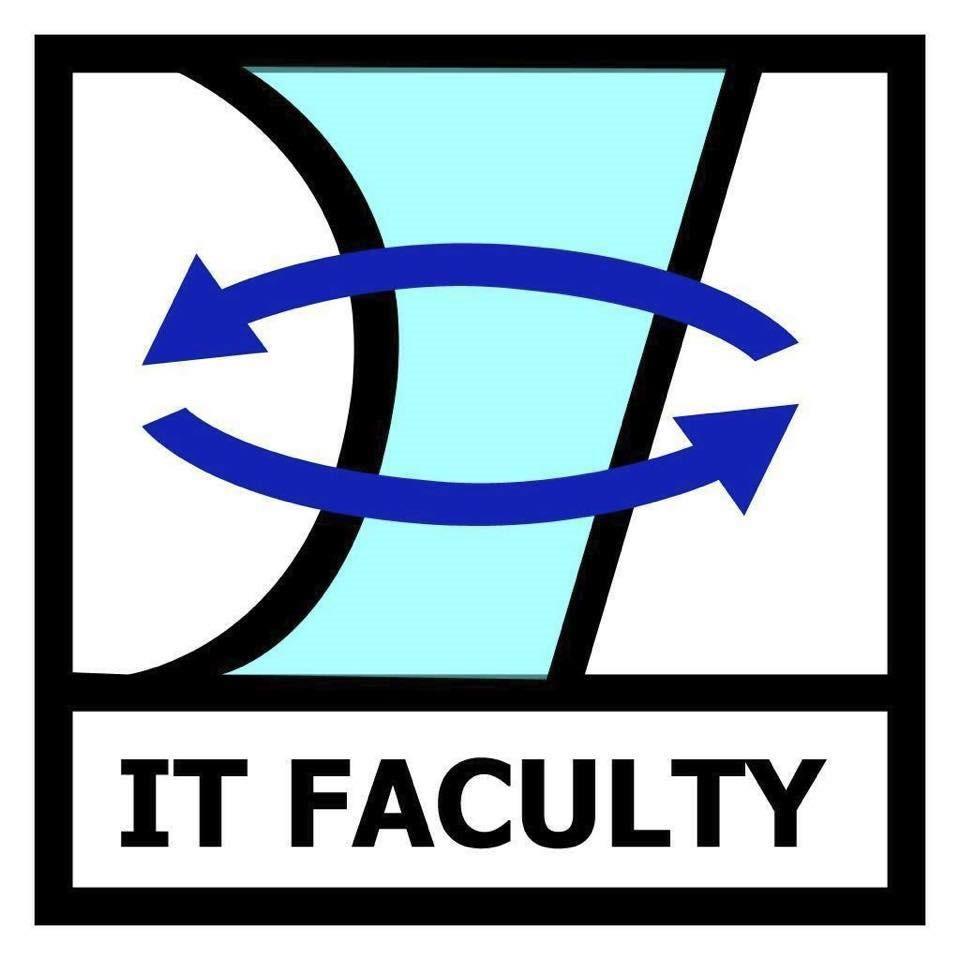
ĐẠI HỌC ĐÀ NẴNG

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA CÔNG NGHỆ THÔNG TIN

Tel. (+84.0236) 3736949, Fax. (84-511) 3842771

Website: http://dut.udn.vn/khoacntt, E-mail: cntt@dut.udn.vn



BÁO CÁO MÔN HỌC

ĐỒ HỌA MÁY TÍNH

ĐỀ TÀI :

TÌM HIỂU CẤU TRÚC DỮ LIỆU QUAD-EDGE

|  |  |
| --- | --- |
| HỌ TÊN SINH VIÊN | MÃ SINH VIÊN |
| Hoàng Đức An | 102220001 |
| Ngô Quốc Anh | 102220002 |
| Nguyễn Văn Bắc | 102220005 |

CBHD : PGS.TS. Nguyễn Tấn Khôi

Đà nẵng, 05/2024

MỤC LỤC

[CHƯƠNG 1: CƠ SỞ LÝ THUYẾT 2](#_Toc167046147)

[1.1. Ý tưởng 2](#_Toc167046148)

[1.2. Cơ sở lý thuyết 2](#_Toc167046149)

[CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG 5](#_Toc167046150)

[2.1. PHÁT BIỂU BÀI TOÁN 5](#_Toc167046151)

[2.2. PHÂN TÍCH CHỨC NĂNG 5](#_Toc167046152)

[2.2.1. Chức năng lưu trữ các điểm (Voronoi Cells) 5](#_Toc167046153)

[2.2.2. Chức năng tìm kiếm 5](#_Toc167046154)

[2.2.3. Công nghệ sử dụng 5](#_Toc167046155)

[2.3. THIẾT KẾ CƠ SỞ DỮ LIỆU 5](#_Toc167046156)

[2.4. TỔ CHỨC CHƯƠNG TRÌNH 5](#_Toc167046157)

[2.4.1. Tổ chức thư mục 5](#_Toc167046158)

[2.4.2. Tập tin “Quadedge.h” 6](#_Toc167046159)

[2.4.3. Tập tin “Voronoi.cpp” 6](#_Toc167046160)

[2.4.4. Tập tin “main.cpp” 8](#_Toc167046161)

[CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ 11](#_Toc167046162)

[3.1. KẾT QUẢ THỰC NGHIỆM 11](#_Toc167046163)

[3.1.1. Giao diện chương trình 11](#_Toc167046164)

[3.1.2. Kết quả thực thi 12](#_Toc167046165)

[3.2. NHẬN XÉT ĐÁNH GIÁ KẾT QUẢ 12](#_Toc167046166)

DANH SÁCH HÌNH ẢNH

[*Hình 1. Các thành phần quanh cạnh e* 2](#_Toc167046167)

[*Hình 2. Biểu diễn hướng theo chiều ngược chiều kim đồng hồ* 3](#_Toc167046168)

[*Hình 3. Biểu diễn hướng theo cùng chiều kim đồng hồ* 4](#_Toc167046169)

[*Hình 4. Giao diện chương trình* 11](#_Toc167046170)

[*Hình 5. Kết quả thực thi* 12](#_Toc167046171)

MỞ ĐẦU

1. Tổng quan về đề tài

Trong thời đại kỹ thuật ngày nay, việc hiểu và áp dụng cấu trúc dữ liệu là một phần quan trọng của việc giải quyết các vấn đề trong lĩnh vực khoa học máy tính và công nghệ thông tin. Trong báo cáo này, chúng tôi tập trung vào việc tìm hiểu và áp dụng cấu trúc dữ liệu "quad edge" - một cấu trúc mạnh mẽ và linh hoạt được sử dụng rộng rãi trong lý thuyết đồ thị và các ứng dụng liên quan.

Mục tiêu chính của báo cáo là cung cấp một cái nhìn tổng quan về cấu trúc dữ liệu quad edge và cách sử dụng nó để giải quyết các vấn đề phức tạp trong lý thuyết đồ thị. Chúng tôi cũng cung cấp một số ví dụ minh họa cụ thể và viết chương trình minh họa để giúp độc giả hiểu rõ hơn về cách quad edge có thể được triển khai trong thực tế.

Bằng cách hiểu sâu hơn về cấu trúc dữ liệu quad edge và khả năng áp dụng của nó, có cái nhìn sâu sắc hơn về lý thuyết đồ thị và có thêm công cụ mạnh mẽ để giải quyết các vấn đề trong lĩnh vực này.

2. Mục đích và ý nghĩa của đề tài

2.1. Mục đích

Về kiến thức: Cung cấp những kiến thức về cấu trúc dữ liệu Quad-Edge, từ đó có thể áp dụng vào các bài toán thực tế trong lĩnh vực đồ họa máy tính, xử lý hình ảnh, hệ thống thông tin địa lý (GIS), và các lĩnh vực liên quan khác.

Về kĩ năng: Nâng cao tư duy logic, giải quyết vấn đề, thiết kế thuật toán và lập trình thông qua việc, tìm hiểu và thực hiện dự án này.

2.2. Ý nghĩa

Góp phần củng cố kiến thức đã học trong học phần “Đồ họa máy tính”.

**3. Phương pháp thực hiện**

3.1. Bố cục của đồ án

Đồ án bao gồm các nội dung sau:

Mở đầu

Chương 1: Cơ sở lý thuyết

Chương 2: Phân tích thiết kế hệ thống

Chương 3: Triển khai và đánh giá kết quả

# CƠ SỞ LÝ THUYẾT

## Ý tưởng

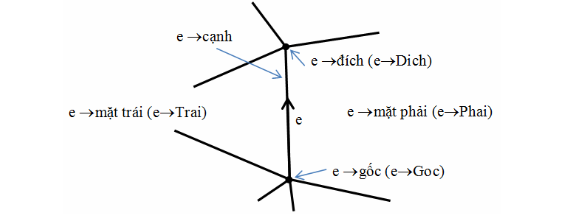
Ý tưởng cơ bản đằng sau cấu trúc quad-edge là thay vì lấy đỉnh hoặc mặt làm trọng tâm, thì nó tập trung vào các cạnh. Chúng ta dễ dàng nhận thấy rằng một cạnh, trong một mạng lưới đa giác đóng, nằm giữa chính xác hai mặt và chính xác hai đỉnh. Với mỗi cạnh của phép chia nhỏ luôn có hướng ứng với chiều các cạnh, ta dễ dàng tìm thấy các đỉnh lân cận, các cạnh, các mặt và cạnh đối xứng theo hướng ngược lại.

## Cơ sở lý thuyết

Cơ sở lý thuyết của quad-edge dựa trên một số quan sát quan trọng về đặc điểm của đỉnh đồ thị trong một mạng lưới đa giác. Dưới đây là một số điểm chính:

+ Cạnh duy nhất: Một cạnh duy nhất, trong một mạng lưới đa giác đóng, nằm giữa chính xác hai mặt và chính xác hai đỉnh. Điều này tạo ra một mối liên hệ mạnh mẽ giữa các đỉnh, cạnh và mặt trong mạng lưới.

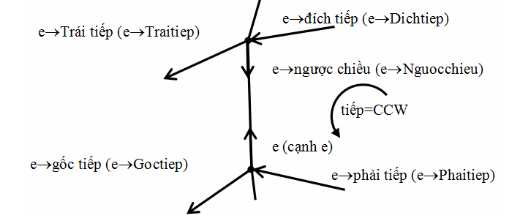
+ Biểu diễn cạnh: Mỗi mẫu tin về cạnh (edge record) chứa bốn cạnh e[0], e[1], e[2], e[3] mỗi cạnh e[r] với r=0,1,2,3 chứa hai trường Data và Next. Trường Data lưu trữ thông tin hình học, trường Next chứa tham chiếu cạnh tiếp theo.



*Hình 1. Các thành phần quanh cạnh e*

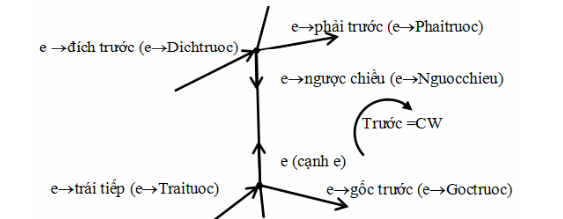
1. **e → cạnh**: Biểu thị cạnh được đề cập trong ngữ cảnh của sơ đồ.
2. **e → đích (e → Dich)**: Biểu thị đỉnh đích của cạnh e.
3. **e → gốc (e → Goc)**: Biểu thị đỉnh xuất phát của cạnh e.
4. **e → mặt phải (e → Phai)**: Biểu thị mặt bên phải liền kề với cạnh e trong ngữ cảnh của đồ thị phẳng.
5. **e → mặt trái (e → Trai)**: Biểu thị mặt bên trái liền kề với cạnh e trong ngữ cảnh của đồ thị phẳng.

+ Duyệt đồ thị:



*Hình 2. Biểu diễn hướng theo chiều ngược chiều kim đồng hồ*

1. **e → cạnh**: Chỉ ra cạnh được đề cập trong sơ đồ.
2. **e → đích tiếp (e → Dichtiep)**: Biểu thị đỉnh đích tiếp theo của cạnh e.
3. **e → gốc tiếp (e → Goctiep):**  Biểu thị đỉnh gốc tiếp theo của cạnh e.
4. **e → mặt phải tiếp (e → Phaitiep)**: Biểu thị mặt bên phải tiếp theo liền kề với cạnh e trong ngữ cảnh của đồ thị phẳng.
5. **e → mặt trái tiếp (e → Traitiep)**: Biểu thị mặt bên trái tiếp theo liền kề với cạnh e trong ngữ cảnh của đồ thị phẳng.
6. **e → ngược chiều (e → Nguocchieu)**: Biểu thị hướng ngược lại của cạnh e.
7. **tiếp = CCW**: "CCW" là viết tắt của "Counter ClockWise" tức là ngược chiều kim đồng hồ. Điều này chỉ ra hướng di chuyển tiếp theo của cạnh trong đồ thị phẳng.



*Hình 3. Biểu diễn hướng theo cùng chiều kim đồng h**ồ*

# PHÂN TÍCH THIẾT KẾ HỆ THỐNG

## PHÁT BIỂU BÀI TOÁN

Cho một tập hợp hữu hạn các điểm P={p1,p2,...,pn} trong một mặt phẳng, xây dựng sơ đồ Voronoi của tập hợp các điểm này bằng cách sử dụng cấu trúc dữ liệu Quad-Edge. Trong đó, Sơ đồ Voronoi (Voronoi Diagram) là một phân chia của mặt phẳng thành các vùng dựa trên khoảng cách đến một tập hợp các điểm cho trước. Mỗi vùng trong sơ đồ Voronoi được gọi là một cell Voronoi (Voronoi cell), và nó chứa tất cả các điểm trong mặt phẳng mà gần với điểm xác định vùng đó hơn bất kỳ điểm nào khác trong tập hợp.

## PHÂN TÍCH CHỨC NĂNG

### Chức năng lưu trữ các điểm (Voronoi Cells)

Các điểm được khởi tạo sẵn trong chương trình, sử dụng một kiểu dữ liệu đặc biệt do người dùng tự định nghĩa (Struct) là “Vertex”, sau đó được thêm vào class Quad-Edge để lưu trữ, và cuối cùng là tham số trong hàm khởi tạo VoronoiDiagram để lưu trữ và sử dụng

### Chức năng tìm kiếm

Khi người dùng click vào một địa điểm bất kì trong màn hình ứng dụng, chương trình sẽ xử lý dư liệu đầu vào là tọa độ của chuột khi được click, sau đó cho ra tọa độ của Voronoi cell gần điểm đó nhất và in ra màn hình.

### Công nghệ sử dụng

Thư viện đồ họa OpenGL và ngôn ngữ lập trình C/C++.

## THIẾT KẾ CƠ SỞ DỮ LIỆU

Cơ sở dữ liệu là một vector lưu trữ một tập các “Vertex” được khai báo sẵn trong chương trình.

## TỔ CHỨC CHƯƠNG TRÌNH

### Tổ chức thư mục

Dự án bao gồm 1 thư múc chứa mã nguồn gồm các file .h, .cpp.

### Tập tin “Quadedge.h”

class Vertex {

    private:

    double x, y;

    public:

    Vertex(double x, double y);

    double GetX() const;

    double GetY() const;

    double Vertex::distanceTo(const Vertex& other);

};

class QuadEdge {

public:

    Vertex\* origin;

    Vertex\* dest;

    QuadEdge\* twin;

    QuadEdge\* next;

    QuadEdge (Vertex\* origin, Vertex\* dest);

    void makeTwin();

};

class VoronoiDiagram {

private:

    std::vector<QuadEdge\*> edges;

public:

    const std::vector<QuadEdge\*>& getEdges() const;

    VoronoiDiagram(const std::vector<Vertex>& sites);

    ~VoronoiDiagram();

    Vertex findClosestSite(const Vertex& point);

};

### Tập tin “Voronoi.cpp”

Vertex::Vertex(double x, double y) : x(x), y(y) {}

double Vertex::GetX() const

{

    return x;

}

double Vertex::GetY() const

{

    return y;

}

double Vertex::distanceTo(const Vertex &other)

{

    return sqrt((x - other.x) \* (x - other.x) + (y - other.y) \* (y - other.y));

}

QuadEdge ::QuadEdge(Vertex \*origin, Vertex \*dest) : origin(origin), dest(dest), twin(nullptr), next(nullptr) {}

void QuadEdge::makeTwin()

{

    twin = new QuadEdge(new Vertex(dest->GetX(), dest->GetY()), new Vertex(origin->GetX(), origin->GetY()));

    twin->twin = this;

}

const std::vector<QuadEdge \*> &VoronoiDiagram::getEdges() const

{

    return edges;

}

VoronoiDiagram::VoronoiDiagram(const std::vector<Vertex> &sites)

{

    for (int i = 0; i < sites.size(); i++)

    {

        for (int j = i + 1; j < sites.size(); j++)

        {

            Vertex \*v1 = new Vertex(sites[i].GetX(), sites[i].GetY());

            Vertex \*v2 = new Vertex(sites[j].GetX(), sites[j].GetY());

            QuadEdge \*edge = new QuadEdge(v1, v2);

            edge->makeTwin();

            edges.push\_back(edge);

        }

    }

}

VoronoiDiagram::~VoronoiDiagram()

{

    for (QuadEdge \*edge : edges)

    {

        delete edge->origin;

        delete edge->dest;

        delete edge->twin->origin; // delete the origin of the twin edge

        delete edge->twin;         // delete the twin edge

        delete edge;               // delete the edge itself

    }

    edges.clear(); // clear the vector of edges

}

Vertex findClosestSite(const Vertex &point)

{

    double minDistance = 10000;

    Vertex closestSite(0, 0);

    for (QuadEdge \*edge : edges)

    {

        double dist1 = point.distanceTo(\*(edge->origin));

        double dist2 = point.distanceTo(\*(edge->dest));

        if (dist1 < minDistance)

        {

            minDistance = dist1;

            closestSite = \*(edge->origin);

        }

        if (dist2 < minDistance)

        {

            minDistance = dist2;

            closestSite = \*(edge->dest);

        }

    }

    return closestSite;

}

### Tập tin “main.cpp”

void initGL()

{

    glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Black and opaque

    glEnable(GL\_POINT\_SMOOTH);

    glEnable(GL\_BLEND);

    glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

    glPointSize(5.0);

    glLineWidth(1.0);

}

void display()

{

    glClear(GL\_COLOR\_BUFFER\_BIT);

    // Render the vertices

     glBegin(GL\_POINTS);

    glColor3f(0.0f, 1.0f, 1.0f); // Cyan color

    for (const Vertex& site : sites) {

        glVertex2d(site.x, site.y);

    }

    glEnd();

    glFlush();

}

void reshape(GLsizei width, GLsizei height)

{

   if (height == 0) height = 1;                // To prevent divide by 0

    GLfloat aspect = (GLfloat)width / (GLfloat)height;

    glViewport(0, 0, width, height);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    if (width >= height)

        gluOrtho2D(-3.0 \* aspect, 3.0 \* aspect, -3.0, 3.0);

    else

        gluOrtho2D(-3.0, 3.0, -3.0 / aspect, 3.0 / aspect);

}

Vertex normalizeVertex(const Vertex& v, double maxVal) {

    return Vertex(v.x / maxVal, v.y / maxVal);

}

void mouseClick(int button, int state, int x, int y) {

    if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) {

        int windowWidth = glutGet(GLUT\_WINDOW\_WIDTH);

        int windowHeight = glutGet(GLUT\_WINDOW\_HEIGHT);

        // Convert pixel coordinates to normalized OpenGL coordinates

        double normalizedX = (2.0 \* x / windowWidth) - 1.0;

        double normalizedY = 1.0 - (2.0 \* y / windowHeight);

        Vertex mouse(normalizedX, normalizedY);

        std::cout << "Mouse click at: (" << normalizedX << ", " << normalizedY << ")" << std::endl;

        Vertex closestSite = vd->findClosestSite(mouse);

        std::cout << "The closest site to point (" << normalizedX << ", " << normalizedY

                  << ") is at (" << closestSite.x << ", " << closestSite.y << ")." << std::endl;

    }

}

int main(int argc, char\*\* argv) {

    sites.push\_back(Vertex(-0.1, -0.1));

    sites.push\_back(Vertex(0.1, 0.1));

    sites.push\_back(Vertex(-0.5, 1));

    sites.push\_back(Vertex(-0.5, -1));

    sites.push\_back(Vertex(1, -0.5));

    sites.push\_back(Vertex(-1, -0.5));

    sites.push\_back(Vertex(0, 1));

    sites.push\_back(Vertex(0, -1));

    sites.push\_back(Vertex(1, 0));

    sites.push\_back(Vertex(-1, 0));

    sites.push\_back(Vertex(0.5, 1));

    sites.push\_back(Vertex(0.5, -1));

    sites.push\_back(Vertex(1, 0.5));

    sites.push\_back(Vertex(-1, 0.5));

    sites.push\_back(Vertex(-1, 1));

    sites.push\_back(Vertex(-1, -1));

    sites.push\_back(Vertex(1, 1));

    sites.push\_back(Vertex(1, -1));

     vd=new VoronoiDiagram(sites);

    glutInit(&argc, argv);

    glutInitWindowSize(840, 768);   // Set the window's initial width & height - non-square

    glutInitWindowPosition(100, 100); // Position the window's initial top-left corner

    glutCreateWindow("Voronoi Diagram");  // Create window with the given title

    glutDisplayFunc(display);       // Register callback handler for window re-paint event

    glutReshapeFunc(reshape);       // Register callback handler for window re-size event

    initGL();                       // Our own OpenGL initialization

    glutMouseFunc(mouseClick);

    glutMainLoop();                 // Enter the infinite event-processing loop

    delete vd;

    return 0;

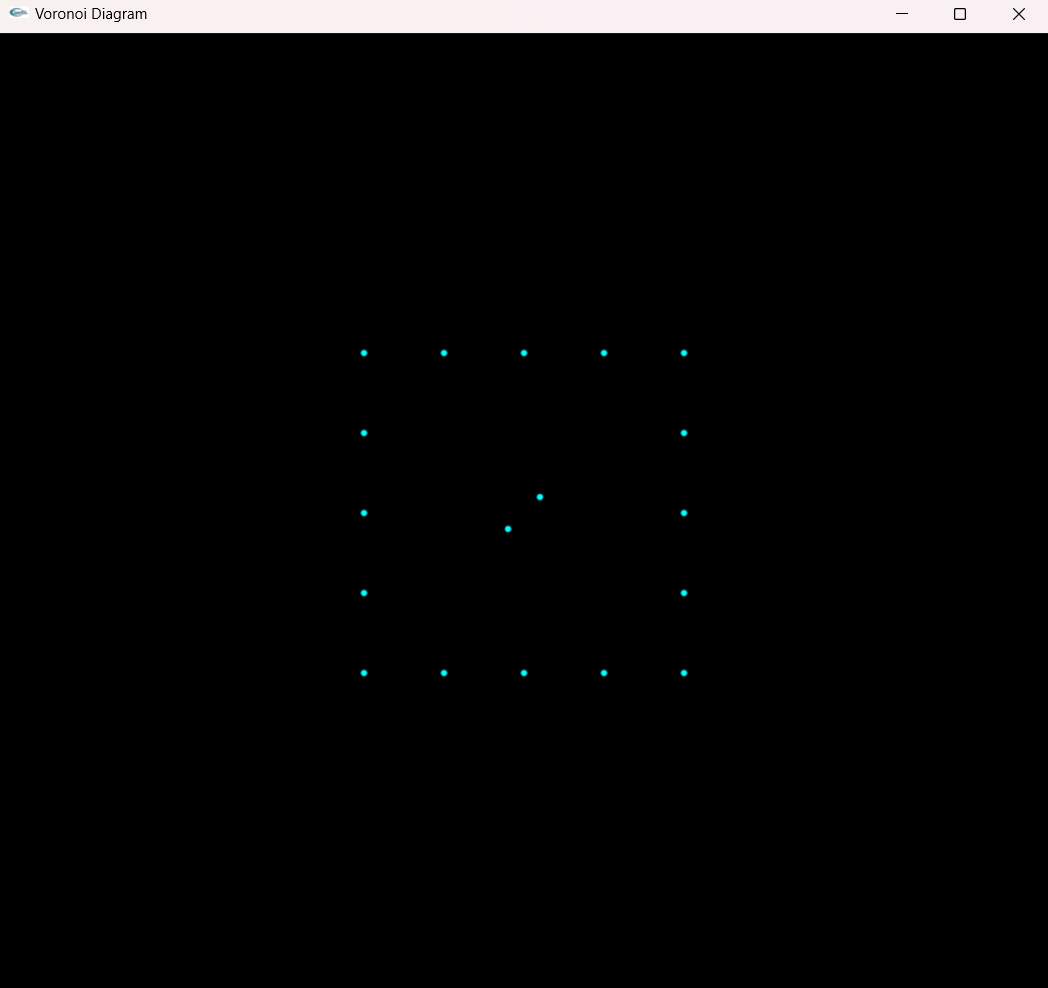
}

# TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

Chương này trình bày về kết quả triển khai hệ thống, cấu hình hệ thống và các thành phần chức năng. Kết quả được đánh giá thông qua các kịch bản thực nghiệm khác nhau nhằm thể hiện các ưu/nhược của giải pháp đề xuất.

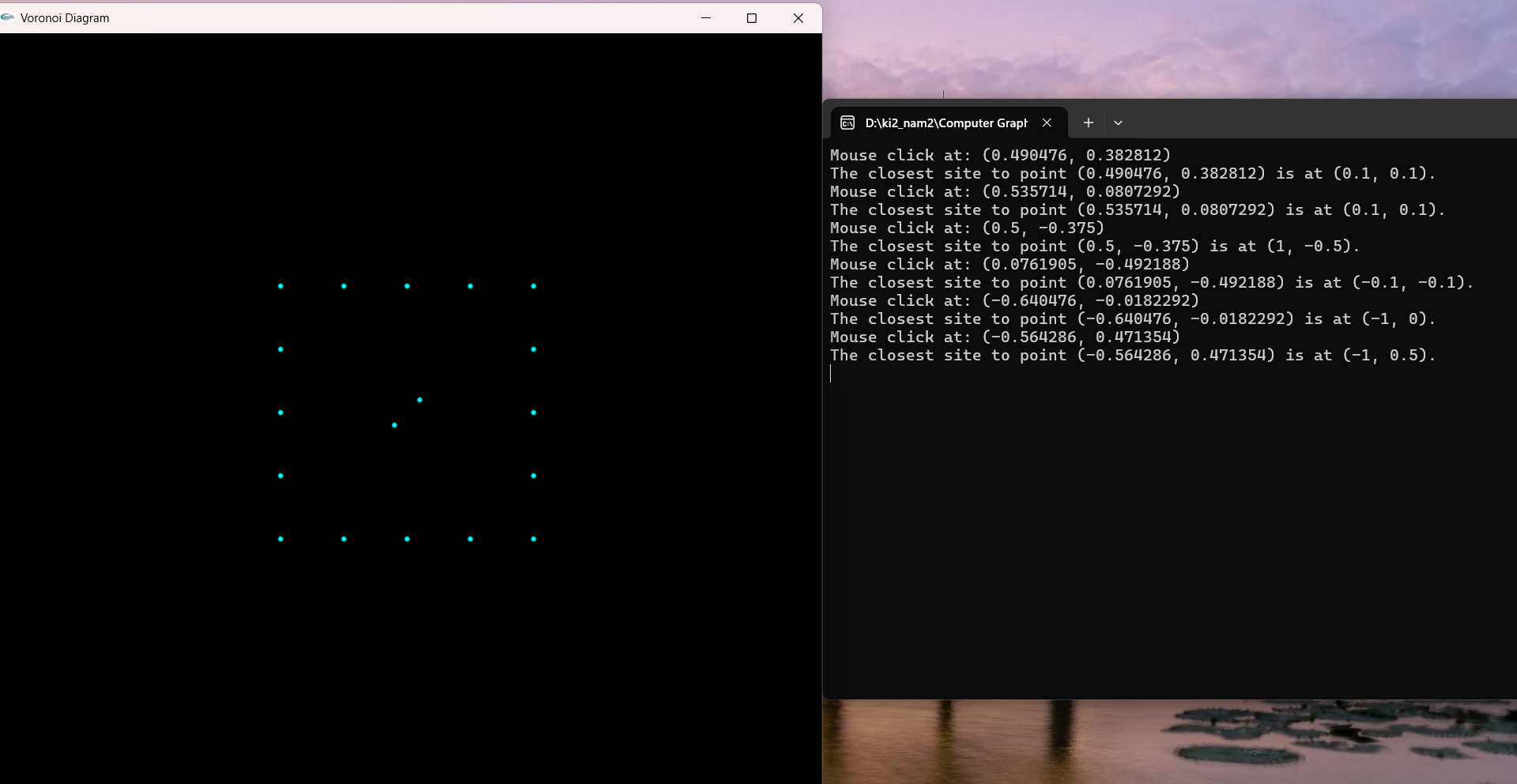
## KẾT QUẢ THỰC NGHIỆM

### Giao diện chương trình



*Hình 4. Giao diện chương trình*

### Kết quả thực thi



*Hình 5. Kết quả thực thi*

## NHẬN XÉT ĐÁNH GIÁ KẾT QUẢ

Qua kết quả thực nghiệm, tác giả có những nhận xét như sau:

* Vấn đề 1: Tọa độ các điểm Voronoi Cell trả về là khó nhận biết, chỉ người tạo nên ứng dụng mới biết rõ
* Vấn đề 2: Có áp dụng QuadEdge tuy nhiên chỉ áp dụng tính chất lưu trữ, tính chất duyệt qua các điểm một cách mạnh mẽ của QuadEdge vẫn chưa được áp dụng, còn phải thông qua bên thứ ba là class “Vertex”

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

1. KẾT QUẢ ĐẠT ĐƯỢC

Trong thời gian tìm hiểu, nghiên cứu cơ sở lý thuyết và triển khai ứng dụng công nghệ, đồ án đã đạt được những kết quả sau:

*Về mặt lý thuyết,* đồ án đã đạt được:

• Hiểu rõ cấu trúc dữ liệu Quad-Edge và cách thức hoạt động của nó.

• Nắm vững các khái niệm cơ bản về sơ đồ Voronoi và ứng dụng của nó trong các lĩnh vực khác nhau.

• Phân tích và chứng minh tính hiệu quả của cấu trúc dữ liệu Quad-Edge trong việc lưu trữ và truy xuất các thông tin hình học và topo của đồ thị planar.

Về mặt thực tiễn ứng dụng, đồ án đã đạt được:

• Phát triển thành công một ứng dụng xây dựng sơ đồ Voronoi sử dụng cấu trúc dữ liệu Quad-Edge.

• Triển khai ứng dụng đồ họa trực quan sử dụng OpenGL để hiển thị và tương tác với sơ đồ Voronoi.

• Kiểm chứng và đánh giá hiệu suất của thuật toán xây dựng sơ đồ Voronoi qua các trường hợp thử nghiệm.

Kết quả đóng góp của đồ án được thể hiện như sau:

* Phát triển một số thuật toán cho vấn đề xây dựng sơ đồ Voronoi:

• Đề xuất và triển khai một thuật toán tối ưu để xây dựng sơ đồ Voronoi từ một tập hợp các điểm trong mặt phẳng, sử dụng cấu trúc dữ liệu QuadEdge để giảm thiểu thời gian tính toán và bộ nhớ.

* Xây dựng được ứng dụng đồ họa trực quan:

• Phát triển một ứng dụng sử dụng OpenGL để hiển thị sơ đồ Voronoi và cho phép người dùng tương tác với sơ đồ bằng các sự kiện chuột.

• Chứng tỏ tính hiệu quả và trực quan của việc sử dụng cấu trúc dữ liệu QuadEdge trong việc quản lý và cập nhật sơ đồ Voronoi.

Tuy nhiên vẫn còn tồn tại các vấn đề như sau:

* Vấn đề triển khai cấu trúc dữ liệu Quad-Edge

• Vẫn còn phụ thuộc vào bên thứ ba là class “Vertex”, và hàm xử lý dữ liệu thuộc về lớp này.

* Vấn đề xử lý biên giới

• Việc xử lý các biên giới của sơ đồ Voronoi trong các trường hợp đặc biệt (như điểm nằm trên biên của không gian) cần được cải thiện để đảm bảo tính chính xác và ổn định của thuật toán.

2. KIẾN NGHỊ VÀ HƯỚNG PHÁT TRIỂN

Trong tương lai, có một số hướng phát triển tiềm năng để nâng cao hiệu quả và ứng dụng của cấu trúc dữ liệu Quad-Edge trong việc xây dựng sơ đồ Voronoi:

* Nâng cao thuật toán để xử lý tập dữ liệu lớn:

• Nghiên cứu và phát triển các thuật toán mới hoặc cải tiến thuật toán hiện tại để xử lý hiệu quả hơn với các tập dữ liệu lớn và phức tạp.

* Cải thiện xử lý biên giới và các trường hợp đặc biệt:

• Phát triển các phương pháp xử lý biên giới và các trường hợp đặc biệt để đảm bảo tính chính xác và ổn định của sơ đồ Voronoi.

* Ứng dụng trong lĩnh vực khác

• Khám phá các ứng dụng mới của sơ đồ Voronoi và cấu trúc dữ liệu QuadEdge trong các lĩnh vực như quy hoạch đô thị, mô phỏng môi trường, và phân tích dữ liệu không gian.

* Tích hợp với các công nghệ hiện đại

• Kết hợp với các công nghệ hiện đại như học máy và trí tuệ nhân tạo để tạo ra các ứng dụng thông minh và tự động hóa quá trình xây dựng và cập nhật sơ đồ Voronoi.

* Tối ưu hóa và mở rộng ứng dụng đồ họa

• Nâng cao giao diện đồ họa và khả năng tương tác của ứng dụng, giúp người dùng có thể thao tác và khám phá sơ đồ Voronoi một cách dễ dàng và hiệu quả hơn.TÀI LIỆU THAM KHẢO

Tiếng Việt

[1] VoronoiFortuneMesh, thầy Nguyễn Tấn Khôi

Internet

[2] [Quad-Edge Data Structure and Library (cmu.edu)](https://www.cs.cmu.edu/afs/andrew/scs/cs/15-463/2001/pub/src/a2/quadedge.html)

[3] [Quad-edge - Wikipedia](https://en.wikipedia.org/wiki/Quad-edge#:~:text=A%20quad%2Dedge%20data%20structure,earlier%20winged%20edge%20data%20structure.)