASE: Node trong =====
17    // POST-ORDER: Tinh con truoc, cha sau
18
19    // Buoc 1: Goi de quy cho tat ca con
20    for (int i = 0; i < 8; i++) {
21        if (children[i] != nullptr) {
22            children[i]->computeMassDistribution();
23        }
24    }
25
26    // Buoc 2: Tong hop tu cac con
27    totalMass = 0.0;

```

```

28     Vec3 weightedSum = Vec3();
29
30     for (int i = 0; i < 8; i++) {
31         if (children[i] != nullptr && children[i]->totalMass > 0) {
32             double m = children[i]->totalMass;
33             Vec3 com = children[i]->centerOfMass;
34
35             weightedSum = weightedSum + com * m;
36             totalMass += m;
37         }
38     }
39
40     // Buoc 3: Tinh trong tam
41     if (totalMass > 0) {
42         centerOfMass = weightedSum / totalMass;
43     }
44 }

```

Listing 10.2: OctreeNode::computeMassDistribution()

### 10.1.4 Milestone: Debug Verification

```

1 // Trong main loop, sau khi build tree:
2 double totalBodyMass = 0.0;
3 for (const auto& b : bodies) {
4     totalBodyMass += b.mass;
5 }
6
7 root->computeMassDistribution();
8
9 std::cout << "Sum of body masses: " << totalBodyMass << std::endl;
10 std::cout << "Root totalMass: " << root->totalMass << std::endl;
11 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

```
1 Vec3 OctreeNode::calculateForce(const Body* target, double theta,
2                                 double G, double softening) {
3     // ===== CASE 1: Node rỗng =====
4     if (totalMass == 0.0) {
5         return Vec3(); // Không có lực
6     }
7
8     // ===== CASE 2: Là với 1 body =====
9     if (isExternal && body != nullptr) {
10        // Không tính lực lên chính nó
11        if (body == target) {
12            return Vec3();
13        }
14
15        // Tính lực trực tiếp
16        Vec3 r = body->pos - target->pos;
17        double distSq = r.x*r.x + r.y*r.y + r.z*r.z;
18        double dist = sqrt(distSq + softening * softening);
19
20        // a = G * m / r^2 * direction
21        double magnitude = G * body->mass / (dist * dist * dist);
22        return r * magnitude;
23    }
24 }
```

```

25 // ===== CASE 3: Node trong - BARNES-HUT LOGIC =====
26 Vec3 r = centerOfMass - target->pos;
27 double distSq = r.x*r.x + r.y*r.y + r.z*r.z;
28 double dist = sqrt(distSq);
29
30 // Kich thuoc node
31 double s = halfSize * 2.0;
32
33 // *** MULTIPOLE ACCEPTANCE CRITERION ***
34 // Neu s/d < theta, cum nay "du xa" de xap xi
35 if (dist > 0 && (s / dist) < theta) {
36     // Dung centerOfMass va totalMass de tinh
37     double distSoftened = sqrt(distSq + softening * softening);
38     double magnitude = G * totalMass / (distSoftened * distSoftened *
distSoftened);
39     return r * magnitude;
40 }
41
42 // Neu khong du xa, phai di sau vao cac con
43 Vec3 totalForce;
44 for (int i = 0; i < 8; i++) {
45     if (children[i] != nullptr) {
46         totalForce = totalForce + children[i]->calculateForce(
47             target, theta, G, softening
48         ); // DE QUY!
49     }
50 }
51
52 return totalForce;
53 }

```

Listing 11.1: OctreeNode::calculateForce()

### 11.1.3 Code: Main Loop with Barnes-Hut

```

1 // Constants
2 const double THETA = 0.5; // Accuracy parameter
3 const double G = 1.0; // Simplified gravity constant
4 const double SOFTENING = 0.1;
5 const double DT = 0.01; // Time step
6 const double UNIVERSE_SIZE = 100.0;
7
8 void simulationStep(std::vector<Body>& bodies) {
9     // ===== STEP 1: Rebuild Octree =====
10    // (Tree phai rebuild moi frame vi bodies di chuyen)
11
12    Vec3 center(0, 0, 0);
13    OctreeNode* root = new OctreeNode(center, UNIVERSE_SIZE);
14
15    for (auto& body : bodies) {
16        root->insert(&body);
17    }
18
19    // ===== STEP 2: Compute Mass Distribution =====
20    root->computeMassDistribution();
21
22    // ===== 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

```
1 #include <omp.h>
2
3 void simulationStep(std::vector<Body>& bodies) {
4     // ... Build tree ...
5
6     // *** PARALLEL FORCE CALCULATION ***
7     // Chi cần thêm 1 dòng này!
8     #pragma omp parallel for
9     for (size_t i = 0; i < bodies.size(); i++) {
10         bodies[i].acc = root->calculateForce(&bodies[i], THETA, G,
11         SOFTENING);
12     }
13
14     // Update cũng có thể parallel
15     #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

```
1 // Anh xa khoi luong sang mau sac (Quang pho sao)
2 // Do lun (Red Dwarf) -> Vang (Sun) -> Xanh khong lo (Blue Giant)
3 Color massToColor(double mass, double minMass, double maxMass) {
4     // Normalize mass to [0, 1]
5     double t = (mass - minMass) / (maxMass - minMass);
6     t = fmax(0.0, fmin(1.0, t)); // Clamp
7
8     unsigned char r, g, b;
9
10    if (t < 0.25) {
11        // Do dam -> Do nhut (Red Dwarf)
12        double local_t = t / 0.25;
13        r = 139 + (unsigned char)(116 * local_t); // 139 -> 255
14        g = 0 + (unsigned char)(69 * local_t);    // 0 -> 69
15        b = 0;
16    }
17    else if (t < 0.5) {
18        // Do nhut -> Vang (Orange to Yellow)
19        double local_t = (t - 0.25) / 0.25;
20        r = 255;
```

```

21     g = 69 + (unsigned char)(186 * local_t);    // 69 -> 255
22     b = 0 + (unsigned char)(100 * local_t);     // 0 -> 100
23 }
24 else if (t < 0.75) {
25     // Vang -> Trang (Yellow to White)
26     double local_t = (t - 0.5) / 0.25;
27     r = 255;
28     g = 255;
29     b = 100 + (unsigned char)(155 * local_t);   // 100 -> 255
30 }
31 else {
32     // Trang -> Xanh (White to Blue Giant)
33     double local_t = (t - 0.75) / 0.25;
34     r = 255 - (unsigned char)(100 * local_t);   // 255 -> 155
35     g = 255 - (unsigned char)(100 * local_t);   // 255 -> 155
36     b = 255;
37 }
38
39 return (Color){ r, g, b, 255 };
40 }

```

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

```
1 #pragma once
2 #include <cmath>
3
4 /**
5  * @brief 3D Vector class for physics calculations
6  *
7  * Tai sao dung struct thay vi class?
8  * -> Vec3 la "Plain Old Data" (POD), khong can encapsulation
9  * -> Performance tot hon khi copy
10 */
11 struct Vec3 {
12     double x, y, z;
13
14     // ===== CONSTRUCTORS =====
15
16     // Default constructor - khoi tao (0, 0, 0)
17     Vec3() : x(0.0), y(0.0), z(0.0) {}
18
19     // Parameterized constructor
20     Vec3(double x_, double y_, double z_) : x(x_), y(y_), z(z_) {}
21
22     // ===== OPERATOR OVERLOADING =====
23     // Cho phep viet code toan hoc tu nhien: v1 + v2, v * 2.0, etc.
24
25     // Vector + Vector
26     Vec3 operator+(const Vec3& other) const {
27         return Vec3(x + other.x, y + other.y, z + other.z);
28     }
29
30     // Vector - Vector
31     Vec3 operator-(const Vec3& other) const {
32         return Vec3(x - other.x, y - other.y, z - other.z);
33     }
34
35     // Vector * Scalar
36     Vec3 operator*(double scalar) const {
37         return Vec3(x * scalar, y * scalar, z * scalar);
38     }
39
40     // 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;

```

```

99     }
100
101     // Cross product (tich co huong)
102     Vec3 cross(const Vec3& other) const {
103         return Vec3(
104             y*other.z - z*other.y,
105             z*other.x - x*other.z,
106             x*other.y - y*other.x
107         );
108     }
109 };
110
111 // Scalar * Vector (cho phep viet: 2.0 * vec)
112 inline Vec3 operator*(double scalar, const Vec3& vec) {
113     return vec * scalar;
114 }

```

Listing 16.1: include/Vec3.hpp

## 16.2 Body.hpp - Celestial Body Structure

```

1  #pragma once
2  #include "Vec3.hpp"
3  #include "raylib.h"
4
5  /**
6   * @brief Represents a celestial body (star, planet, particle)
7   *
8   * Moi Body co:
9   * - Position (vi tri trong khong gian 3D)
10  * - Velocity (van toc - huong va toc do di chuyen)
11  * - Acceleration (gia toc - duoc tinh moi frame tu luc hap dan)
12  * - Mass (khoi luong - quyet dinh luc hap dan)
13  * - Color (mau sac hien thi)
14  */
15  struct Body {
16      // ===== PHYSICS PROPERTIES =====
17      Vec3 pos;          // Position (meters in simulation space)
18      Vec3 vel;          // Velocity (meters/second)
19      Vec3 acc;          // Acceleration (meters/second^2) - reset moi frame
20      double mass;       // Mass (kilograms)
21
22      // ===== VISUAL PROPERTIES =====
23      Color color;       // Raylib Color for rendering
24      float radius;      // Visual radius for drawing
25
26      // ===== CONSTRUCTORS =====
27
28      Body()
29          : pos(Vec3()), vel(Vec3()), acc(Vec3()),
30            mass(1.0), color(WHITE), radius(0.1f) {}
31
32      Body(Vec3 position, Vec3 velocity, double m)
33          : pos(position), vel(velocity), acc(Vec3()),
34            mass(m), color(WHITE), radius(0.1f) {}
35

```

```

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),

```

```

94         static_cast<float>(pos.y),
95         static_cast<float>(pos.z)
96     };
97
98     // Draw solid sphere
99     DrawSphere(rlPos, radius, color);
100 }
101
102 /**
103  * @brief Draw as a point (faster for many bodies)
104  */
105 void drawPoint() const {
106     Vector3 rlPos = {
107         static_cast<float>(pos.x),
108         static_cast<float>(pos.y),
109         static_cast<float>(pos.z)
110     };
111     DrawPoint3D(rlPos, color);
112 }
113 };

```

Listing 16.2: include/Body.hpp

## 16.3 OctreeNode.hpp - Complete Octree Implementation

```

1 #pragma once
2 #include "Vec3.hpp"
3 #include "Body.hpp"
4 #include <array>
5 #include <cmath>
6
7 /**
8  * @brief Octree Node for Barnes-Hut algorithm
9  *
10 * Octree la cau truc cay phan hoach khong gian 3D.
11 * Moi node dai dien cho mot hinh lap phuong (cube) trong khong gian.
12 *
13 * Co 2 loai node:
14 * 1. External (Leaf): Khong co con, chua toi da 1 body
15 * 2. Internal: Co it nhat 1 con, luu totalMass va centerOfMass
16 */
17 class OctreeNode {
18 public:
19     // ===== SPATIAL PROPERTIES =====
20     Vec3 center;           // Tam cua hinh lap phuong
21     double halfSize;      // Nua canh (bounds: center +/- halfSize)
22
23     // ===== PHYSICS PROPERTIES (for Barnes-Hut) =====
24     double totalMass;      // Tong khoi luong cua node va cac con
25     Vec3 centerOfMass;     // Trong tam khoi luong (weighted average)
26
27     // ===== TREE STRUCTURE =====
28     Body* body;            // Con tro den body (chi khi la
                             // External)
29     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 {

```

```

88     int index = 0;
89     if (point.x >= center.x) index |= 1;
90     if (point.y >= center.y) index |= 2;
91     if (point.z >= center.z) index |= 4;
92     return index;
93 }
94
95 /**
96  * @brief Calculate the center position of a child octant
97  * @param i Octant index (0-7)
98  * @return Center position of the child cube
99  */
100 Vec3 getOctantCenter(int i) const {
101     double offset = halfSize / 2.0;
102     return Vec3(
103         center.x + ((i & 1) ? offset : -offset),
104         center.y + ((i & 2) ? offset : -offset),
105         center.z + ((i & 4) ? offset : -offset)
106     );
107 }
108
109 /**
110  * @brief Check if a point is within this node's bounds
111  */
112 bool contains(const Vec3& point) const {
113     return (point.x >= center.x - halfSize && point.x < center.x +
114     halfSize) &&
115     (point.y >= center.y - halfSize && point.y < center.y +
116     halfSize) &&
117     (point.z >= center.z - halfSize && point.z < center.z +
118     halfSize);
119 }
120
121 // ===== CORE ALGORITHMS =====
122
123 /**
124  * @brief Subdivide this node into 8 children
125  *
126  * Duoc goi khi can chen body thu 2 vao External node.
127  */
128 void subdivide() {
129     double childSize = halfSize; // Moi con co kich thuoc = nua cha
130
131     for (int i = 0; i < 8; i++) {
132         Vec3 childCenter = getOctantCenter(i);
133         children[i] = new OctreeNode(childCenter, childSize);
134     }
135
136     isExternal = false; // Node nay tro thanh Internal
137 }
138
139 /**
140  * @brief Insert a body into the octree
141  * @param newBody Pointer to the body to insert
142  *
143  * THUAT TOAN INSERT (De quy):
144  *
145  * Case 1: External node, rong

```

```

143     *   -> Gan body vao, xong
144     *
145     * Case 2: External node, da co body
146     *   -> Chia thanh 8 con (subdivide)
147     *   -> Chen body cu vao octant tuong ung
148     *   -> Chen body moi vao octant tuong ung
149     *
150     * Case 3: Internal node (da co con)
151     *   -> Tim octant chua body moi
152     *   -> Tao node con neu chua ton tai
153     *   -> Goi insert de quy vao node con
154     */
155 void insert(Body* newBody) {
156     // Kiem tra body co nam trong bounds khong
157     if (!contains(newBody->pos)) {
158         // Body ngoai bounds - co the skip hoac expand tree
159         return;
160     }
161
162     // CASE 1: External node va rong
163     if (isExternal && body == nullptr) {
164         body = newBody;
165         return;
166     }
167
168     // CASE 2: External node DA CO body
169     if (isExternal && body != nullptr) {
170         // Luu body cu
171         Body* existingBody = body;
172         body = nullptr;
173
174         // Chia node
175         subdivide();
176
177         // Chen lai body cu
178         int existingIndex = getOctantIndex(existingBody->pos);
179         children[existingIndex]->insert(existingBody); // DE QUY
180
181         // Chen body moi
182         int newIndex = getOctantIndex(newBody->pos);
183         children[newIndex]->insert(newBody); // DE QUY
184
185         return;
186     }
187
188     // CASE 3: Internal node
189     int index = getOctantIndex(newBody->pos);
190
191     // Tao node con neu chua co
192     if (children[index] == nullptr) {
193         Vec3 childCenter = getOctantCenter(index);
194         children[index] = new OctreeNode(childCenter, halfSize);
195     }
196
197     children[index]->insert(newBody); // DE QUY
198 }
199
200 /**

```



```

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         //  $F = G * m_1 * m_2 / r^2$ 
290         //  $a = F / m_1 = G * m_2 / r^2$ 
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
311         double magnitude = G * totalMass / (distSoftened *
312         distSoftened * distSoftened);
313         return r * magnitude;
314     }
315 }

```

```

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  */

```

```

370     int maxDepth() const {
371         if (isExternal) return 1;
372
373         int maxChildDepth = 0;
374         for (int i = 0; i < 8; i++) {
375             if (children[i] != nullptr) {
376                 maxChildDepth = std::max(maxChildDepth, children[i]->
maxDepth());
377             }
378         }
379         return 1 + maxChildDepth;
380     }
381 };

```

Listing 16.3: include/OctreeNode.hpp

## 16.4 Galaxy.hpp - Galaxy Initialization

```

1  #pragma once
2  #include "Body.hpp"
3  #include <vector>
4  #include <random>
5  #include <cmath>
6
7  /**
8   * @brief Galaxy initialization utilities
9   *
10  * Cung cap cac ham de tao galaxy ban dau voi:
11  * - Phan bo khoi luong thuc te
12  * - Van toc xoay de galaxy on dinh
13  */
14
15 namespace Galaxy {
16
17 /**
18  * @brief Create a disk-shaped galaxy
19  * @param numBodies Number of stars
20  * @param diskRadius Radius of the disk
21  * @param diskHeight Thickness of the disk
22  * @param centralMass Mass of central black hole (optional)
23  * @return Vector of initialized bodies
24  */
25 std::vector<Body> createDiskGalaxy(int numBodies, double diskRadius,
26                                   double diskHeight, double centralMass
27                                   = 0.0) {
28     std::vector<Body> bodies;
29     bodies.reserve(numBodies);
30
31     // Random number generator
32     std::random_device rd;
33     std::mt19937 gen(rd());
34     std::uniform_real_distribution<double> angleDist(0.0, 2.0 * M_PI);
35     std::uniform_real_distribution<double> heightDist(-diskHeight/2,
diskHeight/2);
36     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 {
88     std::vector<Body> bodies;
89     bodies.reserve(numBodies);
90
91     std::random_device rd;

```

```

91     std::mt19937 gen(rd());
92     std::uniform_real_distribution<double> dist(-1.0, 1.0);
93     std::uniform_real_distribution<double> massDist(0.5, 2.0);
94
95     for (int i = 0; i < numBodies; i++) {
96         // Random point in sphere using rejection sampling
97         Vec3 pos;
98         do {
99             pos = Vec3(dist(gen), dist(gen), dist(gen));
100         } while (pos.magnitudeSq() > 1.0);
101
102         pos = pos * clusterRadius;
103
104         // Small random velocity
105         Vec3 vel(dist(gen) * 0.1, dist(gen) * 0.1, dist(gen) * 0.1);
106
107         double mass = massDist(gen);
108
109         bodies.push_back(Body(pos, vel, mass));
110     }
111
112     return bodies;
113 }
114
115 /**
116  * @brief Create two colliding galaxies
117  */
118 std::vector<Body> createCollidingGalaxies(int bodiesPerGalaxy, double
separation) {
119     // Galaxy 1 at origin
120     auto galaxy1 = createDiskGalaxy(bodiesPerGalaxy, 20.0, 2.0, 100.0);
121
122     // Galaxy 2 offset and moving toward galaxy 1
123     auto galaxy2 = createDiskGalaxy(bodiesPerGalaxy, 20.0, 2.0, 100.0);
124
125     // Offset galaxy 2
126     for (auto& body : galaxy2) {
127         body.pos.x += separation;
128         body.vel.x -= 0.5; // Moving toward galaxy 1
129     }
130
131     // Combine
132     galaxy1.insert(galaxy1.end(), galaxy2.begin(), galaxy2.end());
133     return galaxy1;
134 }
135
136 } // namespace Galaxy

```

Listing 16.4: include/Galaxy.hpp

## 16.5 main.cpp - Complete Simulation Loop

```

1 /**
2  * @file main.cpp
3  * @brief 3D N-Body Simulation using Barnes-Hut Algorithm
4  *

```

```

5  * Main entry point for the simulation.
6  * Handles:
7  * - Raylib window initialization
8  * - Main loop (Input -> Physics -> Render)
9  * - Camera controls
10 * - Performance monitoring
11 */
12
13 #include "raylib.h"
14 #include "Vec3.hpp"
15 #include "Body.hpp"
16 #include "OctreeNode.hpp"
17 #include "Galaxy.hpp"
18
19 #include <vector>
20 #include <iostream>
21 #include <chrono>
22
23 // ===== SIMULATION CONSTANTS =====
24
25 namespace Config {
26     // Window
27     constexpr int SCREEN_WIDTH = 1280;
28     constexpr int SCREEN_HEIGHT = 720;
29     constexpr int TARGET_FPS = 60;
30
31     // Simulation
32     constexpr int NUM_BODIES = 10000;
33     constexpr double UNIVERSE_SIZE = 200.0; // Half-size of simulation
34     space
35     constexpr double G = 1.0; // Gravitational constant (
36     scaled)
37     constexpr double SOFTENING = 0.5; // Softening parameter
38     constexpr double DT = 0.016; // Time step (~60 FPS)
39     constexpr double THETA = 0.5; // Barnes-Hut accuracy (0.5
40     is typical)
41
42     // Visual
43     constexpr bool DRAW_OCTREE = false; // Toggle octree
44     visualization
45 }
46
47 // ===== COLOR MAPPING =====
48
49 /**
50  * @brief Map star mass to stellar color (Red Dwarf -> Blue Giant)
51  */
52 Color massToColor(double mass, double minMass, double maxMass) {
53     double t = (mass - minMass) / (maxMass - minMass);
54     t = fmax(0.0, fmin(1.0, t)); // Clamp to [0, 1]
55
56     unsigned char r, g, b;
57
58     if (t < 0.33) {
59         // Red -> Orange
60         double lt = t / 0.33;
61         r = 200;
62         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) /
95         (maxMass - minMass));
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++) {

```

```

170         bodies[i].acc = root->calculateForce(
171             &bodies[i],
172             Config::THETA,
173             Config::G,
174             Config::SOFTENING
175         );
176     }
177
178     // Step 4: Update positions
179     for (auto& body : bodies) {
180         body.update(Config::DT);
181     }
182
183     // Cleanup
184     int treeNodes = root->countNodes();
185     int treeDepth = root->maxDepth();
186     delete root;
187     root = nullptr;
188
189     auto physicsEnd = std::chrono::high_resolution_clock::now();
190     physicsTime = std::chrono::duration<double, std::milli>(physicsEnd
- physicsStart).count();
191
192     // ===== RENDERING =====
193     BeginDrawing();
194     ClearBackground(Color{5, 5, 15, 255}); // Deep space blue
195
196     BeginMode3D(camera);
197     // Draw all bodies
198     for (const auto& body : bodies) {
199         body.draw();
200     }
201
202     // Optional: Draw octree wireframe
203     if (Config::DRAW_OCTREE && root != nullptr) {
204         root->drawWireframe(Color{0, 100, 0, 100});
205     }
206
207     // Reference grid
208     DrawGrid(20, 10.0f);
209     EndMode3D();
210
211     // UI Overlay
212     DrawRectangle(10, 10, 300, 150, Color{0, 0, 0, 150});
213     DrawText("N-Body Simulation (Barnes-Hut)", 20, 20, 20, WHITE);
214     DrawText(TextFormat("Bodies: %d", (int)bodies.size()), 20, 50,
16, GREEN);
215     DrawText(TextFormat("FPS: %d", GetFPS()), 20, 70, 16, GREEN);
216     DrawText(TextFormat("Physics: %.2f ms", physicsTime), 20, 90,
16, YELLOW);
217     DrawText(TextFormat("Tree Nodes: %d", treeNodes), 20, 110, 16,
GRAY);
218     DrawText(TextFormat("Tree Depth: %d", treeDepth), 20, 130, 16,
GRAY);
219
220     DrawText("Controls: Mouse to orbit, R to reset", 20,
SCREEN_HEIGHT - 30, 16, GRAY);
221     EndDrawing();

```

```
222         frameCount++;
223     }
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:

$$\vec{F}_{ij} = G \frac{m_i m_j}{r_{ij}^2} \hat{r}_{ij} \quad (17.1)$$

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:

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

#### 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 ( $\epsilon$ )

### 17.3.1 Vấn đề: Division by Zero

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

$$\vec{a} = G \frac{m}{r^2} \rightarrow \infty \quad (17.3)$$

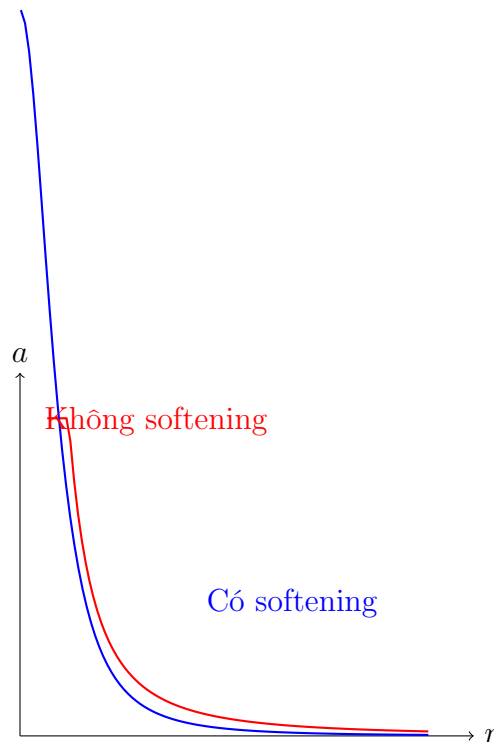
Điều này gây ra:

- 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  $\sqrt{r^2 + \epsilon^2}$ :

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



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

💡 Mẹo

Chọn giá trị  $\epsilon$ :

- 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

```

1 /**
2  * @brief Calculate gravitational acceleration with softening
3  * @param target The body receiving the force
4  * @param source The body exerting the force
5  * @param G Gravitational constant
6  * @param epsilon Softening parameter
7  * @return Acceleration vector
8  */
9 Vec3 calculateGravitationalAcceleration(
10     const Body& target,
11     const Body& source,
12     double G,
13     double epsilon
14 ) {
15     // Step 1: Compute displacement vector (from target to source)
16     Vec3 r = source.pos - target.pos;
17
18     // Step 2: Compute squared distance
19     double distSq = r.x*r.x + r.y*r.y + r.z*r.z;
20
21     // Step 3: Apply softening
22     // d = sqrt(r^2 + eps^2)
23     double softDistSq = distSq + epsilon * epsilon;
24     double softDist = std::sqrt(softDistSq);
25
26     // Step 4: Compute acceleration magnitude
27     // |a| = G * M / d^2
28     // But we need direction, so: a = G * M * r / d^3
29     // (r/d gives unit vector, then multiply by G*M/d^2)
30     double magnitude = G * source.mass / (softDist * softDist * softDist);
31
32     // Step 5: Return acceleration vector
33     return r * magnitude;
34 }

```

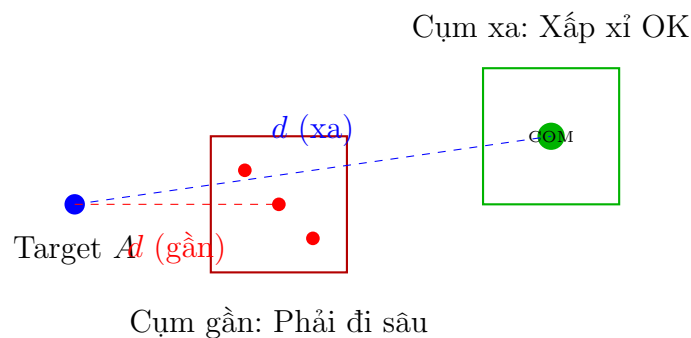
Listing 17.1: Complete Force Calculation

# 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
- $\theta$  = tham số độ chính xác (thường chọn 0.5)

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

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



## 18.3 Ảnh hưởng của $\theta$

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

$\theta$	Độ chính xác	Tốc độ	Ghi chú
0	Tuyệt đối (= Brute-force)	Chậm nhất	$O(N^2)$
0.3	Rất cao	Chậm	Scientific simulations
<b>0.5</b>	<b>Tốt</b>	<b>Nhanh</b>	<b>Khuyến nghị</b>
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

---

```

1: function CALCULATEFORCE(node, targetBody,  $\theta$ ,  $G$ ,  $\epsilon$ )
2:   if node is empty then
3:     return  $\vec{0}$ 
4:   end if
5:   if node is external AND contains a body then
6:     if body == targetBody then
7:       return  $\vec{0}$  ▷ Không tự hấp dẫn
8:     end if
9:     return DirectForce(targetBody, node.body,  $G$ ,  $\epsilon$ )
10:  end if
11:   $\vec{r} \leftarrow$  node.centerOfMass – targetBody.pos
12:   $d \leftarrow |\vec{r}|$ 
13:   $s \leftarrow$  node.size
14:  if  $s/d < \theta$  then ▷ Multipole criterion
15:    return ApproximateForce(targetBody, node.totalMass, node.centerOfMass,  $G$ ,  $\epsilon$ )
16:  else
17:     $\vec{F} \leftarrow \vec{0}$ 
18:    for each child in node.children do
19:       $\vec{F} \leftarrow \vec{F} +$  CALCULATEFORCE(child, targetBody,  $\theta$ ,  $G$ ,  $\epsilon$ )
20:    end for
21:    return  $\vec{F}$ 
22:  end if
23: 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 **hiệt 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

```
1 /**
2  * @brief Map star mass to realistic stellar color
3  * @param mass The star's mass
4  * @param minMass Minimum mass in simulation
5  * @param maxMass Maximum mass in simulation
6  * @return RGB Color
7  *
8  * THUẬT TOÁN:
9  * 1. Normalize mass to range [0, 1]
10 * 2. Map to color gradient:
11 * 0.0 -> Red (M-class, Red Dwarf)
12 * 0.25 -> Orange (K-class)
13 * 0.5 -> Yellow (G-class, like Sun)
14 * 0.75 -> White (F/A-class)
15 * 1.0 -> Blue (O/B-class, Blue Giant)
16 */
17 Color massToStellarColor(double mass, double minMass, double maxMass) {
18     // Step 1: Normalize to [0, 1]
19     double t = (mass - minMass) / (maxMass - minMass);
20     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 /**
```

```

2  * @brief Map mass to visual radius for rendering
3  *
4  * Note: Trong thực tế, bán kính sao  $\sim M^{0.8}$  (Main Sequence)
5  * Nhưng để visual đẹp hơn, ta dùng logarithmic scaling
6  */
7  float massToRadius(double mass, double minMass, double maxMass) {
8      const float MIN_RADIUS = 0.03f;
9      const float MAX_RADIUS = 0.3f;
10
11     // Logarithmic scaling (looks better visually)
12     double logMin = std::log(minMass);
13     double logMax = std::log(maxMass);
14     double logMass = std::log(mass);
15
16     double t = (logMass - logMin) / (logMax - logMin);
17     t = std::clamp(t, 0.0, 1.0);
18
19     return MIN_RADIUS + static_cast<float>(t) * (MAX_RADIUS - MIN_RADIUS);
20 }
21
22 // Trong main loop:
23 void applyVisuals(std::vector<Body>& bodies) {
24     // Tìm min/max mass
25     double minM = bodies[0].mass, maxM = bodies[0].mass;
26     for (const auto& b : bodies) {
27         minM = std::min(minM, b.mass);
28         maxM = std::max(maxM, b.mass);
29     }
30
31     // Apply
32     for (auto& b : bodies) {
33         b.color = massToStellarColor(b.mass, minM, maxM);
34         b.radius = massToRadius(b.mass, minM, maxM);
35     }
36 }

```

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 di xuyên qua nhau. Để thực tế hơn:

```
1 /**
2  * @brief Check and handle collisions between bodies
3  * @param bodies Vector of all bodies
4  *
5  * Note:  $O(N^2)$  - Trong production, dùng spatial hashing
6  */
7 void handleCollisions(std::vector<Body>& bodies) {
8     std::vector<int> toRemove;
9
10    for (size_t i = 0; i < bodies.size(); i++) {
11        for (size_t j = i + 1; j < bodies.size(); j++) {
12            double dist = bodies[i].pos.distance(bodies[j].pos);
13            double minDist = bodies[i].radius + bodies[j].radius;
14
15            if (dist < minDist) {
16                // INELASTIC COLLISION: Merge into body i
17                // Conservation of momentum:  $m_1*v_1 + m_2*v_2 = (m_1+m_2)*v_{final}$ 
18                double totalMass = bodies[i].mass + bodies[j].mass;
19
20                bodies[i].vel = (bodies[i].vel * bodies[i].mass +
21                                bodies[j].vel * bodies[j].mass) /
22                totalMass;
23
24                // Center of mass position
25                bodies[i].pos = (bodies[i].pos * bodies[i].mass +
26                                bodies[j].pos * bodies[j].mass) /
27                totalMass;
28
29                bodies[i].mass = totalMass;
30
31                toRemove.push_back(j);
32            }
33        }
34    }
35
36    // Remove merged bodies (reverse order to preserve indices)
37    std::sort(toRemove.begin(), toRemove.end(), std::greater<int>());
38    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

```
1 #include <omp.h>
2
3 // Chi cần thêm 1 dòng để song song hóa tính lực!
4 void updateWithOpenMP(std::vector<Body>& bodies, OctreeNode* root) {
5     // Parallel force calculation
6     // Mọi thread tính lực cho một nhóm bodies
7     #pragma omp parallel for schedule(dynamic)
8     for (size_t i = 0; i < bodies.size(); i++) {
9         bodies[i].acc = root->calculateForce(
10             &bodies[i], THETA, G, SOFTENING
11         );
12     }
13
14     // Parallel position update
15     #pragma omp parallel for
16     for (size_t i = 0; i < bodies.size(); i++) {
17         bodies[i].update(DT);
18     }
19 }
20
21 // 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)

```
1 // Lưu lịch sử vị trí
2 struct Body {
3     // ... existing members ...
4     std::deque<Vec3> trail; // Vị trí trước đó
5     static const int TRAIL_LENGTH = 50;
6
7     void updateTrail() {
8         trail.push_front(pos);
9         if (trail.size() > TRAIL_LENGTH) {
10             trail.pop_back();
11         }
12     }
13
14     void drawTrail() const {
15         for (size_t i = 1; i < trail.size(); i++) {
16             // Fade out theo vị trí
17             unsigned char alpha = 255 - (255 * i / TRAIL_LENGTH);
18             Color trailColor = {color.r, color.g, color.b, alpha};
19
20             Vector3 start = toRaylib(trail[i-1]);
21             Vector3 end = toRaylib(trail[i]);
22             DrawLine3D(start, end, trailColor);
23         }
24     }
25 };
```

Listing 22.1: Simple Trail Effect

# Chương 23

## Kết luận

### 23.1 Những gì nhóm sẽ đạt được

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

#### 1. Kiến thức DSA:

- Hiểu sâu về Con trỏ và Quản lý bộ nhớ động
- Thành thạo Hệ 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!**